INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

HEAVEN
Eine hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array Datenbankmanagement Systeme

Bernd Reiner

Institut für Informatik
der Technischen Universität München

HEAVEN
Eine hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array Datenbankmanagement Systeme

Dipl.-Inf., Dipl.-Ing. (FH) Bernd Reiner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. P. P. Spies
Prüfer der Dissertation:	1. Univ.-Prof. R. Bayer, Ph.D., em.
	2. Univ.-Prof. A. Kemper, Ph.D.

Die Dissertation wurde am 27.01.2005 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 09.06.2005 angenommen.

Title of the Thesis:

HEAVEN

A Hierarchical Storage and Archive Environment for
Multidimensional Array Database Management Systems

Author:

Dipl.-Inf. Dipl.-Ing. Bernd Reiner

Keywords:

Multidimensional Array Data, Database Management Systems, Tertiary Storage,
Hierarchical Storage Management, Large Data, Archive Data, High Performance Computing

Abstract:

This thesis presents HEAVEN, a ***H**ierarchical Storag**e** and **A**rchive En**v**ironment for Multidim**e**nsional Array Database Ma**n**agement Systems*. HEAVEN provides a solution for an intelligent management of large-scale datasets held on tertiary storage systems. Scientific large-scale experiments and corresponding simulation programs often generate large amounts of multidimensional array data. Data volume may reach hundreds of terabytes (up to petabytes). Presently and in the near future the only practicable way for storing such large volumes of multidimensional data are tertiary storage systems. An identified major bottleneck today is the fast and efficient access to and evaluation of high performance computing results. We address the necessity of developing techniques for efficient retrieval of requested subsets of large datasets from mass storage devices. Furthermore, we show the benefit of managing large spatio-temporal data sets with database management systems. But commercial (multidimensional) database systems are optimized for performance with primary and secondary memory access. So tertiary storage memory is only supported in an insufficient way for storage or retrieval of multidimensional array data. This means database systems need a smart connection to tertiary storage systems with optimized access strategies. HEAVEN combines the advantages of both techniques, storing large amounts of data on tertiary storage media and optimizing data access for retrieval with multidimensional database management systems. We present techniques like the Super-Tile-Concept, Intra- and Inter-Super-Tile-Clustering, Query-Scheduling and Caching, keeping our main focus on reducing tertiary storage access time for multidimensional array data. Additionally we extended the multidimensional query language of the database system using a new developed concept called Object Framing. With this extension, users will no longer be restricted to performing range queries in the shape of multidimensional hyper cubes. Now they will be able to formulate range queries by indicating complex frames. Furthermore we dramatically reduce query execution time by using pre-calculated results of computational multidimensional operations. In summary, HEAVEN is optimized towards high performance computing and shows, as we have expected, a very good performance.

Sometimes our light goes out but is blown into flame by another human being. Each of us owes deepest thanks to those who have rekindled this light.
Albert Schweitzer (1875 – 1965)

Dank

Die Entstehung einer Dissertation, vom ersten Entwurf bis hin zur Vollendung, ist ein langwieriger Prozess, der ohne vielseitige Hilfe nur schwer zu bewerkstelligen ist. Aus diesem Grund möchte ich verschiedenen Personen danken, die diese Arbeit ermöglicht haben.

Besonderer Dank gebührt Herrn Prof. Rudolf Bayer, Ph.D. für die Überlassung des interessanten Themas und seine Unterstützung. Seine wertvollen Ratschläge waren immer sehr hilfreich. Außerdem sorgte er für ein angenehmes Arbeitsklima, was sich sehr positiv auswirkte. Danken möchte ich auch Herrn Prof. Alfons Kemper, Ph.D., der freundlicherweise das Zweitgutachten übernahm.

Wissenschaftliche Arbeiten werden wesentlich von dem Umfeld beeinflusst, in dem sie entstehen. Aus diesem Grund danke ich den Mitarbeitern des Lehrstuhls Datenbanken und Wissensbasen der Technischen Universität München. Besonders hervorheben möchte ich die „gute Seele des Lehrstuhls“ Evi Kollmann. Weiterhin danke ich meinen Kollegen Andreas Dehmel, Karl Hahn, Wolfgang Wohner und Robert Widhopf-Fenk von FORWISS (Bayerisches Forschungszentrum für Wissensbasierte System). Sie haben mir ein anregendes, informatives und unterstützendes Arbeitsumfeld geboten, das ich sehr vermissen werde. Mein ganz besonderer Dank gilt Karl Hahn, für seine Diskussionsbereitschaft und die inhaltlichen Beiträge. In unzähligen Stunden konnte ich mit ihm meine Ideen, von den ersten Fragmenten, bis hin zu ihrer Ausreifung besprechen.

Dankbar erwähnen möchte meine Eltern, Edeltraud und Prof. Dr. Ludwig Reiner, die mich immer unterstützt und bestätigt haben. Sie versuchten nie, mich in eine bestimmte Richtung zu lenken, sondern legten besonderen Wert auf meine persönlichen Interessen und Berufswünsche. Auf diesem Weg standen sie mir immer mit Rat und Tat zur Seite.

Bernd Reiner München, Januar 2004

Denken ist einfach, handeln schwierig, aber seine Gedanken in die Tat umzusetzen, ist das Allerschwierigste.
Johann Wolfgang von Goethe (1749 - 1832)

Inhalt

Every great and deep difficulty bears in itself its own solution. It forces us to change our thinking in order to find it.
Niels Henrik David Bohr (1885 – 1962)

Kapitel 1 Einleitung

HEAVEN[1] realisiert eine ***h**ierarchische Sp**e**icher- und **A**rchi**v**ierungsumgebung für multidim**e**nsionale Array Date**n**bank Management Systeme*. Zunächst werden im Kapitel 1.1 bestehende Probleme aufgezeigt. Kapitel 1.2 beschreibt das interdisziplinäre, europäische Projekt ESTEDI, in dessen Rahmen grundlegende Arbeiten dieser Dissertation entstanden. Die Ziele werden in Kapitel 1.3 erläutert. Abschließend folgt in Kapitel 1.4 ein Überblick über den Aufbau und die Struktur der Arbeit.

1.1 Problem

Viele Phänomene der Natur mit ihren Messwerten, können als Array-Daten mit entsprechender Dimensionalität modelliert werden: Dazu zählen Klimasimulationen, von Satelliten übertragene Daten der Atmosphäre, Berechnungen der Strömungsdynamik, Simulationen chemischer Prozesse oder Simulationen in der Kosmologie und der Genetik. Auch umfassende wissenschaftliche Experimente und Simulationen an Großrechnern, generieren multidimensionale Daten (Rasterdaten). Als eine gemeinsame Charakteristik gilt das immense Datenvolumen, das gespeichert werden muss. Der Datenumfang kann mehrere hundert Terabyte (10^{12} Byte), bis hin zu mehreren Petabyte (10^{15} Byte) erreichen. Bei CERN (*Conseil Europeen pour la Recherche Nucleaire*), einer europäischen Organisation für Nuklear-Forschung in der Schweiz, werden zum Beispiel jedes Jahr etwa 3 Petabyte neue Daten gespeichert. Ab dem Jahr 2007 beträgt der jährliche Datenzuwachs etwa 10 Petabyte [HS03]. Aufgrund des großen Volumens, können diese Daten nicht mehr auf Festplatten gespeichert werden. Typischerweise werden die Daten als Dateien in automatisierten *Tertiärspeichersystemen* (TS-System), mit bis zu tausenden von Magnetbändern, optischen oder magnetooptischen Medien gespeichert. Die mittleren Zugriffszeiten dieser Art von Speichertechnologien sind, verglichen mit Festplatten, relativ langsam. Trotzdem sind Tertiärspeicher bei riesigem Datenvolumen der gegenwärtige und auch zukünftige Entwicklungsstand, da Speichermedien wie Magnetbänder einen viel günstigeren €/GByte-Preis als Festplatten aufweisen (siehe Kapitel 2.2.3).

---

[1] HEAVEN → ***H**ierarchical Storag**e** and **A**rchive En**v**ironment for multidim**e**nsional Array Database Ma**n**agement Systems*.

Nicht das Speichern dieser großen Datenvolumen bereitet Probleme, sondern vor allem die Verarbeitung und der selektive Zugriff auf die Daten. Diese Probleme sind einerseits in der Verfügbarkeit von Hard- und Software begründet. Aufgrund der fehlenden Rechnerleistung können teilweise die großen Datenmengen, die zum Beispiel von einer Marssonde gesendet werden, erst in einigen Jahren vollständig ausgewertet werden. Andererseits liegen die Probleme vor allem in der unzureichenden Datenorganisation. Gewonnene Rohdaten werden direkt in der Ordnung des Generierungsprozesses als Datei auf *Tertiärspeichermedien* (TS-Medien) abgelegt. Diese „naive" Speicherung in der Prozessordnung, bietet keine Information zum Inhalt des Datenarchivs und ist nicht für einen direkten, anwendungsorientierten Datenzugriff geeignet. Zunächst müssen zur Beantwortung wissenschaftlicher Fragestellungen, verarbeitbare Daten mit hohem Aufwand aus den im Tertiärspeicher abgelegten Rohdaten-Dateien extrahiert werden. Dieser Vorgang wird auch als primäre Datenverarbeitung (Data Processing) bezeichnet. Voraussetzung hierfür ist, dass der genaue Speicherort im Tertiärspeicherarchiv und die Datenstruktur der Rohdaten bekannt sind. Bearbeitungszeiten von einigen Wochen, bis hin zu Monaten, sind dabei nicht ungewöhnlich [Laut02]. Das liegt auch daran, dass Wissenschaftler mit Dateien, Verzeichnissen und unterschiedlichen, oft komplexen Datenformaten umgehen und ihre Daten meist selbst verwalten müssen. Um ein beschleunigtes Datenretrieval zu erreichen, wird zumindest in Teilbereichen dazu übergegangen, eine Datenbank einzusetzen. So können Metadaten der gewonnenen Rohdaten verwaltet werden, um das Auffinden der Datensätze zu vereinfachen. Auch das manuelle Speichern einer geringen Anzahl ausgewählter multidimensionaler Datensätze als *Binary Large Objects* (BLOB) in der Datenbank, wird vereinzelt praktiziert. Allerdings wird die Datenbank nur als Speichermanager verwendet, um komplette Datensätze (BLOBs) an Applikationen zu liefern. Es können aber keine direkten, multidimensionalen Operationen auf dem Datenbank-Server durchgeführt werden. Diese Operationen werden von Analyse- und Simulationswerkzeugen übernommen. Wie bei der Speicherung als Datei auf Tertiärspeicher, können die BLOBs nur komplett angefasst und übertragen werden.

Dabei benötigen Wissenschaftler nur etwa ein bis zehn Prozent der angeforderten Daten für die weitere Verarbeitung [GKDT02, LT01]. Das liegt vor allem an der Art der Speicherung der Daten als Dateien auf Tertiärspeicher (bzw. BLOB) und den verwendeten mangelhaften Retrieval-Werkzeugen. Es ist derzeit nicht möglich, auf Teilbereiche einer Datei zuzugreifen, die zum Beispiel auf Magnetband gespeichert sind. Das bedeutet allerdings, dass eine Datei die kleinste Zugriffsgranularität darstellt. Also wird gegenwärtig immer eine komplette Datei vom Tertiärspeicher geladen und zum Nutzer übertragen. Hier wird dann von den Analyse- und Simulationswerkzeugen der benötigte Teilbereich der Daten zur Weiterverarbeitung verwendet und der Rest verworfen. Abbildung 1.1 zeigt eine Auswahl typischer Zugriffe (schwarze Bereiche) auf multidimensionale Daten. In diesem Beispiel gehen wir von Temperaturmesswerten aus, die durch Längengrad, Breitengrad und Höhe über Normal Null näher definiert werden.
Der linke Würfel in Abbildung 1.1 stellt eine Ausschnittsbildung dar. Eine typische Anfrage könnte wie folgt lauten: Ermittle alle Temperaturmesswerte von 48° 13' bis 48° 27' nördlicher Breite und von 11° 46' bis 11° 70' östlicher Länge in der Höhenstufe von 800 Meter bis

1000 Meter über Normal Null (kleiner schwarzer Würfel). Der Zugriff des mittleren Würfels setzt eine Dimension auf einen Wert fest, wodurch sich die Dimensionalität des Ergebnisses um eins reduziert. Eine denkbare Anfrage ist: Ermittle den kompletten Temperaturquerschnitt über die Höhen- und Längengrade bei 47° 80' nördlicher Breite. Beim rechten Beispiel der Abbildung 1.1 werden jeden Monat Temperaturmessungen durchgeführt und als separate Dateien gespeichert. So erhalten wir als weitere Dimension die Zeit. Dadurch können Anfragen einen Schnitt durch mehrere Dateien bedeuten. Eine Beispielanfrage lautet: Bestimme die Verteilung der Durchschnittstemperatur von Januar 2003 bis Juni 2003 auf einer Höhe von 800 Meter über Normal Null. Diese typischen Anfragen zeigen sehr deutlich, dass zur Berechnung der Ergebnismenge eine viel geringere Datenmenge notwendig ist, als in Wirklichkeit vom TS-Medium geladen wird. Gerade Berechnungen quer durch mehrere Dateien stoßen aufgrund des Datenvolumens rasch an die Grenzen der vorhandenen Hardware, wie z.B. der Hauptspeicher (*Main Memory*). Das bedeutet, dass ein Großteil interessanter und wichtiger Simulationen und Analysen zur Zeit nicht durchgeführt werden können.

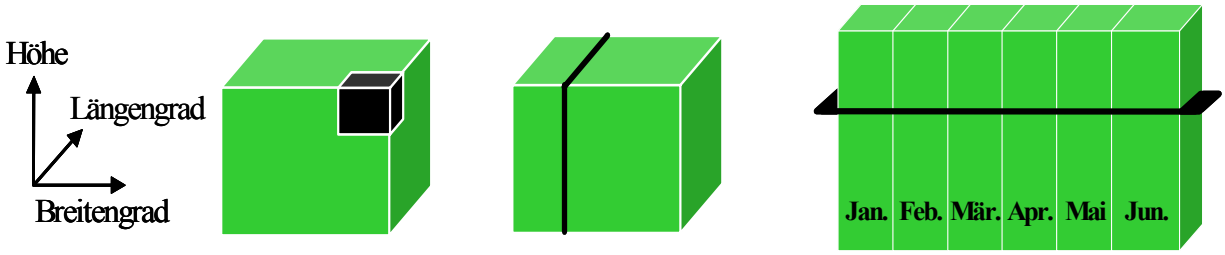


Abbildung 1.1: Eine Auswahl typischer Zugriffe auf multidimensionale Daten.

Durch die großen Datenvolumina wird das Netzwerk sehr schnell überlastet. Aus diesem Grund ist es auch üblich, Datenmedien an unterschiedliche Standorte per Post zu verschicken: „Unterschätze nie die Bandbreite eines mit Magnetbändern voll beladenen LKW, der da über die Autobahn braust" [Tane03]. Durch die Möglichkeit, auf Teilbereiche von Objekten zuzugreifen, könnte eine große Netzwerkbelastung vermieden werden und die Datenübertragung via Netzwerk bzw. Internet wird in diesem Anwendungsbereich wieder interessant. Die Übertragung von 200 GByte reiner Nutzdaten (10 % der Objektdaten) über eine schnelle Internetverbindung mit 8 MBit/s (asymmetrische DSL-Verbindung) dauert etwa eine Stunde. Müssen jedoch die kompletten Objekte (2 TByte) übertragen werden, wartet der Anwender 10 Stunden. Bei dieser Betrachtung wird eine optimale Netzwerkauslastung, ohne Netzwerkschwankungen angenommen.

Die Tertiärspeicherschicht in einem hierarchischen Speichersystem ist charakterisiert durch sehr große Datenvolumina und sehr hohe Anfragezeiten für wahlfreie Zugriffe (Random Access). Dies begründet sich durch die Verwendung zahlreicher günstiger Wechselmedien (z.B. Magnetbänder), die sich eine kleine Anzahl an teuren Schreib-/Lesestationen (Laufwerke) und Robotermechanismen teilen. Die hohe Zugriffszeit wird typischerweise dominiert

durch die Medienwechselzeit (12 s – 40 s) und der mittleren Zugriffszeit[2] (27 s – 95 s). Festplatten sind bei der mittleren Zugriffszeit um einen Faktor 10^3 bis 10^4 schneller. Die Transferrate eines TS-Systems ist dagegen sehr gut und nur um etwa den Faktor Zwei langsamer als die Transferraten von Festplatten. Wahlfreier Zugriff auf Daten, die auf Tertiärspeicher abgespeichert wurden, kann unakzeptable Verzögerung mit sich bringen, da Medien während des Datenzugriffes in den Schreib-/Lesestationen gewechselt werden müssen. Die Notwendigkeit des Medienwechsels wird durch die Datenverteilung auf den Bändern diktiert. Vernünftiges Platzieren der Daten auf TS-Medien ist aus diesen Gründen kritisch und kann sehr stark die Performance des Gesamtsystems beeinflussen. In den meisten Fällen werden die gewonnenen Rohdaten einfach in Generierungsordnung auf den TS-Medien abgespeichert. Gerade eine Optimierung in diesem Bereich kann die Zugriffszeiten drastisch reduzieren. Das Problem der optimalen Platzierung von Daten auf Magnetbändern kann auf zwei Teilprobleme herab gebrochen werden. 1) Eine geschickte Verteilung der Daten auf unterschiedliche Medien kann häufige Medienwechsel vermeiden. 2) Eine optimale Anordnung der Daten innerhalb eines Mediums reduziert Spul- und Suchoperationen. Natürlich müssen Anfragen entsprechend dieser Datenverteilung optimiert werden, um eine Reduzierung der Zugriffszeiten zu erreichen. Typische Zugriffsmuster der Anwender können Einfluss auf die Organisation und Verteilung der Daten auf, bzw. zwischen TS-Medien nehmen.

Um den hier erwähnten Defiziten zu begegnen, ist der Einsatz eines multidimensionalen Array-Datenbanksystems ein viel versprechender Ansatz. Die Vorteile der Verwendung eines multidimensionalen Array-Datenbanksystems zur Speicherung und Verwaltung der Daten liegen auf der Hand:

- Vermeidung von Redundanz und Inkonsistenz
- Zugriffskontrolle und Zugriffsbeschränkungen
- Mehrbenutzerbetrieb
- Zentrale Verwaltung der Daten
- Datenabstraktion und Datenunabhängigkeit
- Transaktionsverwaltung (ACID-Paradigma) und Recovery
- Multidimensionale Anfragesprache

Einziges Problem, das gegen den Einsatz von Datenbanksystemen spricht, ist das enorme Datenvolumen von mehreren hundert Terabyte bis hin zu mehreren Petabyte. Heute sind Gigabyte-Datenbanken üblich, Terabyte-Datenbanken die aktuelle Herausforderung und Petabyte-Datenbanken ein wichtiges Forschungsgebiet. Zum Vergleich: Bei heutiger Festplattengtechnologie (120 GByte) würde ein Petabyte an Daten über 8.700 Festplatten belegen. Ein nahe liegender Gedanke ist daher, die Vorteile der Datenverwaltung durch ein Datenbanksystem und die Datenhaltung auf TS-Medien zu verbinden.

[2] Mittlere Zugriffszeit bei TS-Systemen ist die durchschnittlich benötigte Zeit für die Lokalisierung einer Datei und die Positionierung des Lesekopfes auf einem Tertiärspeicher-Medium. Bei Magnetbändern ist die mittlere Zugriffszeit definiert als die Zeit, die benötigt wird, das Band von Beginn bis zur Mitte zu positionieren.

Zentrales Thema dieser Arbeit war daher die Realisierung einer effizienten, direkten und automatisierten Anbindung von TS-Systemen an ein multidimensionales Array Datenbanksystem. Dadurch wird eine transparente, hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array Datenbank Management Systeme mit nahezu unlimitiertem Speichervolumen geschaffen. Diese automatisierte und direkte Kopplung eines multidimensionalen Array-Datenbanksystems mit TS-Systemen wurde erstmals mit HEAVEN realisiert. Dabei wurde speziell auf optimierte Datenzugriffe auf Festplatte und TS-Medien wert gelegt.

Die Verschmelzung eines multidimensionalen Datenbanksystems mit TS-Systemen bringt sehr viele Vorteile gegenüber der bisherigen, klassischen Verarbeitungsmethode im Umfeld der Großrechenzentren (siehe auch Kapitel 1.3). Mit HEAVEN werden alle hier angesprochenen Defizite systematisch beseitigt. Grundlegende Arbeiten entstanden im Rahmen des europäischen Projektes ESTEDI, das kurz vorgestellt wird.

1.2 Projekt ESTEDI

Das interdisziplinäre Projekt ESTEDI wurde unter der Antragsnummer IST-1999-1109 von der Europäischen Kommission während der Laufzeit vom 1. Februar 2000 bis 30. April 2003 gefördert. ESTEDI steht für *European Spatio-Temporal Data Infrastructure for High-Performance Computing*. Erklärtes Ziel war es, für Hochleistungsrechenzentren (*High-Performance Computing*, HPC), eine flexible und performante Infrastruktur für sehr große Datenvolumen zu entwickeln und zu etablieren. Dabei sollte vor allem der bestehende Flaschenhals im Bereich des Retrieval und der Evaluierung dieser Rasterdaten (Array-Daten) beseitigt werden. Erreicht wurde dies durch Einsatz eines multidimensionalen *Datenbank Management Systems* (DBMS) mit direkter, automatisierter Anbindung an TS-Systeme für die Verwaltung der riesigen Datenmengen (hunderte TByte bis mehrere PByte). Dabei wurde nicht nur konzeptuelle Arbeit geleistet, sondern unter konsequenter Einbeziehung der Nutzer auf die Praxistauglichkeit der entwickelten Systeme geachtet. Auch die in Kapitel 1.1 aufgeführten Defizite sind bewusst behandelt und Lösungskonzepte erarbeitet worden. Für den Nutzer dieser Infrastruktur bedeutet das eine Verbesserung der Servicegüte in Bezug auf Performance, Funktionalität und Verwendbarkeit.

Im Projekt ESTEDI haben sich bedeutende Supercomputing-Zentren aus Europa und Russland zusammengefunden, um mit zwei Datenbankspezialisten diese Probleme zu lösen. Die ESTEDI Projektpartner sind im Einzelnen:

- *Deutsches Luft- und Raumfahrtzentrum* (DLR) aus Oberpfaffenhofen, Deutschland, mit dem *Deutschen Fernerkundungsdatenzentrum* (DFD), Europas größtem Satellitenbilderarchiv; Archivgröße: 1 PByte.
- *Deutsches Klimarechenzentrum* (DKRZ), bzw. *Max-Planck-Institut für Meteorologie* (MPI-Met) aus Hamburg, Deutschland, mit dem Forschungsschwerpunkt Klimasimulation und Wettervorhersagen; Archivgröße: 4 PByte.

- *Council of Central Laboratory of the Research Councils* (CCLRC) aus Chilton, England, mit dem Forschungsschwerpunkt Klimasimulation.
- *Interuniversity Consortium of the North Eastern Italy for Automatic Computing* (Cineca) in Bologna, Italien, Rechenzentrum mit dem Forschungsgebiet kosmologische Simulationen und der visuellen Aufbereitung; Archivgröße: 900 TByte.
- *Institute for High-Performance Computing and Databases* (IHPC&DB) aus St. Petersburg, Russland, mit dem Forschungsgebiet der Analyse und Simulation im Bereich der Genetik.
- *Swiss Center for Scientific Computing* (CSCS) aus Manno in der Schweiz, mit dem Forschungsschwerpunkt der Analyse und Simulation chemischer Prozesse; Archivgröße: 70 TByte.
- *Numerical Mechanics Application International* S.A. (Numeca International), aus Brüssel in Belgien, ein industrieller Partner mit dem Schwerpunkt Visualisierung im Bereich Strömungsdynamik.
- University of Surrey aus Guildford, England, mit dem Schwerpunkt Strömungsdynamik bei Flüssigkeiten, Gasen und Verbrennungsprozessen und Vertreter des Forschungsverbundes ERCOFTAC (*European Research Community on Flow, Turbulence and Combustion*).
- Active Knowledge GmbH (jetzt rasdaman GmbH) aus München, Deutschland, Datenbankspezialist und Vertreiber des multidimensionalen Array-Datenbanksystems RasDaMan (*Raster Data Management*).
- *Bayerisches Forschungszentrum für Wissensbasierte Systeme* (FORWISS; Arbeitgeber des Autors) aus Garching bei München, Deutschland. Datenbankspezialist im Bereich multidimensionaler Datenbanken. Neben der Entwicklungsarbeit im Bereich der TS-Anbindung und der Parallelisierung war FORWISS zuständig für die Projektkoordination, die Finanzabwicklung und die technische Leitung.

Die Anwendungsgebiete der ESTEDI-Projektpartner reichen von E-Commerce mit Satellitendaten, Data-Mining bei Klimamodellen und genetischen Daten, bis hin zu chemischen Prozesssimulationen, Analysen im Bereich der Strömungsdynamik und kosmologischen Simulationen.
Abbildung 1.2 visualisiert Beispieldaten zweier ESTEDI Projektpartner. Die linke Seite zeigt ein Satellitenbild[3] von Europa, das vom Deutschen Luft- und Raumfahrtzentrum über ihre, im Rahmen des ESTEDI Projektes entstandene Internetplattform EOWEB (*Earth Observation WEB*, http://eoweb.dlr.de) zum Verkauf angeboten wird. Dabei handelt es sich um einen Vegetationsindex, wobei die „Aktivität“ der Vegetation von braun nach grün zunimmt. Bei der rechten Darstellung handelt es sich um eine Simulation[4] zur Klimaerwärmung des Deutschen Klimarechenzentrums. Zu erkennen ist die atmosphärische Temperaturverteilung der Erde, zwei Meter über der Oberfläche, in einer Zeitspanne von 10 Jahren (2000 – 2009). Die

[3] Abgeleitet aus NOAA-AVHRR (National Oceanic and Atmonsperic Administration – Advanced Very High Resolution Radiation) Daten, Juli 2002.
[4] Simulation des Treibhauseffektes entsprechend dem IS92a Szenarium des IPCC'92 (*Intergovernmental Panel on Climate Change*) mit einem jährlichen Anstieg der Konzentration des Treibhausgases CO_2 um 1 %.

Erdoberfläche befindet sich auf der linken Vorderseite des Datenwürfels (Längen- und Breitengrad). In den mittleren Breitengraden ist der Äquator mit höheren Temperaturen (rötlich gefärbt) zu sehen. Die Regionen des Nord- und Südpols weisen kältere Regionen auf, die blau dargestellt sind. Jahreszeitliche periodische Schwankungen über 10 Jahre können gut in der Zeitdimension erkannt werden.

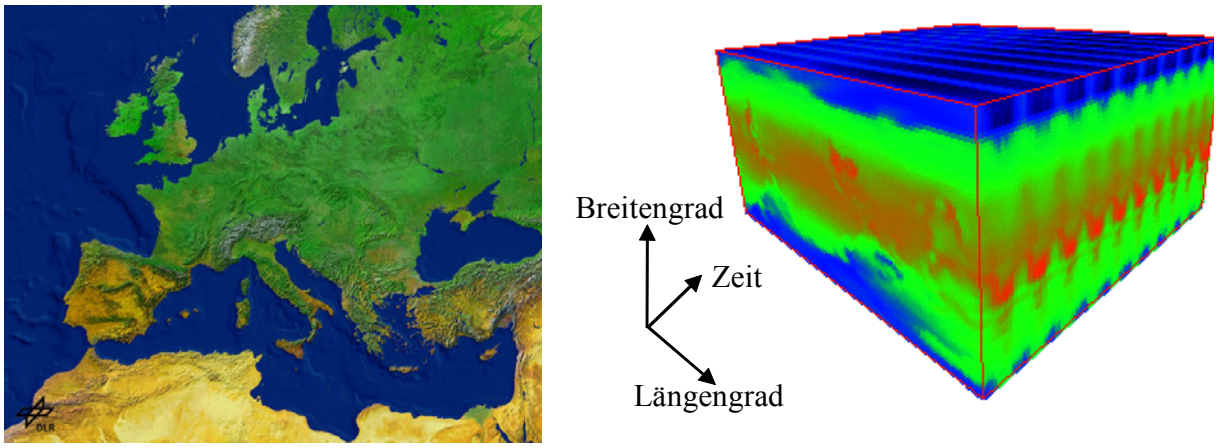


Abbildung 1.2: Links: Satellitenbild Europa (DLR); Rechts: Klimamodell der Erde (DKRZ).

Nach der Beschreibung einiger Anwendungsgebiete, folgt die Projektphilosophie von ESTEDI. Es wurde zunächst in einem ersten Schritt unter Führung von ERCOFTAC, eine umfassende Anforderungsanalyse erstellt. Dazu ist eine *User Interest Group* (UIG) aus potentiellen Nutzern gebildet worden. In dieser UIG wurden nicht nur die initialen Anforderungen aufgenommen, sondern auch die Projektergebnisse laufend von Anwendern bewertet. Diese Bewertungen flossen direkt in die weitere Entwicklung ein.
Die prototypische Realisierung der ESTEDI-Plattform erfolgte durch die beiden Datenbank Spezialisten auf der Basis des multidimensionalen Array-Datenbanksystems RasDaMan. Zur Evaluierung entwickelte jeder HPC-Partner für sein Spezialgebiet eine Applikation auf der generischen Plattform, die sich im Alltagseinsatz beim Kunden und den UIG-Mitgliedern bewähren musste. Die Referenzarchitektur der ESTEDI-Infrastruktur für HPC-Applikationen mit den logischen Zugriffspfaden für Daten und Metadaten ist in Abbildung 1.3 dargestellt.
Der obere Bereich der Abbildung zeigt die HPC-Applikation, die über eine deklarative Anfragesprache, bzw. eine Java/C++ API (*Application Programming Interface*), auf das multidimensionale DBMS RasDaMan zugreift. Die applikationsspezifischen Metadaten werden direkt in dem Basis-RDBMS (*Relationales DBMS*) gespeichert. Die von HPC-Datenerzeugern generierten multidimensionalen Daten werden in RasDaMan eingefügt. RasDaMan selbst nutzt ein konventionelles RDBMS, wie z.B. Oracle oder IBM/DB2, als Speicher- und Transaktionsmanager. Daten können sowohl in der Basis RDBMS als auch auf TS-Systemen gespeichert werden.

Tabelle 1.1 enthält eine Aufstellung ausgewählter Aussagen einzelner Projektpartner über Verbesserungen des operativen Geschäftes, die durch das Projekt ESTEDI in diesem Zusammenhang erreicht wurden [GKDT02].

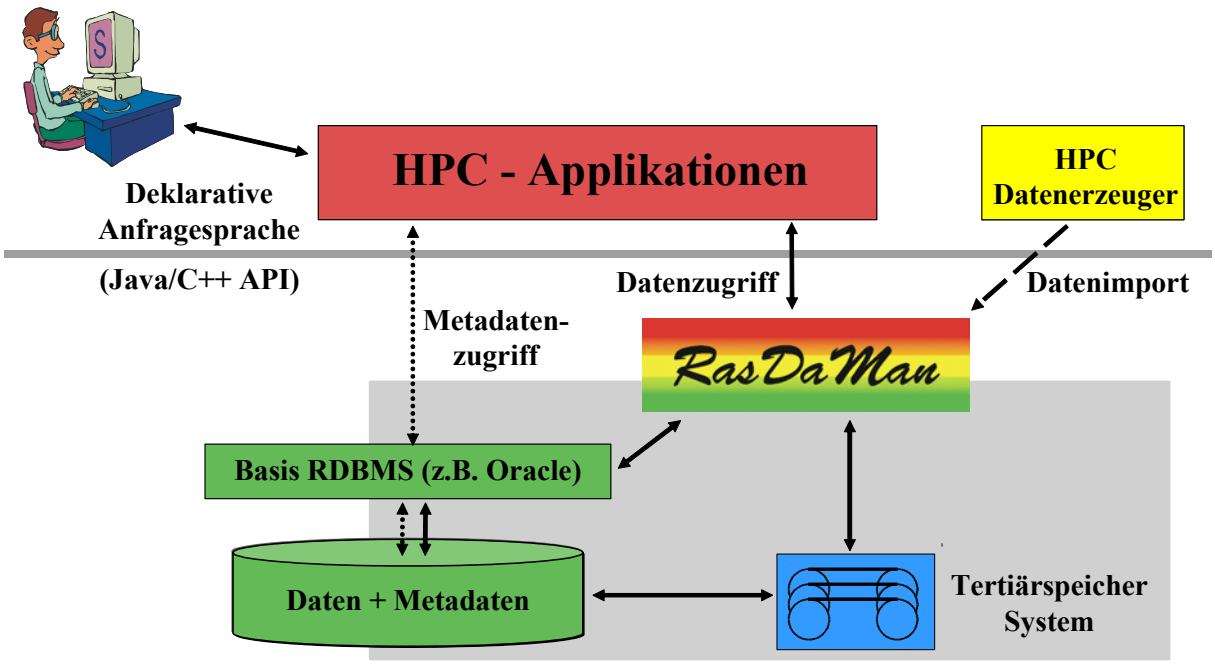


Abbildung 1.3: Referenzarchitektur der ESTEDI-Infrastruktur für HPC-Applikationen.

	Vor ESTEDI	Nach ESTEDI
Datenverwaltung	Unzureichende Datenorganisation. Daten oft durch Wissenschaftler selbst verwaltet.	Datenorganisation verbessert, da Daten und Metadaten in DBMS liegen.
Datenvorbereitung für Analysen	Primäres Datenprozessing notwendig.	Nicht notwendig, da bereits korrekt in DBMS eingefügt.
Datengranularität	Zugriff nur auf komplette Dateien möglich → Großes Datenvolumen.	Nur benötigte Daten (Teilbereiche) werden übertragen → Geringes Datenvolumen.
Datenretrieval	Langsam, da Dateien erst lokalisiert und dann komplett übertragen werden müssen. Teilweise manuell. Jeder Datensatz (Datei) muss separat angefordert werden.	Performanter und Automatisierter Zugriff auf Daten (Platte und TS-Medium) durch multidimensionale Anfragesprache und HEAVEN. Mehrere Objekte können durch eine Query analysiert werden.
Datenbearbeitung	Mehrere externe Werkzeuge notwendig, teilweise selbst zu entwickeln.	Analysen direkt in DBMS ausführen. Nur Analyseergebnisse werden zum Client übertragen.
Mehrbenutzerbetrieb	Nur eingeschränkt möglich.	Bereits durch DBMS unterstützt.

Tabelle 1.1: Erreichte Verbesserungen durch das Projekt ESTEDI [GKDT02].

Aus der Sicht der Datenbankforschung wurden zwei wissenschaftlich interessante Aspekte beleuchtet. Ein zentrales Thema war die Parallelisierung der Anfragebearbeitung des multidimensionalen Datenbanksystems. Als Stichwörter werden hier Inter- und Intra-Query Parallelisierung, sowie Inter- und Intra-Objekt Parallelisierung für multidimensionale Anfragen genannt. Parallelisierung wird im Kapitel 3.7.3, bzw. in der Dissertation von Karl Hahn mit dem Titel „Parallele Anfrageverarbeitung für multidimensionale Array-Daten“ behandelt [HR02, HRHB02, HRH03, Hahn05]. Das zweite zentrale Thema behandelt die effiziente Anbindung tertiärer Speichertechnologien an multidimensionale Datenbanksysteme. Das führt zu einem nutzerorientierten, einfachen und flexiblen Zugang zu extremen (nahezu unlimitierten) Datenvolumina [Rein01, RH02a, RHHB02, RH04a, RH04b]. Gerade die Entwicklungen in diesem Bereich waren für die Partner des ESTEDI-Projektes wesentlich, um den Übergang der Datenhaltung als Dateien zur Datenverwaltung in DBMS zu vollziehen. Grundlegende Arbeiten dieser TS-Anbindung erfolgten im ESTEDI-Projekt. Sie wurden für diese Dissertation erweitert, verfeinert und wissenschaftlich untersucht. Die in Abbildung 1.3 dargestellten, grau hinterlegten Bereiche, waren Gegenstand der Entwicklung von HEAVEN. Nähere Information über das Projekt ESTEDI sind bei [Rein03a] zu finden.

1.3 Ziele

Das Ziel dieser Arbeit ist es, eine geeignete und effiziente Infrastruktur für das Verwalten und das Retrieval sehr großer Mengen (Multi-PByte) multidimensionaler Daten über die gesamte Speicherhierarchie zu erarbeiten. Dabei sollen vor allem die bereits erwähnten Vorteile der Datenverwaltung durch ein DBMS berücksichtigt werden. Weiterhin ist es notwendig, aufgrund des hohen Datenvolumens, eine Speicherung der multidimensionalen Daten auf kostengünstigen Tertiärspeichermedien, wie z.B. das Magnetband, zu ermöglichen. Angestrebt wird die Verschmelzung des multidimensionalen Array-DBMS RasDaMan mit einem automatisierten TS-System, um ein nahezu unlimitiertes Speichervolumen zu erhalten. Die Ziele von HEAVEN lassen sich wie folgt zusammenfassen:

1. **Allgemeine Ziele:**
   - HEAVEN soll eine aktive, hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array-Daten realisieren. „Aktiv“ bedeutet in diesem Kontext, dass Anfragen über alle Ebenen der Speicherhierarchie automatisch beantwortet werden und keine weitere Nutzerinteraktion notwendig ist. Dies soll durch die Fusionierung eines mulidimensionalen Array-DBMS mit automatisierten TS-Systemen erreicht werden.
   - Angestrebt wird ein hochgradig erweiterbares und skalierbares Datenmanagement, das durch intelligente Automatisierung der Datenauslagerung, Datenspeicherung und des Datenretrievals realisiert werden soll. Damit können mit HEAVEN nahezu unlimitierte Datenbankgrößen erreicht werden.
   - Nutzern und Applikationen soll eine transparente Sicht auf die gespeicherten Daten gegeben werden, unabhängig vom tatsächlichen physikalischen Speicherort.

- Die Verwendung der multidimensionalen Anfragesprache von RasDaMan, soll das Datenretrieval vereinfachen. Die bisher notwendigen Werkzeuge bzw. Arbeitsschritte eines Anwenders werden reduziert. Wissenschaftler müssen nicht mehr mit Dateien, Verzeichnissen, Dateinamen und Dateiformaten umgehen. Sie können Daten auf einer adäquaten semantischen Ebene anfragen.

2. Erwünschte Funktionalität:

- *HEAVEN* soll die Möglichkeit schaffen, selektiv auf bestimmte Teilbereiche multidimensionaler Objekte zuzugreifen, die auf TS-Medien gespeichert wurden. Der „Nutzungsgrad", der von Wissenschaftlern angeforderten Daten, steigt dadurch um ein Vielfaches. Bisher verwendeten Wissenschaftler nur etwa 10% der angeforderten Daten [GKDT02]. Auch Datenzugriffe, die einen Schnitt durch eine Menge von Datenobjekten bedeuten (z.B. bei Zeitreihen) sollen ermöglicht werden. Dabei soll speziell auf eine dem jeweiligen Speicherort angepasste Datengranularität geachtet werden (Punkt 3). Das entlastet Netzwerke und ermöglicht neue Anwendungsgebiete, wie zum Beispiel die Datensimulation über Internet, da eine minimale Ergebnismenge übertragen wird.
- Eine Update-Funktionalität der auf TS-Medien gespeicherten Daten, unter Berücksichtigung der Konsistenzerhaltung, soll ermöglicht werden. Bisher waren oft nur Leseoperationen auf diese Objekte (Read-Only) möglich.
- Die Erweiterung der multidimensionalen Anfragesprache um neue Anfragekonstrukte, wie zum Beispiel das *Object-Framing*, soll neben neuen Analysemöglichkeiten auch das zu ladende Datenvolumen reduzieren.
- Eine geeignete, nach bestimmten Anfragemustern entsprechende Modellierung, der in der Datenbank und somit auf TS-Medien gespeicherten Daten, soll ein effizientes Datenretrieval gewährleisten. So ist eine primäre Datenverarbeitung vor der Simulation nicht mehr nötig. Bisher wurden Datensätze entsprechend ihrer Generierungsordnung gespeichert. Eine zeitaufwändige Rohdatenextraktion entfällt.
- Realisierung der Möglichkeit zur einfachen und adaptiven Archivierung von multidimensionalen Rasterdaten. Als Schlagwort dient hier auch *Information Lifecycle Management*.

3. Optimierung für effiziente TS-Zugriffe:

- Bei TS-Zugriffen sollen vor allem teure Medienwechsel und Suchoperationen reduziert werden. Aspekte der Performance spielen naturgemäß eine wesentliche Rolle im Bereich des Retrievals riesiger Datenmengen. Deshalb soll nicht nur eine einfache Anbindung eines TS-Systems an RasDaMan realisiert werden. Spezielle Optimierungskriterien sollen bei Datenzugriffen berücksichtigt und entwickelt werden. Schlagwörter hierzu sind Indexierung, *Clustering*, Query-*Scheduling* und *Caching*.
- Durch eine dem jeweiligen Speicherort angepasste Datengranularität, sollen die jeweiligen Eigenschaften der Speichermedien für ein effizientes Datenretrieval ge-

nutzt werden. So soll zum Beispiel bei Zugriffen auf TS-Medien, die gute Transferrate genutzt und teuere Operationen zur Positionierung vermieden werden.

- Das unter Punkt 2 aufgeführte Object-Framing, kann das von tertiären Speichermedien zu ladende Datenvolumen enorm reduzieren. Das führt zu einer Steigerung der Performance.
- Die Nutzung und Anpassung der von RasDaMan angebotenen Inter- und Intra-Query Parallelität, bzw. Inter- und Intra-Objekt Parallelität, ermöglichen eine beschleunigte Anfragebearbeitung. Der Datenzugriff auf Tertiärspeicher wird dabei durch Ausnutzung paralleler Hardware-Strukturen beschleunigt.
- Das entwickelte System soll gleichermaßen hinsichtlich operativer Daten und historischer Daten optimiert werden.

Bei der Formulierung der angestrebten Ziele, sind neben eigenen Ideen und Vorstellungen, auch Anforderungen und Wünsche der ESTEDI-Projektpartner eingeflossen. Zusammenfassend lässt sich sagen, dass bei der Realisierung von HEAVEN versucht wurde, den Balanceakt zwischen schnellem Zugriffsverhalten und günstigem Speicherplatz geeignet umzusetzen.

1.4 Aufbau

Um einen Überblick über Aufbau und Struktur der Arbeit zu vermitteln, folgt eine kurze Beschreibung der Inhalte der einzelnen Kapitel:

Kapitel 1 Einleitung, zeigt bestehende Defizite der Datenspeicherung und vor allem des Datenretrievals in Anwendungsbereichen mit sehr großen Datenvolumina (bis mehrere PByte). Diese Probleme motivieren die Notwendigkeit der Entwicklung von HEAVEN. Die grundlegende Architektur von HEAVEN entstand im Rahmen des ESTEDI Projektes (Kapitel 1.2). Erklärtes Ziel war es, für Hochleistungsrechenzentren, eine flexible und performante Infrastruktur für sehr große Datenvolumen zu etablieren. In Kapitel 1.3 werden die Ziele von HEAVEN dargestellt. Aufbau und Struktur der Arbeit wird in Kapitel 1.4 gezeigt.

Kapitel 2 Grundlagen, erläutert zunächst einige fundamentale Begriffe, die für das Verständnis der Arbeit von Bedeutung sind. In Kapitel 2.1 erfolgt eine Abgrenzung des Begriffes „multidimensionale Daten“, da dieser im Umfeld von DBMS unterschiedlich geprägt ist. Weiterhin wird in Kapitel 2.2 näher auf TS-Systeme mit unterschiedlichen Medientypen eingegangen. Nach der Eingliederung tertiärer Speichermedien in die Speicherhierarchie von Datenverarbeitungssystemen werden einige unterschiedliche TS-Medien vorgestellt und deren Speichertechnologie besprochen. HEAVEN wurde bewusst medienunabhängig konzipiert, um den Anforderungen an unterschiedliche Medientypen zu entsprechen. Allerdings gilt im HPC-Bereich das Magnetband in unterschiedlichen Ausprägungen als bevorzugtes Speichermedium. Aus diesem Grund wird die Frage geklärt, ob Magnetbänder noch zeitgemäß sind oder von Festplatten als Speichermedium abgelöst werden. Mit robotergesteuerten, hierarchischen Speichermanagement-Systemen, eröffnet sich die Möglichkeit, tausende von Medien automatisch zu verwalten. Diese automatisierten Systeme werden in HEAVEN zur Speicherung von

Daten auf TS-Medien verwendet. In Kapitel 2.3 werden die unterschiedlichen Möglichkeiten der Kopplung von DBMS und TS-Systemen betrachtet. Weiterhin wird in Kapitel 2.4 der Stand der Forschung und Entwicklung im Bereich der Anbindung von Tertiärspeicher-Systeme an DBMS anhand dreier Entwicklungen vorgestellt. Dabei wird auf essentielle Defizite und Probleme dieser Systeme eingegangen. Abschließend beschreibt Kapitel 2.5 grundlegende Konzepte des multidimensionalen Array-DBMS RasDaMan, das um Methoden der TS-Anbindung erweitert wurde. Das DBMS RasDaMan und hierarchische Speichermanagement-Systeme bilden die Grundlage von HEAVEN.

Kapitel 3 HEAVEN, beschreibt das im Rahmen dieser Dissertation entwickelte System HEAVEN, eine hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array-DBMS. Dabei wird das multidimensionale Array-DBMS RasDaMan mit einem hierarchischen Speichermanagement-System verschmolzen und mit effizienter und optimierter Retrievalfunktionalität angereichert. Die Systemarchitektur von HEAVEN wird in Kapitel 3.1 präsentiert. Kapitel 3.2 beschreibt, wie die Anbindung eines hierarchischen Speichermanagement Systems an das Array-DBMS RasDaMan erfolgt. Nach einer allgemeinen Beschreibung, folgt die konkrete Anbindung des hierarchischen Speichermanagement Systems TIVOLI der Firma IBM. Kapitel 3.3 diskutiert die Vor- und Nachteile einer direkten Anbindung eines Bandlaufwerkes an RasDaMan. Geeignete Möglichkeiten der Optimierung bei Zugriffen auf TS-Medien werden in Kapitel 3.4 klassifiziert. Schlagwörter sind das im Rahmen dieser Arbeit neu entwickelte Super-Kachel-Konzept und Inter- bzw. Intra-Super-Tile-Clustering, die in Kapitel 3.5 behandelt werden. Weiterhin zeigt Kapitel 3.6 das Exportieren von Daten auf TS-Systeme und Kapitel 3.7 das entsprechende Datenretrieval von tertiären Speichermedien. Die Funktionsweise von Update- und Delete-Operationen werden in Kapitel 3.8 erklärt. Kapitel 3.9 erläutert die Realisierung der Caching-Funktionalität von Array-Daten. Mit Object-Framing wird eine Technik präsentiert, die eine Reduzierung der TS-Zugriffskosten mit sich bringt und erweiterte Möglichkeiten bezüglich der Anfragesprache von RasDaMan bietet (Kapitel 3.10). Abschließend folgen in Kapitel 3.11 die Vorteile der Nutzung eines erweiterten Systemkataloges zur Verwaltung vorberechneter Ergebnisse von Aggregatoperationen.

Kapitel 4 Performance Evaluierung, bewertet das im Rahmen dieser Arbeit entwickelte System HEAVEN. Dabei wird zunächst in Kapitel 4.1 die verwendete Testumgebung vorgestellt. Weiterhin folgt in Kapitel 4.2 eine kurze Beschreibung der verwendeten Testdaten. Nach dieser Einführung wird in Kapitel 4.3 die Performance des Datenexports auf tertiäre Speichermedien untersucht. Das Kapitel 4.4 evaluiert über alle Ebenen der Speicherhierarchie die Performance beim Datenretrieval.

Kapitel 5 Diskussion, grenzt das entwickelte System HEAVEN gegenüber den Systemen SDM, MDBS, APRIL, StorHouse/RM, CERA und Postgres ab und diskutiert die gewonnenen Ergebnisse. Erkenntnisse aus dem ESTEDI-Projekt fließen in den Vergleich der Systeme mit ein. Als entscheidendes Bewertungskriterium gilt eine effiziente Verwaltung und Speicherung von multidimensionalen Array-Daten. Es werden essentielle Defizite und Probleme der in Kapitel 2.4 bzw. bei [Rein03b] untersuchten Systeme gegenüber HEAVEN aufgezeigt.

Kapitel 6 Zusammenfassung, stellt die umfangreichen Ergebnisse dieser Arbeit zusammenfassend dar.

Das **Notationsverzeichnis** enthält eine Auflistung der in dieser Arbeit verwendeten Nomenklatur.

In einem **Glossar** sind die in dieser Arbeit verwendeten Akronyme enthalten. Beim ersten Auftreten der Akronyme werden diese *kursiv* gesetzt.

Fundamental progress has to do with the reinterpretation of basic ideas.
Alfred North Whitehead (1861 - 1947)

Kapitel 2 Grundlagen

In diesem Kapitel werden zunächst einige Grundlagen und Begriffe erläutert, die für das Verständnis dieser Arbeit von Bedeutung sind. Zunächst erfolgt in Kapitel 2.1 eine Abgrenzung des Begriffes „multidimensionale Daten“, da dieser im Umfeld von DBMS unterschiedlich geprägt ist. Weiterhin wird in Kapitel 2.2 näher auf TS-Systeme eingegangen. Nach der Eingliederung tertiärer Speichermedien in die Speicherhierarchie von Datenverarbeitungssystemen, werden einige unterschiedliche TS-Medien vorgestellt und deren Speichertechnologie besprochen. HEAVEN wurde bewusst medienunabhängig konzipiert, um den Anforderungen an unterschiedliche Medientypen zu entsprechen. Allerdings gilt im HPC-Bereich das Magnetband in unterschiedlichen Ausprägungen als bevorzugtes Speichermedium. Aus diesem Grund wird die Frage geklärt, ob Magnetbänder noch zeitgemäß sind oder von Festplatten als Speichermedium abgelöst werden. Mit robotergesteuerten, hierarchischen Speichermanagement-Systemen, eröffnet sich die Möglichkeit, tausende von Medien automatisch zu verwalten. In Kapitel 2.3 werden die unterschiedlichen Möglichkeiten der Kopplung von DBMS und TS-Systemen betrachtet. Weiterhin wird in Kapitel 2.4 der Stand der Forschung und Entwicklung in diesem Bereich anhand dreier Entwicklungen vorgestellt. Es wird auf essentielle Defizite und Probleme dieser Systeme eingegangen. Abschließend werden in Kapitel 2.5 Besonderheiten multidimensionaler Array-DBMS gezeigt und ein Vergleich mit relationalen DBMS angestellt. Weiterhin werden grundlegende Konzepte des multidimensionalen Array-DBMS RasDaMan (Raster Data Management) beschrieben, welches in Kapitel 3 um Methoden der TS-Anbindung erweitert wurde. Das DBMS RasDaMan und hierarchische Speichermanagement-Systeme bilden die Grundlage von HEAVEN.

2.1 Multidimensionale Daten

Ursprünglich wurden DBMS entwickelt, um alphanumerische Daten effizient zu speichern, zu verwalten und abzufragen. Die Anforderung an die Datenhaltung ist im Laufe der Zeit gestiegen und es wurde notwendig, auch vermehrt multidimensionale Daten verwalten zu können. Dabei gibt es im Kontext von DBMS unterschiedliche Ausprägungen des Begriffes „multidimensionale Daten“. Als Hauptvertreter können zum Beispiel Data Warehouse Systeme, bzw.

OLAP (*Online Analytical Processing*) Systeme, *Geographische Informationssysteme* (GIS) und Array-DBMS unterschieden werden. Da im Rahmen dieser Arbeit das Array-DBMS RasDaMan betrachtet wird, folgt eine Abgrenzung gegenüber den anderen Einsatzgebieten. Diese Abgrenzung ist nicht vollständig: Sie hebt vor allem prinzipielle Unterschiede hervor. Betrachtet man die Quelldaten eines Data Warehouse Systems, so gruppieren sich diese in verschiedene Dimensionen, wie etwa Produkt, Zeit, Kunde und Region. Bei diesem multidimensionalen Würfel (Hyperquader bzw. Hypercube) handelt es sich typischerweise um einen dünn besetzten Raum (meist unter 1 %). Ein dreidimensionaler Datenwürfel ist beispielhaft in Abbildung 2.1 (links) zu erkennen.

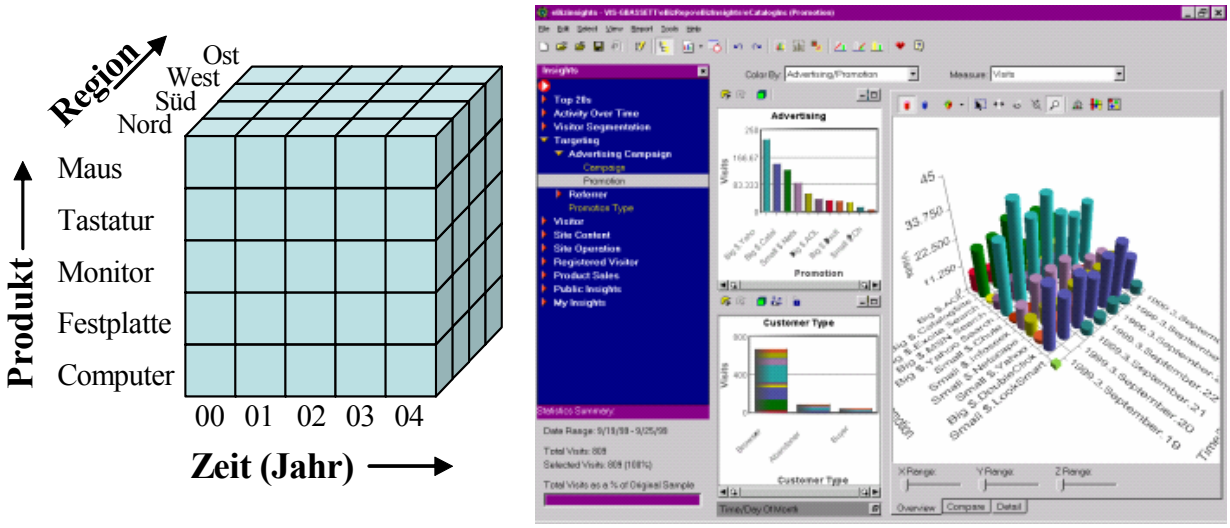


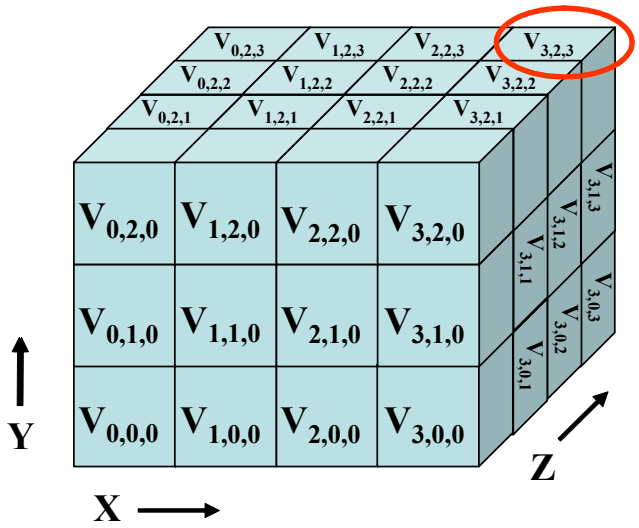
Abbildung 2.1: Links: Dreidimensionaler Datenwürfel eines Data Warehouse Systems; Rechts: OLAP-Anwendung mit graphischer Auswertung von Geschäftsprozessen.

Die rechte Seite der Abbildung 2.1 zeigt beispielhaft die Datenanalyse einer OLAP-Anwendung mit graphischer Repräsentation von Geschäftsprozessen auf der Basis eines Data Warehouses. Der dünn besetzte Raum wird meist als eine Menge von Tupel, bestehend aus Koordinatenvektor und Wert in relationalen DBMS gespeichert. Dies bedeutet, dass Zellkoordinaten explizit abgelegt werden. Bei Array-Daten oder Rasterdaten hingegen, handelt es sich um einen dicht besetzten Raum. Hier werden die multidimensionalen Daten in linearisierter Form gespeichert. Die Zellkoordinaten werden dabei implizit durch eine spezielle Linearisierungsordnung der Arrays gespeichert. Bei beiden Methoden werden einzelne Zellen durch eine eindeutige Koordinate näher definiert. Abbildung 2.2 stellt die Speicherung von multidimensionalen Daten als Array (links), bzw. als Koordinatenvektor (c_x, c_y, c_z) und Wert ($v_{x,y,z}$) in einer relationalen Tabelle (rechts) gegenüber.
In diesem Beispiel wird für jede Position im Raum ein Zellwert (value) $v_{x,y,z}$ gespeichert, der sich aus *Rot-Grün-Blau-Werten* (RGB-Werten) und einem Temperaturwert zusammensetzt. Bei der Speicherung der Daten als Vektoren in einer relationalen Tabelle müssen zusätzlich die Koordinaten (c_x, c_y, c_z) gespeichert werden. Aufgrund des zusätzlichen Speicheraufwan-

des ist diese Methode nur für dünn besetzte, multidimensionale Daten geeignet. Es stellt sich die Frage, ab welcher Datendichte, die linearisierte Speicherung, der vektorisierten Speicherung vorzuziehen ist. Überschreitet die Datendichte δ den Grenzwert δ_{grenz}, so ist die linearisierte Speicherung aufgrund des geringeren Speicherbedarfs besser geeignet. Diese Grenze der Datendichte δ_{grenz} lässt sich wie folgt bestimmen:

$$\delta_{grenz} = \frac{S_A}{S_V} = \frac{\prod_{i=1}^{d} a_i \sum_{j=1}^{n} t_j}{\prod_{i=1}^{d} a_i \left(\sum_{k=1}^{d} c_k + \sum_{j=1}^{n} t_j \right)} = \frac{\sum_{j=1}^{n} t_j}{\sum_{k=1}^{d} c_k + \sum_{j=1}^{n} t_j}$$

Der Zähler beschreibt den Speicherbedarf der Array-Speicherung S_A und der Nenner den Speicherbedarf der vektorisierten Speicherung S_V (Datenraum komplett besetzt) des gesamten Objektes, bzw. einer Koordinate. Wobei a_i die Ausdehnung des Objektes entlang einer Dimension i bezeichnet, mit der Dimensionalität d. Die Größe eines einzelnen Datentyps eines Zellwertes $v_{x,y,z}$, setzt sich aus n Datentypen mit der Größe t_j zusammen. Bei der vektorisierten Speicherung muss zusätzlich für jede Dimension k ein Koordinatenwert mit der Größe c_k gespeichert werden.



c_x	c_y	c_z	R	G	B	Temp
0	0	0	123	234	222	34,43
1	0	0	111	222	125	33,77
2	0	0	220	229	155	35,12
3	0	0	159	145	201	36,24
0	1	0	247	233	255	37,01
1	1	0	139	217	238	37,88
2	1	0	244	156	224	38,03
3	1	0	123	219	210	39,98
0	2	0	234	255	199	38,04
1	2	0	244	245	202	38,35
•						
•						
•						
3	2	3	214	237	202	40,59

Abbildung 2.2: Speicherung von dreidimensionalen Daten als Array (links) und als Koordinatenvektor und Wert in einer relationalen Tabelle (rechts).

Betrachten wir das Beispiel aus Abbildung 2.1 eines dreidimensionalen Datensatzes, erhalten wir eine Datendichte von 0,31 als δ_{grenz}: Mit einem zusammengesetzten Zellwert aus RGB-Werten (je 1 Byte) und einem Fließkommawert doppelter Genauigkeit für die Temperatur (8 Byte). Für den Koordinatenvektor werden je 8 Byte benötigt. Bei einer Objektausdehnung von $a_x = 1200$, $a_y = 720$, $a_z = 128$ ergibt das bei komplett besetztem Raum, einen Speicherbedarf von $S_A = 1{,}1$ GByte bei der Array-Speicherung, gegenüber $S_V = 3{,}6$ GByte bei der vektorisierten Speicherung. Bei höher dimensionalen Daten, bzw. weniger komplexen Zellwerten,

reduziert sich der Grenzbereich der Datendichte δ_{grenz} weiter. Bei einem Objekt mit einer weiteren Dimension erhalten wir $\delta_{grenz} = 0{,}25$. Diese Beispiele verdeutlichen, dass es durchaus sinnvoll ist, nicht nur voll, bzw. sehr dicht besetzte Datenräume als Array-Daten zu speichern. Der Speicherbedarf bei komplett besetztem Raum steigt bei einer weiteren, vierten Dimension mit einer Ausdehnung $a_4 = 300$ von $S_A = 1{,}1$ GByte auf $S_A = 339{,}9$ GByte und von $S_V = 3{,}6$ GByte auf $S_V = 1328{,}6$ GByte. Dabei zeigt sich der „Fluch der Dimensionalität“.

Nach dieser Ausführung werden geographische Informationssysteme näher betrachtet. In einem GIS werden inhaltlich unterschiedliche Objektklassen in verschiedenen Schichten (Layer, Themes) abgelegt. Die linke Seite der Abbildung 2.3 zeigt unterschiedliche Schichten: Wie zum Beispiel Hydrologie, Topologie und Bodennutzung, die einer Landkarte, bzw. einem Satellitenbild überlagert wurden. Diese Themen können einzeln ausgewertet und präsentiert, aber auch zueinander in Beziehung gesetzt werden. Die rechte Seite der Abbildung 2.3 stellt eine Landkarte mit überlagerten Markierungen (Themen) eines GIS dar. Einige GIS sind in der Lage, sowohl Vektordaten, als auch Rasterdaten zu verarbeiten. Allerdings ist die Dimensionalität meist auf zwei Dimensionen (maximal drei Dimensionen) beschränkt. Im Gegensatz dazu, ist das Array-DBMS RasDaMan, in der Lage, Daten beliebiger Dimensionalität zu speichern und zu verarbeiten.

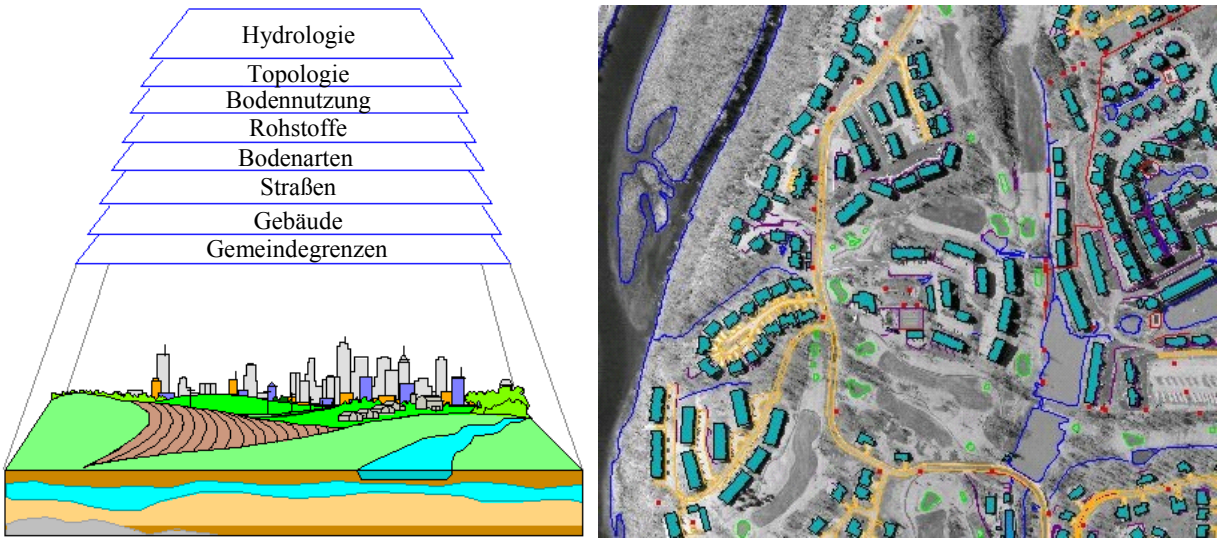


Abbildung 2.3: Links: Schichtenmodell mit unterschiedlichen Objektklassen eines GIS; Rechts: Abbildung einer Landkarte mit Markierungen durch ein GIS.

Neben den genannten Unterschieden zu einem Array-DBMS, gibt es weitere Abgrenzungen bezüglich der Anfragesprache, physikalische Datenspeicherung, Indexierung usw. Weiterführende Betrachtungen zu Data Warehouse, OLAP und GIS sind bei [BG01, RSV02] zu finden. Eine formale Definition des in dieser Arbeit verwendeten logischen und physikalischen Datenmodells folgen in Kapitel 2.5.

2.2 Tertiärspeichersysteme

In diesem Kapitel werden Grundlagen von TS-Systemen betrachtet. Zunächst erfolgt eine Eingliederung der TS-Systeme in die Speicherhierarchie eines Datenverarbeitungssystems. Danach werden unterschiedliche Arten von TS-Medien beschrieben und die Frage geklärt, ob Magnet-Bänder noch zeitgemäß sind. Abschließend wird auf die Funktionsweise hierarchischer Speichermanagement Systeme eingegangen.

2.2.1 Speicherhierarchie

Grundsätzlich sind Speichermedien charakterisiert durch Zugriffszeit[5], Datenrate und Speicherkapazität. Dabei besteht ein prinzipieller Konflikt zwischen der Minimierung der Zugriffszeit und der Maximierung der Speicherkapazität. In Rechnersystemen werden unterschiedliche Speichertechnologien zu Speicherhierarchien kombiniert, um einen Kompromiss aus schnellem Zugriff und großen Speicherkapazitäten bei angemessenen Kosten zu erreichen. Abbildung 2.4 zeigt eine solche Speicherhierarchie, dargestellt als Pyramide.

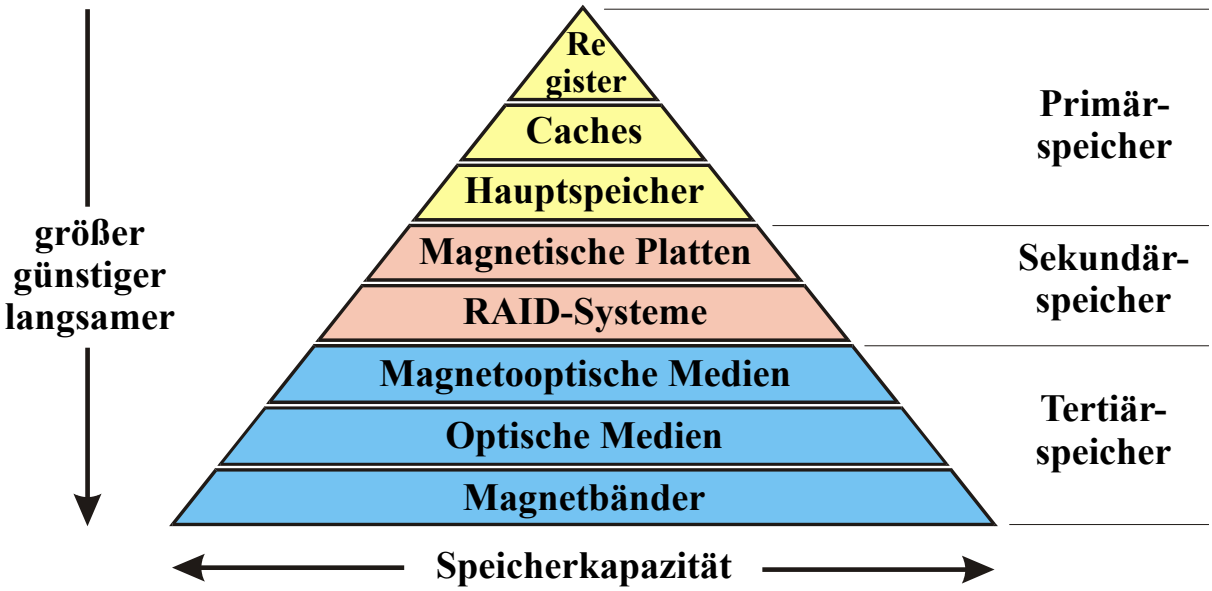


Abbildung 2.4: Speicherhierarchie eines Datenverarbeitungssystems.

An der Spitze der Pyramide ist der Speicher mit der geringsten Zugriffszeit, geringsten Kapazität und den höchsten Kosten angesiedelt. Je weiter unten sich Speichertechnologien in der Pyramide befinden, desto größer und günstiger, aber auch langsamer, wird dieser Speicher. Der Grund, wieso Speicherhierarchien in Rechnersystemen funktionieren, liegt in der Lokalität der Datenzugriffe. Diese Lokalitätseigenschaften werden auf jeder Hierarchieebene durch geeignete Speicherverwaltungs- und Ersetzungsstrategien unterstützt. Damit wird erreicht, dass sich Daten, die demnächst vom Prozessor benötigt werden, möglichst nahe an der Spitze

[5] Benötigte Zeit für die Lokalisierung einer Datei und Positionierung des Lesekopfes eines Speichermediums. Das Lesen gespeicherter Information, bzw. Schreiben neuer Information ist nicht in der Zugriffszeit enthalten.

der Pyramide befinden. Bei fehlenden Lokalitätseigenschaften, wie zum Beispiel bei wahlfreien und gleichverteilten Zugriffsmustern, müssen die Daten bei jeder Referenz von einer langsamen Speicherhierarchie geladen werden, dem permanenten Speicherort. Durch die Lokalitätseigenschaft und optimierten Ersetzungsstrategien werden hohe Trefferraten erreicht [HR99, SH99].

Generell lassen sich Primär-, Sekundär- und Tertiärspeicher unterscheiden. Diese Einteilung wird jedoch in der Literatur nicht einheitlich verwendet. Teilweise werden Tertiärspeicher nicht explizit aufgeführt, sondern zu den Sekundärspeichern gezählt. Im Kontext dieser Arbeit sind die Begriffe wie in Abbildung 2.4 belegt. Typische Zugriffszeiten und Kapazitäten diverser Speichertechnologien werden in Tabelle 2.1 verglichen (Stand April 2004).

Speicherart		**Typische Zugriffszeit**	**Typische Kapazität**
Primärspeicher	Register	$\leq$ 1 ns	256 Byte – 1 KByte
	Caches	5 – 10 ns	128 KByte – 4 MByte
	Hauptspeicher	8 – 20 ns	64 MByte – 1 GByte
Sekundärspeicher	Magnetische Platten	5 – 15 ms	60 GByte –180 GByte
	RAID-System	5 – 15 ms	100 GByte – 4 TByte
Tertiärspeicher	MO-Medien	20 – 50 ms	128 MByte – 9,5 GByte
	Optische Medien	80 – 200 ms	650 MByte – 9,4 GByte
	Magnetbänder	10 s – 5 min	20 GByte – 600 GByte

Tabelle 2.1: Typische Zugriffszeiten und Kapazitäten verschiedener Speichertechnologien.

Als Primärspeicher werden alle Speicher mit wahlfreiem Zugriff (Random Access) bezeichnet, auf die der Prozessor direkt mit voller Geschwindigkeit zugreifen kann. Dazu zählen Register eines Prozessors, Caches[6] und der Hauptspeicher (Main Memory, bzw. Arbeitsspeicher). Primärspeicher bieten schnellen Zugriff (Nanosekunden-Bereich) auf Daten, sind aber hinsichtlich ihrer Kapazität begrenzt.

Hintergrundspeicher mit index-sequentiellem[7] (quasi-wahlfreiem) Zugriff, wie magnetische Festplatten, bzw. RAID-Systeme (*Redundant Arrays of Independent Disks*), werden als Sekundärspeicher eingeordnet. Diese Speicher verfügen über große Kapazitäten, weisen aller-

[6] Caches (SRAM, *Static Random Access Memory*) werden durch mehrere Level realisiert: Level 1 (L1) Cache innerhalb des CPU-Chips, Level 2 (L2) Cache auf CPU-Modul, bzw. einen zusätzlichen Level 3 (L3) Cache. Aktuelle Rechnersysteme verwenden SDRAM (*Synchronous Dynamic Random Access Memory*) oder RDRAM (*Rambus DRAM*) Technologie. Die Zugriffszeit ist abhängig von der Taktfrequenz, da die Speichermodule synchron mit dem Speicherbus arbeiten.

[7] Auf größere Speicherblöcke kann wahlfrei zugegriffen werden. Auf einzelne Speicherzellen innerhalb eines Speicherblockes kann nur sequentiell zugegriffen werden.

dings gegenüber Primärspeicher einen bis Faktor 10^6 langsameren Zugriff auf. Dieser gravierende Unterschied in der Zugriffszeit wird auch als Zugriffslücke bezeichnet.

Speichertechnologien mit weit höheren Zugriffszeiten und Speichermedien, auf die nicht direkt zugegriffen werden kann, gliedern sich in die Kategorie der Tertiärspeicher ein. Nicht direkt zugreifbar bedeutet, dass Medien manuell bedient werden müssen oder in robotergesteuerten Bibliotheken organisiert sind und erst bei einem Zugriff in die entsprechenden Schreib-/Lesestationen bewegt werden. Die Zugriffslücke zwischen Sekundär- und Tertiärspeicher erreicht ebenfalls einen Faktor von bis zu 10^6. Zu den TS-Medien zählen *magnetooptische* (MO) Medien, optische Medien und Magnetbänder. Weiterführende Information über Speichertechnologien sind bei [Völz96, MD02] zu finden.

2.2.2 Arten von Tertiärspeichermedien

Grundsätzlich lassen sich drei Tertiärspeichertechnologien unterscheiden. Hierzu gehören magnetooptische Speicher, optische Speicher und Magnetbänder, die sich auf dem Markt in unterschiedlichen Anwendungssegmenten etabliert haben. Magnetooptische und optische Speicher werden überwiegend bei kleinen bis mittleren Datenmengen (GByte bis TByte) eingesetzt, wenn schneller Zugriff erforderlich ist. Für die Speicherung sehr großer Datenmengen (TByte bis PByte) werden vor allem Magnetbänder verwendet. Da HEAVEN im Rahmen des ESTEDI Projektes für den *High Performance Computing* (HPC) Bereich mit sehr großen Datenmengen entwickelt wurde, konzentriert sich diese Arbeit vor allem auf das Speichermedium Magnetband. Magnetooptische und optische Speichermedien, spielen im Bereich der Speicherung großer Datenmengen eine untergeordnete Rolle und werden daher nicht detailliert betrachtet. HEAVEN ist selbstverständlich auch in der Lage, mit diesen Speichermedien zu arbeiten. Weiterhin wird der holographische Speicher als eine zukunftsweisende Speichertechnologie behandelt und bewertet.

2.2.2.1 Magnetooptische Speicher

Die magnetooptische Speicherung (MO bzw. MOD, *Magneto Optical Disc*) nutzt für die Aufzeichnung und für die Wiedergabe zwei unterschiedliche physikalische Effekte. Beim thermomagnetischen Schreibvorgang wird das magnetische Material mit einem Laserstrahl soweit erhitzt, dass die Curie- oder Inversionstemperatur erreicht wird. Danach ist das Material unmagnetisch. Beim Abkühlen wird durch ein angelegtes Magnetfeld, die neu entstehende Magnetisierung in eine gewünschte Richtung, senkrecht zur Schichtebene, erzwungen. Durch diese Methode ist bei Zimmertemperatur eine sehr hohe Koerzitivfeldstärke nötig, um einzelne Bits umkippen zu lassen. Daher können nicht einmal starke Magnetfelder MO-Medien schaden. Werden MO-Medien bei Zimmertemperatur gelagert, garantieren die Hersteller eine Datensicherheit von 30 bis 50 Jahren. Beim Lesevorgang wird der Kerr-Effekt ausgenutzt. Der Kerr-Effekt beschreibt das Phänomen, dass bestimmte Materialien die Polarisierungsrichtung des Lichtes bei der Reflexion um ca. ein Grad ändern. Diese Richtungsänderung ist abhängig von der Magnetisierungsrichtung. Ein Detektor mit Polarisationsfilter misst, in

welche Richtung sich die Polarisationsebene des Laserstrahls gedreht hat und erkennt somit, ob eine Null oder eine Eins geschrieben wurde. MO-Speicher ordnen die Daten in einer Spiralspur von innen nach außen an. MO-Speichermedien sind in festen Gehäusen (Cartridges) untergebracht und werden in der 3,5 und 5 ¼ Zoll-Technologie hergestellt. Durch das feste Gehäuse sind MO-Medien sehr robust gegenüber mechanischen Beschädigungen [Dätw98, Völz96]. Typische Zugriffszeiten und Kapazitäten von MO-Medien sind in Tabelle 2.1 aufgeführt. Der €/GByte-Preis eines MO-Mediums liegt zwischen 8 und 20 €, je nach Ausführung und Hersteller.

2.2.2.2 Optische Speicher

Optische Speichermedien wurden vor allem durch die *Compact Disk* (CD) populär. Nach Angaben der Hersteller tritt die *Digital Versatile Disk* (DVD) die Nachfolge der CD an. Der Lesevorgang optischer Speicher ähnelt dem magnetooptischer Speicher. Einzelne Bits können als Null oder Eins interpretiert werden, da ein Laserstrahl unterschiedlich vom Medium reflektiert wird. Im Gegensatz zu magnetooptischen Speichern wird bei optischen Speichern die unterschiedliche Reflektion durch Vertiefungen (Pits) in der Reflektionsschicht realisiert. Der Bereich zwischen den Pits wird „Land“ genannt. Trifft der Laser auf eine Vertiefung (Pit) bzw. auf eine „normale“ Fläche (Land), so wird der Laserstrahl als Eins interpretiert. Bei einem Übergang von Pit auf Land bzw. Land auf Pit wird eine Null gelesen. Daten werden auf einer spiralförmigen Spur von innen nach außen angeordnet. Im Unterschied zu einer CD, können bei einer DVD je nach Kapazität, Daten in mehreren Schichten mit je 4.7 GByte, bzw. auf beiden Seiten des Mediums gespeichert werden. CD und DVD sind in drei Varianten erhältlich: Medien, die industriell hergestellt werden und vom Anwender nicht beschrieben werden können. CD-R bzw. DVD-R (*Recordable*) Medien, die der Anwender einmal beschreiben kann, so genannte WORM-Medien (*Write Once Read Many*). Und CD-RW, bzw. DVD-RW (*Re-Writable*) Medien, die öfters beschrieben werden können. Weiterführende Information über optische Speicher sind bei [Dätw98, TG01, Völz96] zu finden. Typische Zugriffszeiten und Kapazitäten sind in Tabelle 2.1 aufgelistet. Die €/GByte-Preise liegen für CD-R bei 0,70 €/GByte, für CD-RW bei 1,80 €/GByte, für DVD-R bei 1,90 €/GByte und für DVD-RW bei 2,60 €/GByte. Als Standard der nächsten Generation gilt die *Blu-Ray-Disk*. Diese Blu-Ray-Technologie basiert, anders als bei herkömmlichen DVDs, auf einem blau-violetten Laser, der im Gegensatz zu einem roten Laser feinere Strukturen lesen, bzw. schreiben kann. Dadurch erhöht sich die Kapazität auf 27 GByte pro Speicherschicht, gegenüber 4,7 GByte bei einer DVD. Ein direkter Konkurrent um die Nachfolge der DVD, ist die *Advanced Optical Disc* (AOD), die ebenfalls einen blau-violetten Laser verwendet. Allerdings wird nur eine Kapazität von 20 GByte pro Speicherschicht erreicht. Dieser Nachteil soll durch bessere Kompressionsverfahren ausgeglichen werden. Es ist momentan noch nicht abzusehen, wer die Nachfolge der DVD antreten wird.

Neben Medien wie CDs und DVDs wurde seit 15 Jahren auch im Bereich optischer Bänder geforscht und entwickelt. Allerdings wurde bis heute keine Breitenanwendung dieses Spei-

chertyps erreicht. Die Kapazitäten liegen bei einem TByte mit einer Datenraten von 25 MByte/s [Völz96, WSC00].

2.2.2.3 Magnetbänder

Seit der Entwicklung der ersten Magnetbandgeneration von IBM vor ca. 50 Jahren, wurden sehr viele verschiedene Arten von Magnetbändern entwickelt. Bis heute haben sich allerdings nur einige Technologien auf dem Markt durchgesetzt. Im professionellen Bereich werden vor allem DLT (*Digital Linear Tape*) der Firmen Quantum und Tandberg, Super-DLT von Quantum, LTO (*Linear Tape Open*) von IBM, AIT (*Advanced Intelligent Tape*) von Sony und Mammoth von Exabyte eingesetzt. Im semi-professionellen Umfeld werden weitestgehend DDS (*Digital Data Storage*) Magnetbänder verwendet. Zunächst lassen sich die Magnetbänder entsprechend ihrer Gehäuseformen unterscheiden. In Abbildung 2.5 werden exemplarisch die Bauformen Cartridge und Kassette gezeigt. Diese begriffliche Unterscheidung ist in der Praxis nicht durchgängig. Fälschlicherweise werden teilweise, auch von den Herstellern, Magnetbänder in der Kassettenbauweise als Cartridge bezeichnet. Bei einer Cartridge befindet sich das Band auf nur einer Rolle und wird während eines Arbeitsvorganges aus dem Gehäuse herausgezogen und im Laufwerk auf eine zweite Rolle gespult. Bei der Ausführung als Kassette befinden sich zwei Rollen im Gehäuse des Mediums. Nachteil dieser Gehäuseform ist die komplexere Bandführung zum Schreib/Lese-Kopf, was zu einem höheren Verschleiß des Bandes führt, verglichen mit einer Cartridge. Weiterhin kann eine Cartridge bei gleicher Baugröße mehr Magnetband aufnehmen, da keine zweite Spule vorhanden ist. Die Magnetbänder DLT, Super-DLT und LTO entsprechen der Cartridgebauweise, während AIT, Mammoth und DDS als Kassetten realisiert wurden.

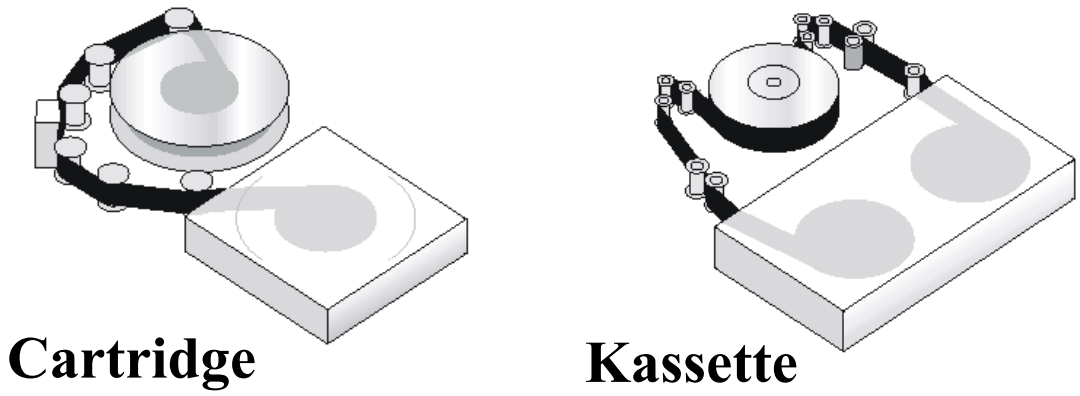


Abbildung 2.5: Unterschiedliche Gehäuseformen von Magnetbänder [GMW01].

Neben der Einteilung in Gehäuseformen, lassen sich Magnetbänder entsprechend ihrer Aufzeichnungsverfahren klassifizieren. Für die oben genannten Medientypen können grundsätzlich die lineare Aufzeichnungsmethode und das Schrägspurverfahren (*Helical Scan*) unterschieden werden (Abbildung 2.6). Das lineare Verfahren wird weiter unterteilt in das lineare parallele Verfahren und das lineare Serpentinenverfahren.

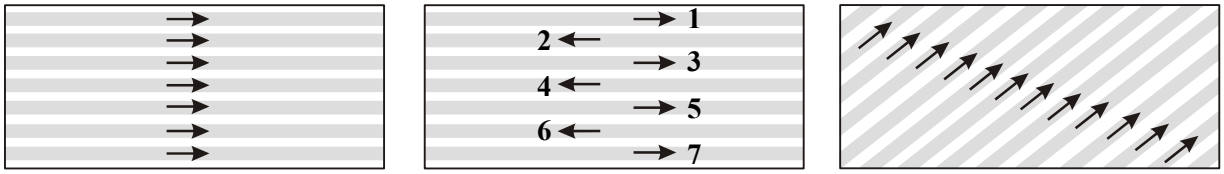


Abbildung 2.6: Aufzeichnungsverfahren bei Magnetbändern.

Aufzeichnungsverfahren: Linear, Parallel
Wird das Magnetband linear parallel beschrieben, findet ein Mehrspurkopf Verwendung, der mehrere Datenspuren gleichzeitig beschreibt. Aufgrund der hohen Anzahl an Datenspuren pro Bandquerschnitt und den damit verbundenen teuren Mehrspurköpfen, wird dieses Verfahren in seiner Reinform meist nicht angewendet.

Aufzeichnungsverfahren: Linear, Serpentinen
Eine andere Variante, ein Magnetband linear zu beschreiben, ist die Serpentinentechnik. Hierbei wird ein einfacher Kopf verwendet, der jeweils nur eine Spur des Bandes beschreibt. Wenn das Bandende erreicht ist, wird die Bandlaufrichtung umgekehrt und in der anderen Richtung beschrieben (Abbildung 2.6, Mitte). Bei der Serpentinentechnik wird eine wesentlich kleinere Datenrate erreicht. Moderne Magnetbänder, wie das DLT, Super-DLT oder LTO, verwenden eine Kombination aus den beiden Lineartechniken. Bei LTO Ultrium Magnetbändern, mit einer Grundkapazität (*Native Capacity*) von 100 GByte, sind 384 Datenspuren realisiert. Diese werden in vier Datenbändern mit je 96 Datenspuren unterteilt. Die Daten werden parallel auf acht Datenspuren innerhalb des gleichen Datenbandes geschrieben, bzw. gelesen. Für die anderen Datenspuren wird die Serpentintechnik verwendet. Um das ganze Magnetband zu beschreiben, muss das Magnetband 48-mal komplett durchlaufen werden [GWM01, Völz96, Watk00, Degr01].

Aufzeichnungsverfahren: Schrägspur
Die Schrägspurtechnik gilt als weiteres Aufzeichnungsverfahren. Es unterscheidet sich grundlegend von den beiden anderen Verfahren. Wie der Name schon vermuten lässt, werden die Datenspuren nicht linear, sondern schräg nebeneinander angeordnet (Abbildung 2.6, rechts). Dieses Verfahren wurde von Videorecorder übernommen und für die Datenspeicherung weiterentwickelt und optimiert. Die Datenspuren sind dabei um sechs Grad geneigt. Das Laufwerk besitzt eine Kopftrommel, die mit je zwei Lese- und Schreibköpfen ausgestattet ist, die im Verhältnis zur Bandlaufrichtung schräg liegen. Beim Lese- bzw. Schreibvorgang rotiert die Kopftrommel entgegen der Laufrichtung des Magnetbandes. Eine wesentlich höhere Datenspurdichte gegenüber den Linearverfahren wird durch ein schnelles Rotieren der Kopftrommel und einer im Verhältnis zur Lineartechnik langsamen Bewegung des Bandes erreicht. Als Vertreter dieses Aufzeichnungsverfahren gelten die AIT-, Mammoth- und DDS-Bänder. Bei Magnetbändern mit Schrägspurtechnik wird auf die Kassette als Gehäuseform zurückgegriffen. Mit dem bereits beschriebenen Nachteil der hohen Bandabnutzung [Exab01, WCMS01].

Um sicher zu gehen, dass die Daten richtig auf das Magnetband geschrieben wurden, müssen die einzelnen Bits Korrektur gelesen werden. Bei dem read-after-write Verfahren, muss das Magnetband nach dem Schreiben noch einmal komplett gelesen und mit den Originaldaten verglichen werden. Da dies sehr zeitaufwändig ist, wird in modernen Bandlaufwerken die read-while-write Methode eingesetzt. Hier wird von einem Lesekopf, der gleich hinter dem Schreibkopf angebracht ist, die Korrektheit der geschriebenen Bits überprüft. Tauchen Fehler auf, so wird der geschriebene Block als fehlerhaft markiert und die Daten erneut geschrieben.

Ein Vergleich unterschiedlicher Magnetbandtypen wird in Tabelle 2.2 gezeigt. Interessant sind vor allem die native Kapazität, die native Transferrate und die mittlere Zugriffszeit. Da Bandlaufwerke über eine in der Hardware implementierte Datenkompression verfügen, geben Hersteller oft als Kapazität ihrer Magnetbänder eine angenommene, komprimierte Kenngröße an. Wobei meist ein Komprimierungsfaktor von 2:1 veranschlagt wird. Der tatsächliche Komprimierungsfaktor hängt logischerweise sehr stark von der Art und Struktur der zu speichernden Daten ab. In dieser Arbeit sind Kapazitäten ohne weitere Angaben native Kapazitäten.

Kennwerte	DLT 8000	Super DLT 600	LTO Ultrium-2	AIT-3	Mammoth 2	DDS-4
Kapazität, native	40 GB	300 GB	200 GB	100 GB	60 GB	20 GB
Kapazität, komprimiert	80 GB	600 GB	400 GB	260 GB	120 GB	40 GB
Transferrate, native	6 MB/s	36 MB/s	40 MB/s	12 MB/s	12 MB/s	2 MB/s
Mittlere Zugriffszeit[8]	60 s	79 s	95 s	27 s	47 s	55 s
Mittlere Medienladezeit	37 s	12 s	20 s	10 s	17 s	24 s
Lebensdauer (Jahre)	30	30	30	30	30	10
Max. Medienbenutzung[9]	15.000	17.850	12.000	30.000	20.000	100

Tabelle 2.2: Technologievergleich unterschiedlicher Magnetbänder (Stand August 2004).

Bei genauer Betrachtung fällt auf, dass Transferraten moderner Bandtechnologien, wie Super-DLT 320, LTO Ultrium-2, AIT-3 oder Mammoth 2, nahe an Transferraten von Festplatten (30 – 60 MB/s) herankommen und teilweise auch erreichen. Bandlaufwerke sind bei der Transferrate um einen Faktor 1,5 bis 5 langsamer als Festplatten. Vergleicht man allerdings die mittlere Zugriffszeit von Festplatten (5 ms – 12 ms) und Bandlaufwerken (27 s – 95 s), macht sich der gravierende Unterschied zwischen index-sequentiellem und sequentiellem Zugriff mit einem Faktor von 10^3 bis 10^4 bemerkbar. Muss das Magnetband noch extra in die Ladestation des Bandlaufwerkes eingelegt werden (meist automatisiert), so addiert sich zur mittleren Zugriffszeit für den ersten Zugriff noch die mittlere Medienladezeit. Die in Tabelle 2.2 aufge-

[8] Zeit, die benötigt wird, um ein Magnetband von Beginn bis zur Mitte des Bandes zu positionieren.
[9] Anzahl der vom Hersteller garantierten Lese-, Schreib-, bzw. Positionierungsoperationen eines Magnetbandes.

führte maximale Medienbenutzung zeigt an, wie oft ein Magnetband vollständig gelesen oder geschrieben werden kann.

Betrachtet man die Entwicklung der Kapazitäten von Magnetbändern, so verdoppeln sich diese alle 12 – 18 Monate. Bei Festplatten werden die Kapazitäten alle 12 - 15 Monaten verdoppelt. Dieses Wachstum geht konform mit der Entwicklung der Speicherdichte magnetischer Medien: Mit einer momentanen Verdoppelungsrate von weniger als 18 Monaten. Etwa 1990 wurde das Hoagland-Gesetz aus dem Jahre 1985 außer Kraft gesetzt, das von einer Verdopplung der Speicherdichte alle 30 – 36 Monate ausging [Hoag85]. So genannte Road Maps der Herstellerfirmen zeigen die geplante, zukünftige Weiterentwicklung ihrer Produkte. Die vierte Generation der Super-DLT Medien soll Ende 2006 auf dem Markt erscheinen und eine Kapazität von 1,2 TByte bei einer Transferrate von über 100 MByte/s aufweisen. LTO Ultrium und AIT Medien sollen immerhin bis 2007 Kapazitäten von 800 GByte haben. IBM hat bereits heute Prototypen eines LTO Magnetbandes mit einer Kapazität von einem TByte vorgestellt [CIEJ03].

2.2.2.4 Zukunftsweisende Speichertechnologien

Gegenwärtigen Speichermedien ist gemeinsam, dass die Information durch eine lokale und möglichst kleine Änderung der optischen bzw. magnetischen Eigenschaften auf der Oberfläche eines Trägers in zwei Dimensionen gespeichert wird. Die dadurch erreichbaren Speicherdichten liegen im Bereich von 70 GBit/cm^2 bei einer CD und 3 TBit/cm^2 bei einer Festplatte. Aktuelle Speicherdichten einer handelsüblichen Festplatte erreichen heute ca. 0,3 – 0,6 TBit/cm^2. Die zukunftsweisende Technologie der holographischen Speicher, könnte die erreichbare Speicherdichte revolutionieren. Daten werden nicht nur auf der Oberfläche, sondern im Volumen eines Mediums gespeichert. Der Übergang in die dritte Dimension erlaubt Speicherdichten mit einem theoretischen Limit von $1/\lambda^3$. Bewegt sich die Wellenlänge λ eines Lasers im sichtbaren Spektralbereich, können Speicherdichten von 10^{12} Bits/cm^3 erreicht werden [Mech01]. Holographische Speicher versprechen Kapazitäten von einigen Terabyte in wenigen Kubikzentimeter großem Medium, bei Zugriffsraten von mehr als 1 Gigabyte pro Sekunde.

Holographische Speicher

Die Grundidee holographischer Speicher besteht in der vollständigen Speicherung des von einem Objekt (Bild, Datenseite) ausgehenden Wellenfeldes, durch Ausnutzen des photorefraktiven Effektes[10]. Um das zu erreichen, werden zwei Laserwellen im holographischen Speichermaterial überlagert. Einer dieser Laserwellen werden Daten aufmoduliert und ist Träger der zu speichernden Information, während die andere eine Referenzwelle darstellt. In der Hologrammebene entsteht durch die Überlagerung ein Muster heller und dunkler Bereiche: Ein so genanntes Interferenzmuster. Dieses Muster wird im holographischen Material

[10] Von A. Ashkin in den Bell Laboratories 1966 zunächst als störender Effekt („optical damage“) entdeckt. Bald stellte sich jedoch heraus, dass die auftretenden Inhomogenitäten des Brechungsindexes durch Licht ein Phasenhologramm darstellen und sich für die dynamische Holographie ausnutzen lassen.

wie auf einem fotografischen Film gespeichert. Beleuchtet später nur die Referenzwelle das Hologramm, so entsteht durch Ablenkung an diesen Bereichen die gesamte ursprüngliche Information. Diese wird ausgelesen und in verarbeitbare Signale umgewandelt. Um eine höhere Speicherkapazität zu erreichen, werden mehrere planare Hologramme übereinander angeordnet. Eine bekannte Realisierung eines solchen Quasivolumenspeichers ist die Tesa-Rom, die von der Universität Mannheim und der Firma Bayer AG entwickelt wird. Auf einer zehn Meter langen Tesafilmrolle lassen sich 10 GByte Daten speichern. Volumenhologramme können dagegen noch weitaus höhere Speicherdichten realisieren, da Datenseiten dreidimensional gespeichert werden. Voraussetzung sind Speichermaterialien mit ausreichender Dicke von mindestens einem Millimeter. Kleinste Änderungen des Beleuchtungswinkels, der Wellenlänge oder der Phasenverteilung, welche die Eigenschaften des Volumenhologramms festlegen, ermöglichen das Speichern einer neuen Datenseite. Durch dieses Verfahren können eine Vielzahl (mehrere Tausende) von Hologrammen an einem Ort überlagert werden. Als Speichermedium werden Materialien verwendet, die auf den Lichteinfall mit der Änderung eines Parameters, wie zum Beispiel der Absorption oder des Brechungsindexes reagieren (z.B. photorefraktive Kristalle). Im Falle der Änderung des Brechungsindexes, ist der Effekt reversibel und ermöglicht somit die Realisierung von wieder beschreibbaren holographischen Speichern.

Volumenholographische Speicher

Ein volumenholographischer Speicher von der Größe eines Zuckerwürfels (ca. 2,8 cm^3) kann eine Speicherkapazität im Bereich von mehreren TByte bei Ausleseraten von mehreren GByte/s und eine mittlere Zugriffszeit von einer Millisekunde erreichen. Diese schnellen Zugriffszeiten können durch paralleles Auslesen realisiert werden. Zum Vergleich: Moderne SCSI-(*Small Computer System Interface*) Festplatten der Baugröße 3,5 Zoll, erreichen Zugriffszeiten von 5 ms bei einer Kapazität von 180 GByte. Ein Nachteil der holographischen Speicherung ist die langsame Schreibzeit, die im Bereich von einigen 100 Millisekunden liegt. Dieser Nachteil setzt sich fort, da technologisch bedingt, beim Ändern eines einzelnen Bits einer Datenseite, die komplette Datenseite neu geschrieben werden muss. Durch die Schreib/Lese-Charakteristika eignet sich der Speicher vor allem für relativ statische Daten und gliedert sich somit in den Bereich der Tertiärspeicher. In solche neuen Speicherverfahren muss allerdings noch viel Entwicklungsarbeit gesteckt werden. Einen großen Einfluss hat vor allem die Entwicklung kostengünstiger Speichermaterialien, um günstige €/GByte-Preise zu erreichen. Die Zukunft wird zeigen, ob holographische Speicher eine Breitenanwendung finden oder andere Speichertechnologien die Speicherprobleme der Zukunft lösen [CWTH02, CZR01, Mech01]. Ein Prototyp einer holographischen Speicherscheibe HVD (*Holographic Versatile Disc*) mit 200 GByte Speicherkapazität, wurde im September 2004 von der japanischen Firma Optware vorgestellt [Optw04].

Gegenwärtig spielen optische und magnetooptische Speichermedien im HPC-Bereich nur eine untergeordnete Rolle. Zum Speichern großer Kapazitäten werden vor allem Magnetbänder eingesetzt. Im nächsten Kapitel wird die Frage geklärt, ob Magnetbänder durch Festplattenspeicher abgelöst werden.

2.2.3 Sind Magnet-Bänder noch zeitgemäß?

Bei den gegenwärtigen Rekordmeldungen und Steigerungsraten der Festplattenkapazitäten, bei gleich bleibenden oder sogar fallenden Preisen, drängt sich die Frage auf, ob Magnetbänder noch zeitgemäß sind oder als langsames und veraltetes Medium ausrangiert werden sollten. Diese Frage wird oft spontan als nicht mehr zeitgemäß beantwortet. Bei genauerer Betrachtung wird jedoch sehr schnell klar, dass keine schnelle und pauschale Beurteilung möglich ist, da selbstverständlich auch die Entwicklung neuer Magnetbänder weiter fortgeschritten ist. Abbildung 2.7 zeigt die Preisentwicklung von Magnetbänder (Tape-Media), Festplatten (SCSI-HDD), Tertiärspeichersysteme (Tape Library) und RAID-Systeme (Enterprise RAID, Redundant Arrays of Independent Disks) in €/GByte der gehobenen Qualitätsklasse [Moor02a, Moor02b, Nall00, Schw02, Sun04a, Sun04b]. Es werden unkomprimierte Magnetbänder (native Kapazität) mit RAID-System Level 0 verglichen.

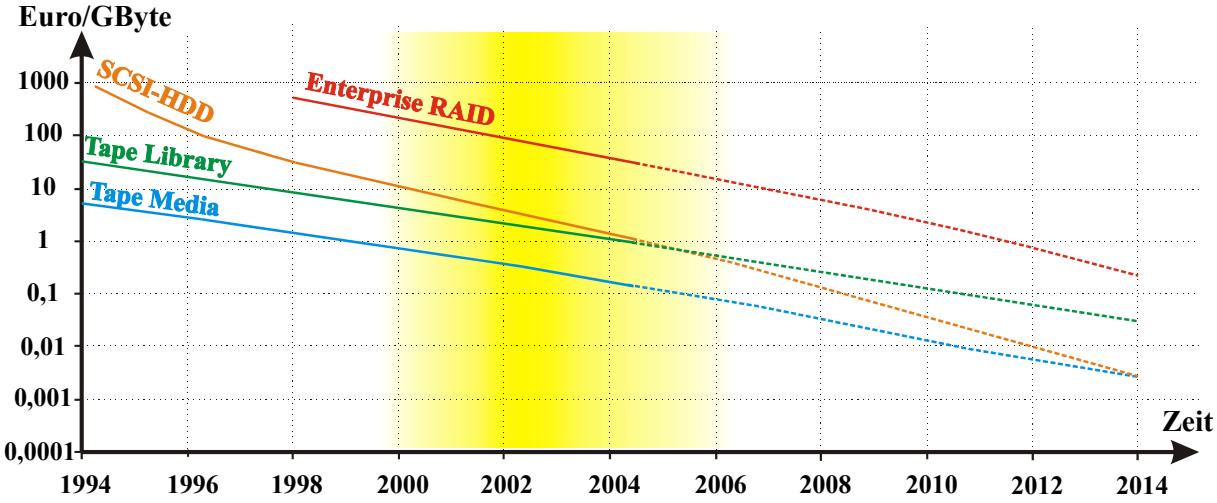


Abbildung 2.7: Preisvergleich Magnetband- und Festplattentechnologie.

Ein Super-DLT Magnetband mit 160 GByte (unkomprimiert) von MAXELL kostet derzeit (August 2004) ca. 49,-- €, was einem €/GByte-Preis von 0,30 €/GByte entspricht [PCS04, Ingr04]. Wird ein realistischer Komprimierungsfaktor von 1,6 angenommen, werden sogar nur 0,19 €/GByte erreicht. Der Preis einer U320-SCSI Festplatte mit 180 GByte von Maxtor kostet ca. 1500,-- €, was zu einem €/GByte-Preis von 5,-- €/GByte führt [Alte04, Ingr04]. Eine vergleichbare IDE (*Integrated Device Equipment*) Festplatte von Maxtor kostet hingegen nur ca. 279,-- €, mit einem €/GByte-Preis von 0,60 €/GByte. Die Preise der IDE-Festplatten liegen annähernd in Regionen der Preise von Magnetbändern. Allerdings werden in RAID-Systemen mit größerem Datenvolumen, ab einer mittleren Qualitätsstufe nur SCSI Festplatten verwendet.

Wird nur die Preisentwicklung von Magnetbändern und SCSI-Festplatten betrachtet, so wird der €/GByte-Preis einer Festplatte den eines Magnetbandes in den kommenden zehn Jahren unterschreiten. Allerdings müssen, um einen objektiven Vergleich zu erhalten, Komplettsys-

teme verglichen werden. In diesem Fall unterschreiten die €/GByte-Preise von RAID-Systemen auch in zehn Jahren noch lange nicht die €/GByte-Preise der TS-Systeme. Diese unterschiedliche Entwicklung wird erklärt durch die Tatsache, dass der €/GByte-Preis bei TS-Systemen mit steigender Anzahl Magnetbänder günstiger wird, da im Wesentlichen nur mehr Platz für die Aufnahme der zusätzlichen Magnetbänder benötigt wird und die Schreib/Lese-Stationen bzw. Roboterarme weiterhin ausreichen. Bei RAID-Systemen ist das etwas anders, da nicht so viele Festplatten pro Speichersystem enthalten sein können. Um ein höheres Speichervolumen zu erreichen, müssen also mehrere teuere Speichersysteme mit entsprechender Hardware (z.B. SCSI-Controller) kombiniert werden, was den €/GByte-Preis in die Höhe treibt. Tabelle 2.3 zeigt eine Preisübersicht von Tape-Systemen (Magnetbandsystemen) und SCSI RAID-Systemen [SM04, PCS04, Ingr04, Sun04a, Sun04b]. Bei steigendem Datenvolumen sinkt der €/GByte-Preis bei Tape-Systemen und steigt bei SCSI RAID-Systemen. Dies spiegelt sich auch im Verlauf des prozentualen Preisvorteils von Tape-Systemen gegenüber RAID-Systemen wider. Bei einem Datenvolumen von 1 TByte liegt der Preis von Tape-Systemen bei 27,9 % gegenüber RAID-Systemen. Bei 50 TByte Datenvolumen liegen die Kosten nur noch bei 1,10 %.

Datenvolumen	Tape-System Preis	RAID-System Preis	Tape-System €/GByte	RAID-System €/GByte
1 TByte	4.838,-- €	17.313,-- €	4,72 €/GByte	17,31 €/GByte
10 TByte	20.075,-- €	550.297,-- €	1,96 €/GByte	53,74 €/GByte
20 TByte	22.982,-- €	1.121.510,-- €	1,12 €/GByte	54,43 €/GByte
50 TByte	30.848,-- €	2.803.775,-- €	0,62 €/GByte	54,76 €/GByte
125 TByte	74.450,-- €	-	0,58 €/GByte	-
1.916 TByte	420.507,-- €	-	0,21 €/GByte	-

Tabelle 2.3: Preisübersicht Tape-Systeme vs. SCSI RAID-Systeme (Stand August 2004).

Ein weiterer wichtiger Punkt, der für Magnetbänder als Speichermedium spricht, ist das jährlich wachsende Datenvolumen. Studien gehen davon aus, dass in den nächsten fünf Jahren das Datenvolumen um den Faktor 10 bis 20 steigt [Moor02a, LVDS03]. Diese Abschätzung wird auch von den HPC-Partnern des ESTEDI-Projektes geteilt, allerdings als eher konservativ angesehen. Aufgrund des immensen Datenvolumens, führt im High Performance Computing Bereich, kein Weg an der Speicherung von Daten auf Magnetbändern vorbei. Gerade bei Applikationen im Bereich historischer Patientendaten und Erdbeobachtungsdaten, werden zukünftig Datenbestände von Exabyte (10^{18} Byte) und sogar Zettabyte (10^{21} Bytes) erwartet [CKK00].

Allerdings kann keine pauschale Entscheidung für und wider Magnetbänder getroffen werden. Grundsätzlich sind das jeweilige Einsatzgebiet und das Datenvolumen entscheidend. Ist ein schneller Zugriff auf große Datenvolumen bis ca. 20 TByte notwendig, so können nur sehr teuere RAID-Systeme eingesetzt werden. Steigt das Datenvolumen allerdings über 20 TByte

an, so gibt es keine Alternative zu robotergesteuerten TS-Systemen. Performante und teuere RAID-Systeme werden meist nur in kleinerer oder mittlerer Ausprägung ($\leq$ 3 – 50 TByte) unterstützend als Cachesystem verwendet, um teilweise langsame Tertiärspeicherzugriffe zu vermeiden. Es wird sich zeigen, ob neue Speichertechnologien, wie zum Beispiel holographische Speichersysteme, irgendwann Magnetbänder ablösen werden. Eine Markteinführung neuer Speichertechnologien innerhalb der nächsten zehn Jahre ist eher unwahrscheinlich. Bis sich diese Systeme auf dem Markt etablieren, vergehen noch weitere Jahre.

2.2.4 Hierarchische Speichermanagement Systeme

Im professionellen Umfeld ist es notwendig, automatisierte Systeme einzusetzen, um Daten auf TS-Medien, wie z.B. Magnetbänder, auszulagern und im Bedarfsfall zurückzuholen. Manuelles Eingreifen ist sehr arbeits- und wartungsintensiv und ab einer gewissen Datenmenge bzw. Medienanzahl nicht mehr vertretbar. Mit den *hierarchischen Speichermanagement* (HSM) Systemen ist seit den achtziger Jahren eine automatisierte Lösung der Datenspeicherung auf TS-Medien vorhanden, die ursprünglich im Großrechnerbereich entstanden ist. Solche Systeme sind seit längerem auch für Unix, Linux und Windows Betriebssysteme erhältlich. Ein HSM-System erweitert lokale Festplatten um die Kapazität einer Vielzahl tertiärer Speichermedien und kann als normales Dateisystem mit nahezu unbegrenzter Speicherkapazität angesehen werden. In Wirklichkeit ist das virtuelle Dateisystem eines HSM-Systems, unterteilt in einen begrenzten Festplattencache auf dem der Anwender arbeitet (Schreiben und Lesen von Daten) und einem angegliederten TS-System, mit einer robotergesteuerten TS-Medienbibliothek. Je nach Wunsch und somit Ausstattung des HSM-Systems, können beliebige tertiäre Speichermedien, wie Magnetband, MOD, CD, DVD usw., verwendet werden. Abbildung 2.8 zeigt die Architektur eines solchen hierarchischen Speichermanagement Systems. Es lässt sich gut die Speicherhierarchie erkennen, bestehend aus Festplatte (Cache) und TS-System. Ein HSM-System erweitert somit die bereits bestehende Speicherhierarchie der Primär- und Sekundärspeicher um die Komponente der Tertiärspeicher.

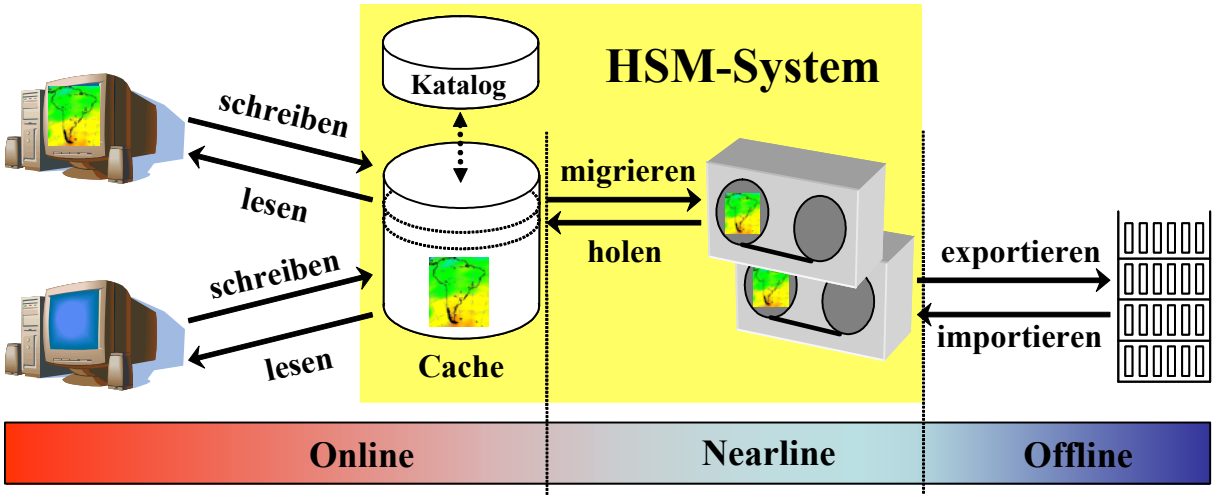


Abbildung 2.8: Architektur eines hierarchischen Speichermanagement Systems.

Hierarchisches Speichermanagement wird von Herstellern in unterschiedlichen Ausprägungen angeboten. Die wesentliche Aufgabe jedes HSM-Systems ist die automatische Migration der Daten vom Festplattencache auf ein tertiäres Medium und das Zurückholen im Bedarfsfall. Die gesamte Komplexität eines HSM-Systems bleibt dem Nutzer verborgen. Lediglich die Zeitverzögerung beim Lesezugriff auf die ausgelagerten Daten, ist im Vergleich zu lokal gespeicherten Daten sehr viel größer. Aus der Sicht des Nutzers, ist das Schreiben von Daten in ein HSM-System ähnlich performant, wie das Speichern von Daten auf der lokalen Festplatte, da die Daten nur in den Festplattencache geschrieben werden. Das Migrieren der Daten, z.B. auf Magnetband, erfolgt anschließend automatisch durch das HSM-System. Üblicherweise werden Dateien eines Dateisystems verwaltet. Allerdings ist es auch denkbar, andere Objekttypen, wie Datenbanktabellen, auszulagern (siehe Kapitel 2.4 und Kapitel 3.2). Für die Verwaltung der in einem HSM-System gespeicherten Daten wird ein Systemkatalog verwendet. In diesem Katalog sind für jede gespeicherte Datei Metadaten abgelegt, wie z.B. ursprünglicher Speicherort, Speichermedium, Besitzer, Zugriffsrechte und Erstellungsdatum. Ein HSM-System bietet dem Benutzer Transparenz auf zwei Ebenen: Die Gerätetransparenz erlaubt es dem Benutzer über eine gleich bleibende Schnittstelle (Festplattencache) auf seine Dateien zuzugreifen, unabhängig vom verwendeten Tertiärspeichergerät (DLT, LTO, MOD, DVD). Zum anderen wird dem Benutzer durch die bestehende Speicherorttransparenz der Zugriff auf Dateien ermöglicht, ohne dass er wissen muss, auf welchem Medium sie gespeichert wurden. Ein HSM-System suggeriert einem Nutzer, bzw. einer Applikation, dass Daten immer lokal (HSM-Cache) gespeichert und somit sofort verfügbar sind. Ein Schlüsselkonzept eines HSM-Systems, ist der Balanceakt zwischen schnellem Zugriffsverhalten und günstigem Speicherplatz.

Verfügbarkeit von Daten

In Bezug auf die Zugreifbarkeit (Verfügbarkeit) von Daten lassen sich die Bereiche Online, Nearline und Offline abgrenzen. Daten, die auf der Festplatte gespeichert sind und auf die sehr schnell zugegriffen werden kann, werden als Online-Daten bezeichnet. Ein entsprechendes Online-Speichergerät wird auch als *Direct Access Storage Device* (DASD) bezeichnet. Wurden Daten auf ein TS-Medium migriert, die sich in einer robotergesteuerten Bibliothek befinden, so spricht man von Nearline-Daten. Die Zugriffszeit auf diese Daten ist sehr viel langsamer als auf Online-Daten. Allerdings ist keine Nutzerinteraktion für das Datenretrieval notwendig. Offline-Daten hingegen sind auf TS-Medien gespeichert. Sie sind allerdings nicht in automatisierten Bibliotheken integriert, sondern zum Beispiel in externen Regalen gelagert. Um diese Daten zu nutzen, muss ein Systemadministrator, nach Aufforderung des HSM-Systems, die Magnetbänder manuell in das HSM-System einfügen. Um auf Offline-Daten zuzugreifen, ist also eine Nutzerinteraktion notwendig.

Arten der Datenhaltung

Entsprechend der Art der Datenhaltung auf TS-Medien, wird zwischen Backup, Archivierung und Auslagerung von Daten unterschieden. Von Backup wird gesprochen, wenn eine Sicherungskopie von Daten erstellt wird. Im Falle eines Datenverlustes kann diese Sicherungskopie wieder eingespielt werden. Backup-Medien werden im Nearline- oder Offline-Bereich gela-

gert. Bei archivierten Daten handelt es sich in der Regel um nichtoperative Daten. Sie wurden aus dem System ausgelagert, da sie nicht mehr oder nur noch selten benötigt werden. Archivdaten werden sowohl im Offline-Bereich (nicht mehr benötigte Daten), als auch im Nearline-Bereich (selten benötigte Daten) gehalten. Bei der Datenauslagerung werden operative Daten aufgrund des großen Volumens und nicht wegen der seltenen Verwendung, wie bei der Archivierung, auf TS-Medien gespeichert. Im Gegensatz zum Backup handelt es sich beim Archiv- und Auslagerungsspeicher um Maßnahmen, den Speicherplatz auf der Festplatte zu reduzieren und günstigere Speichermedien zu nutzen. Allerdings wird meist von Daten, die auf Magnetband archiviert, bzw. ausgelagert wurden, ein Backup dieses Mediums auf einem weiteren Magnetband erstellt. Treten beim Lesen eines Bandes Fehler auf, so wird automatisch auf das Backupmedium zurückgegriffen und ein neues „Original-Band“ erstellt. HSM-Systeme unterstützen im Allgemeinen alle drei Varianten der Datenhaltung.

Auslagerungsstrategien

In modernen HSM-Systemen gibt es unterschiedliche Auslagerungsstrategien, die je nach Verwendungszweck gewählt werden können. Ziele sind oft, möglichst geringe Zugriffszeiten oder eine möglichst hohe Gesamtkapazität zu erhalten. Ersteres lässt sich durch das Migrieren der am wenigsten benötigten Daten erreichen, während letzteres die Auslagerung der größten Dateien erfordert. Mischformen entstehen durch eine Gewichtung beider Kriterien. Mögliche Auslagerungsstrategien gehen von einer explizit durch versierte Nutzer, bzw. Applikationen angestoßenen, bis hin zur vollautomatischen Auslagerung, die komplexen Regeln folgt. Dies wird meist durch einen Hintergrundprozess des Dateisystems mit einer prophylaktischen Kandidatensuche bewerkstelligt. Erreicht der Füllgrad der Festplatte einen bestimmten oberen Schwellwert, werden diese Kandidaten ausgelagert. Bei Erreichen einer zweiten, unteren Schwelle, wird die automatische Migration beendet. Somit wird ein Überlauf der Festplatte vermieden. In einem Unix-Dateisystem selbst, verbleibt anstelle der ausgelagerten Datei ein Verweis (Stub-File oder Anchor) im ersten Datenblock der ursprünglichen Datei. Es sind Einträge (Metadaten), wie ursprüngliche Dateigröße, Zugriffsrechte, Eigentümer usw., weiterhin enthalten und für den Anwender sichtbar.

Realisierungsmöglichkeiten von HSM-Funktionalität

Eine Einbindung der HSM-Funktionalitäten, kann auf Betriebssystemebene in einem Unix-System, durch die integrale Erweiterung eines Dateisystems, durch eine Datenmanagementerweiterung des Betriebssystemkernels oder durch die Verwendung der *X/Open Data Storage Management* API (XDSM) erfolgen. Bei der integralen Erweiterung, muss den Dateisystemtreibern eine zusätzliche Logik für die Ein- und Auslagerung von Dateien auf TS-Systemen hinzugefügt werden. Somit entsteht ein neuer Dateisystemtyp, den Unix über die virtuelle Dateisystemschicht (VFS-Layer, *Virtual File System Layer*) an die Systemschnittstelle anbindet. Die Lösung ist auf diesen einen Dateisystemtyp eingeschränkt. Der Hersteller ist gezwungen, das erweiterte Dateisystem auf jede Unix-Variante zu portieren, um HSM zu nutzen. Ein HSM-System dieser Variante ist der von Sun Microsystems (früher LSC Incorporation) angebotene *Storage Archive Manager* (SAM) mit den unterstützten Dateisystemen *Quick File System* (QFS) und *SAM File System* (SAM-FS) [Sun02, Akel01].

Einen anderen Ansatz verfolgt die Realisierung der HSM-Unterstützung durch eine Kernelerweiterung. Hier wird zwischen der virtuellen Dateisystemschicht und dem Dateisystemtreiber eine weitere Schicht eingebracht. Diese Zwischenschicht ist nach außen hin als neuer HSM-Dateisystemtyp sichtbar. Eingehende Systemaufrufe (z.B. open, close) werden vom VFS-Layer an das HSM-Kernelmodul weitergeleitet. Für residente Dateien wird dann direkt der Dateisystemtreiber aufgerufen. Handelt es sich um Daten, die auf den Tertiärspeicher ausgelagert sind, leitet das HSM-Kernelmodul den Zugriff durch einen Pseudogerätetreiber von Unix an einen Hintergrundprozess weiter: Der übernimmt das Zurückholen der Datei. Erst nach diesem Schritt greift das Dateisystem, wie gewohnt, auf die jetzt wieder lokal gehaltenen Daten zu. Bei der Realisierung dieses Konzeptes bleibt das Dateisystem selbst unverändert. Allerdings muss der Kernel angepasst werden. Als Beispiel dieser Art der Realisierung ist der *Tivoli Space Manager* (TSM) von IBM Corporation unter dem Betriebssystem AIX zu erwähnen [IBM03a, Akel01].
Die dritte und letzte Möglichkeit verfolgt den Weg einer standardisierten Unterstützung der HSM-Funktionalität. Im Jahr 1993 schlossen sich IBM, Legato, Novell, Sun, Veritas und andere zur *Data Migration Interface Group* (DMIG) zusammen und brachten bereits 1994 die *Data Management API* (DMAPI) hervor. Diese DMAPI wurde 1997 von der Open Group unter dem Namen X/Open Data Storage Management zum Standard erhoben [OGS97]. Die HSM-Funktionalität läuft als Hintergrundprozess (Dämon) und XDSM realisiert die Anbindung an den Betriebssystemkernel. Der HSM-Dienst erscheint aus der Sicht der XDSM-Schicht als Applikation, die durch eine Data-Management-Session den Dateizugriff anderer Applikationen steuert. Erfolgt von einer Applikation ein Zugriff auf eine ausgelagerte Datei, wird eine Nachricht an den HSM-Dämon gesendet. Daraufhin wird die gewünschte Datei vom TS-System geladen und in das lokale Dateisystem gespeichert. Solange dieser Vorgang dauert, wird der eigentliche Systemaufruf blockiert. Nach der Freigabe erfolgt ein ganz normaler Dateizugriff. Dieses standardisierte XDSM basierte Konzept, erleichtert und beschleunigt die Portierung der HSM-Unterstützung auf andere Unix Derivate und Dateisysteme. Auf diese Art und Weise wurde die HSM-Funktionalität des Tivoli Space Manager für das *Veritas File System* (VxFS) und FileServ von Adic Incorporation auf Solaris umgesetzt [IBM03a, Adic03, Akel01].

Um der extremen Datenflut im Bereich von mehreren Petabyte zu begegnen, ist es notwendig, die zugrunde liegende Hardwarestruktur entsprechend auszulegen. Eine geeignete Möglichkeit bietet die Parallelität der zugrunde liegenden Hardware, mit einer entsprechenden Anzahl von Bandlaufwerken. Um einen schnellen Zugriff auf Dateien zu ermöglichen, wurde das *General Parallel Files System* (GPFS) entwickelt. Für das Betriebssystem AIX ist von IBM ein Tivoli Space Manager erhältlich, der auf GPFS basiert. Die HSM Unterstützung wurde mittels XDSM implementiert. Ein HSM-System mit noch größerer Skalierbarkeit ist das *High Performance Storage System* (HPSS), das in einem Forschungsprojekt von IBM entstand. Durch eine Bündelung vieler Netzwerkverbindungen, können sogar mehrere Gigabyte pro Sekunde übertragen werden. Die Architektur stützt sich dabei auf das *IEEE Mass Storage System Reference Model* (MSSRM), Version 5. Allerdings existieren weltweit nur sehr wenige Systeme, die auf HPSS basieren [HPSS03].

Auch der Einsatz von hierarchischem Speichermanagement in Kombination mit *Storage Area Networks* (SAN) stellt keinen Widerspruch dar, sondern ergänzt sich. Durch die Verwendung eines eigenen Speichernetzwerkes, wird der Transfer von Massendaten aus dem lokalen Rechnernetz (*Local Area Network*, LAN) in das SAN verlagert. SAN-Umgebungen werden vor allem aus dem Gesichtspunkt der gestiegenen Anforderung an die Hochverfügbarkeit der gespeicherten Daten betrieben.

Neben den bereits erwähnten HSM-Systemen SAM, TSM und FileServ, sind noch weitere Systeme, wie zum Beispiel DiskXtender von Legato Systems Incorporation (früher UniTree von UniTree Software Incorporation) und StorHouse Storage Manager von FileTek Incorporation auf dem Markt erhältlich [Sun02, IBM03a, Adic03, CB00, Lega03b]. Auch bei der Kopplung von TS-Systemen und DBMS, werden teilweise HSM-Systeme eingesetzt. Die unterschiedlichen Möglichkeiten der Kopplung von DBMS und Tertiärspeicher werden in Kapitel 2.3 präsentiert.

2.3 Kopplung von Tertiärspeichersystem und DBMS

Dieses Kapitel beschäftigt sich mit der Kopplung von DBMS und Tertiärspeichersystemen. Zunächst sind in Abbildung 2.9 die verschiedenen Möglichkeiten der Anbindung von Tertiärspeicher dargestellt.

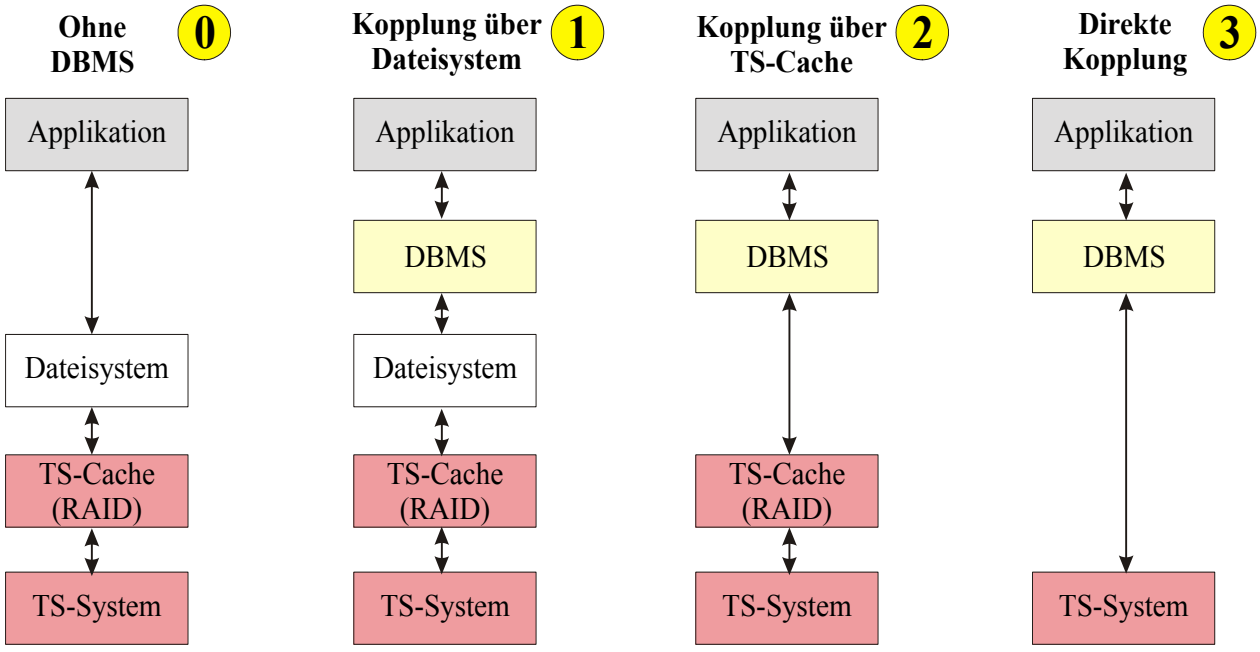


Abbildung 2.9: Möglichkeiten der Anbindung von Tertiärspeicher.

Links ist die direkte Verwendung von TS-Systemen über ein Massenspeicher-Dateisystem zu erkennen. HSM-Systeme werden durch eine automatisierte Kombination aus TS-Cache (meist realisiert als RAID) und TS-System realisiert. Dateien, die Anwender oder Applikationen im

Dateisystem speichern, werden nach bestimmten Regeln zunächst in den TS-Cache transferiert und anschließend automatisiert auf TS-Medien geschrieben, bzw. bei Dateizugriffen geladen (Kapitel 2.2.4). Um DBMS mit Tertiärspeicherkomponenten zu koppeln, sind grundsätzlich drei Arten zu unterscheiden: Einerseits können DBMS über ein Dateisystem mit Tertiärspeicher gekoppelt werden (Abbildung 2.9, Variante 1). Diese Systeme nutzen das darunter liegende Dateisystem, um einen transparenten Zugriff zu den auf Tertiärspeicher abgelegten Daten zu realisieren. Zusätzliche Mechanismen des Dateisystems (installierter HSM-Dämon), kontrollieren die Verschiebung der Daten zum/vom TS-Cache und zum/vom Tertiärspeicher. Die zweite Möglichkeit der Tertiärspeicheranbindung an ein DBMS, zeigt die Variante 2 der Abbildung 2.9. In diesem Fall ist das DBMS nur über den TS-Cache mit dem TS-System gekoppelt. Daten des DBMS können direkt in den TS-Cache geschrieben werden. Diese Art der Realisierung ist bei konventionellen Systemen sehr häufig zu finden. Die dritte Möglichkeit der Anbindung, wird durch eine direkte Kopplung eines DBMS mit einem TS-System erreicht (Abbildung 2.9, Variante 3). Durch diese enge Kopplung kann das DBMS, verglichen mit den anderen Möglichkeiten, den größten Einfluss auf die Speicherung der Daten auf TS-Medien nehmen. Dies bietet vielseitige Möglichkeiten der Optimierung. Nachteilig wirkt sich allerdings aus, dass die komplette Funktionalität des Lesens und Schreibens von Daten in das DBMS integriert werden müssen. Der Aufwand für die Unterstützung weiterer TS-Systeme ist sehr hoch. Aus diesem Grund ist diese Art der Kopplung meist nicht geeignet.

In Kapitel 2.4 folgt ein Überblick über den aktuellen Stand der Forschung im Bereich der Kopplung von TS-Systemen und DBMS.

2.4 Stand der Forschung

Zur Zeit unterstützt keines der auf dem Markt führenden DBMS, wie Oracle, IBM/DB2 Universal Database und Microsoft SQL-Server, eine direkte Anbindung an Tertiärspeicher (ausgenommen Backup). Allerdings gibt es Drittanbieter, die eine Anbindung von DBMS an bestehende HSM-Systeme realisieren. Um einen Überblick über den Stand der Forschung und Entwicklung im Bereich der Anbindung von Tertiärspeicher an DBMS zu erhalten, werden beispielhaft drei Entwicklungen vorgestellt und auf essentielle Defizite und Probleme dieser Systeme eingegangen. Zunächst wird das konventionelle System StorHouse/RM der Firma FileTek beschrieben. Danach folgt das 1995 aus der Not in Eigenentwicklung entstandene System CERA (*Climate and Environmental Data Retrieval and Archiving System*). Das Deutsche Klimarechenzentrum in Hamburg wollte damit dem Problem der großen Datenflut begegnen. Abschließend folgt das an der Universität Berkeley (USA) entwickelte DBMS Postgres, mit der prototypischen Erweiterung eines Speichermanagers für Tertiärspeicher. Die hier beschriebenen Systeme werden vor allem aus der Sicht der Speicherung und des Datenretrievals von multidimensionalen Array-Daten beleuchtet und bewertet. Eine umfassende Beschreibung ist im technischen Bericht „Fusion von Datenbanktechnologie und Tertiärspeichersystem im Bereich sehr großer multidimensionaler Array-Daten – Stand der Forschung“ [Rein03b] zu finden. Dort werden weitere Systeme wie SDM (*Scientific Data Manager*),

MDM (*Meta-Data Management System*) und APRIL (*A Parallel Run-Time Library for Tape-Resident Data*) vorgestellt und bewertet.

2.4.1 FileTek StorHouse/RM

Das System Storhouse der Firma FileTek realisiert eine Komplettlösung für das Speichern, Verwalten und Auffinden von großen Datenvolumen und bietet neben der Funktionalität eines HSM-Systems die Möglichkeit, DBMS anzubinden. StorHouse bietet Schnittstellen zu einigen kommerziellen DBMS, wie Oracle, IBM/DB2 Universal Database, Microsoft SQL-Server und Sybase. Es ist möglich, mit der DBMS-Erweiterung von StorHouse, Daten unabhängig von deren Speicherort mittels SQL (*Structured Query Language*) abzufragen und zu bearbeiten. Im Wesentlichen können die zwei Komponenten StorHouse/SM (*StorHouse Storage Manager*) und StorHouse/RM (*StorHouse Relational Manager*) unterschieden werden.

Der Speichermanager StorHouse/SM realisiert die Funktionalität eines HSM-Systems und verwaltet somit eine mehrschichtige Speicherarchitektur aus Hauptspeicher, RAID, Festplatten und Tertiärspeichermedien. Mittels der Verwaltungskomponente VSAC (*Volume Storage Allocation and Control*) des StorHouse/SM, lassen sich Daten nach bestimmten Zugriffsmustern kombinieren, um sie gemeinsam auf TS-Medien zu schreiben. Durch die Generierung von Dateifamilien, wird die Speichergranularität erhöht. Dadurch wird versucht, bei Zugriffen die notwendigen Suchoperationen zu reduzieren. StorHouse/SM unterstützt weitere Zugriffsoptimierungen, wie das Zusammenfassen von Zugriffen auf ein einzelnes TS-Medium, um hohe Medienwechselkosten zu minimieren. Auch die Reihenfolge der Zugriffe auf ein Magnetband, werden in der Weise optimiert, dass ein serielles Lesen der Daten ermöglicht wird. Dies reduziert teure Such- und Spulkosten.

Die Anbindung kommerzieller DBMS an Tertiärspeicher erfolgt durch den StorHouse Relational Manager. Diese Komponente StorHouse/RM ist ein eigenständiges relationales DBMS, das mit Hilfe des StorHouse/SM die Anbindung an Tertiärspeicher realisiert. Die Anbindung zum Beispiel von Oracle an StorHouse/RM, wird mittels Datenbanklinks und einer ODBC (*Open Database Connectivity*) Schnittstelle bewerkstelligt. Dies bedeutet, dass Tabellen, die in StorHouse liegen, in Oracle mittels Zeiger repräsentiert werden. Eingehende Anfragen werden von Oracle analysiert. Entsprechende Unteranfragen werden automatisch an StorHouse/RM weitergeleitet, wenn dies notwendig ist. Die Verbindung von Oracle zu StorHouse/RM ist für den Anwender transparent. Abbildung 2.10 zeigt das Prinzip der Anfragebearbeitung mittels Oracle in Kombination mit StorHouse/RM. Die Kopplung von StorHouse/RM und IBM/DB2, bzw. Microsoft SQL-Server, erfolgt auf ähnliche Weise [File03b, File03c].

Zur Beantwortung der Anfrage eines Anwenders (1) werden die beiden Tabellen CUSTOMERS und TRANSACTIONS benötigt. Oracle parst nun diese Anfrage und stellt fest, dass die Tabelle TRANSACTIONS über einen Datenbanklink ausgelagert wurde. Daraufhin werden aus der ursprünglichen Anfrage (Query) Unteranfragen generiert, die entsprechend verteilt werden (2a, 2b). Oracle selbst verarbeitet die Unteranfrage an die Tabelle

CUSTOMERS (3a) und StorHouse/RM bearbeitet die Unteranfrage an die Tabelle TRANSACTIONS (3b). Nach der Ausführung gibt StorHouse/RM das Teilergebnis der Unteranfrage an Oracle zurück (4). Oracle bildet nun einen Verbund (Join) der beiden Teilergebnisse und liefert das Gesamtergebnis der Anfrage an die Applikation zurück (5).

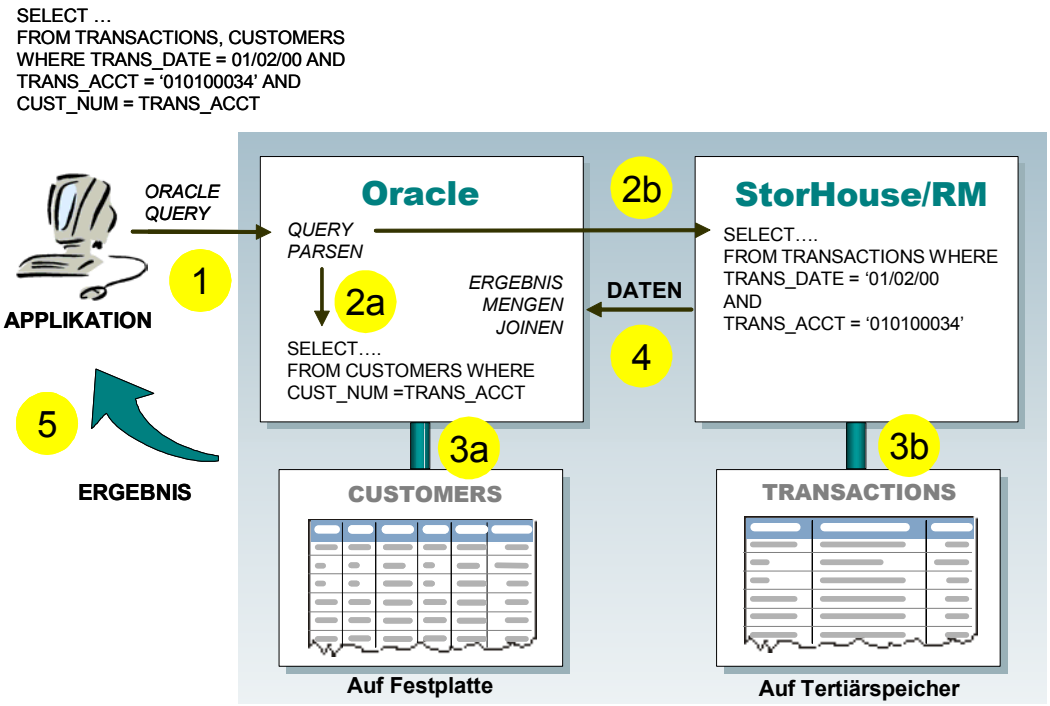


Abbildung 2.10: Anfragebearbeitung mittels Oracle und StorHouse/RM [nach File03a].

Der Aufbau von StorHouse/RM entspricht aus logischer Sicht der klassischen Architektur eines relationalen DBMS mit Tablespaces, Benutzertabellen, Systemtabellen und Indices. Physikalisch werden Benutzertabellen und Indizes als StorHouse/SM-Dateien in der vorhandenen Speicherhierarchie gespeichert. Tabellen werden automatisch aufgrund ihrer Größe, in eine bestimmte Anzahl Segmente unterteilt. Durch das Einfügen neuer Daten in die Tabellen, können neue Segmente entstehen. Alle diese Nutzertabellen-, Systemtabellen- und Index-Segmente, werden als separate Dateien in StorHouse/SM gespeichert. Wie auch in kommerziellen DBMS, können in StorHouse/RM beliebige Objekte, wie z.B. Bilder oder Videos in LOBs (*Large Objects*) gespeichert werden. Analog zu Tabellen, werden je nach Größe eine gewisse Anzahl an LOBs, zu einem Segment zusammengefasst und als Datei gespeichert.

Durch die Möglichkeit einer horizontalen und vertikalen Partitionierung der Daten auf logischer Ebene, kann eine flexiblere Datenverteilung auf unterschiedliche Medien erfolgen. Bei der horizontalen Partitionierung, können verschiedene Zeilen der gleichen Tabelle auf unterschiedliche Speichermedien geschrieben werden. Diese Partitionierung kann eingesetzt werden, um weniger häufig angefragte Einträge einer Tabelle (d.h. historische Daten), auf günstige Speichermedien zu verschieben. Bei der vertikalen Partitionierung werden ein oder mehre-

re Spalten einer Tabelle auf verschiedene Speichermedien abgelegt. Dadurch können Spalten, auf die selten zugegriffen wird, oder deren Inhalt sehr groß ist (z.B. LOB-Spalte), auf TS-Medien geschrieben werden.

Vergleicht man diese Art der Anbindung eines DBMS an Tertiärspeicher mit Abbildung 2.9, entspricht dies einer Kopplung über einen TS-Cache (Variante 2). Zusammenfassend lässt sich sagen, dass StorHouse/RM entwickelt wurde, um relationale Daten kostengünstig auf unterschiedliche Ebenen der Speicherhierarchie zu verwalten. Auf diese Daten ist ein selektiver Zugriff möglich. Da StorHouse/RM die Möglichkeit bietet, beliebig organisierte Daten als LOBs (bzw. BLOBs) zu speichern, können auch multidimensionale Array-Daten eingefügt und auf unterschiedlichen Ebenen der Speicherhierarchie abgelegt werden. Operationen auf BLOBs, beschränken sich auf das Schreiben, bzw. das Lesen kompletter Objekte. Es werden also keinerlei weiterführende Auswertungen dieser Daten auf dem DBMS-Server unterstützt. Solche Auswertungen müssen in entsprechenden Applikationen durchgeführt werden. Das bedeutet, StorHouse/RM ist in Bezug auf BLOBs nur ein rudimentärer Speicher- und Transaktionsmanager. Durch ein geeignetes Datenbankschema kann zumindest die Suche bestimmter Objekte über Metadaten (Schlagworte, Simulationsname, usw.) vereinfacht werden.

Systembewertung
Ein entscheidender Nachteil von StorHouse/RM, in Bezug auf das Datenretrieval multidimensionaler Daten, ist die hohe Granularität des Datentransfers. Hier sind die Segmente (Menge von LOB), bzw. die von StorHouse/SM gebildeten Dateifamilien (Menge von Segmenten), als Zugriffsgranularität zwischen StorHouse/RM und dem TS-System zu nennen. Bei Anfragen, die nur ein Objekt betreffen, müssen alle Objekte eines Segmentes, bzw. einer Dateifamilie, übertragen werden. Das führt zu einer erhöhten Transferzeit und zu einer höheren Netzwerkbelastung. Gerade bei der Datenübertragung zwischen Tertiärspeicher und DBMS könnten durch eine geringere Granularität enorme Kosten eingespart werden. Weiterhin gibt es die Zugriffsgranularität zwischen StorHouse/RM und einer Applikation. Die kleinste Granularität in diesem Bereich sind einzelne Objekte (LOBs). Bei Bereichsanfragen auf Teilobjekte kann somit eine Reduktion der Datenmenge erst in der entsprechenden Applikation erfolgen. Die Firma FileTek sieht Data Warehouse Systeme als ein Hauptanwendungsgebiet für ihr Produkt StorHouse/RM. Vor allem historische Daten können hier auf günstige TS-Medien verschoben werden und stehen im Bedarfsfall jederzeit ohne zusätzliche Nutzerinteraktion zur Verfügung. Sollen allerdings große Mengen an multidimensionalen Daten effizient gespeichert werden, so besteht die Notwendigkeit aufgrund des Datenvolumens, auch operative Daten auf TS-Medien auszulagern. Dies erfordert allerdings ein spezielles Augenmerk auf effizientes Datenretrieval. Da StorHouse/RM für relationale Daten entwickelt wurde, gibt es keinerlei Optimierung: Wie z.B. die Möglichkeit, Teilbereiche von Objekten zu laden oder das Ausnutzen von Clustering bei multidimensionalen Objekten.

Nähere Information über StorHouse/RM sind bei [CB00, CB01, CKKB01, CKK00, Rein03b] zu finden. Neben StorHouse/RM sind weitere, ähnliche Systeme auf dem Markt erhältlich. Wie zum Beispiel DiskXtender Database Edition der Firma Legato [Inmo02, Lega03b].

2.4.2 System CERA

Bereits 1995 erkannte das Deutsche Klimarechenzentrum, ein Projektpartner von ESTEDI aus Hamburg, die Notwendigkeit, ihre Datenhaltung auf Magnetbänder neu zu überdenken. Die Datenorganisation als Dateien auf Betriebssystemebene, entsprach nicht mehr den Anforderungen an die Zugriffsperformance und die einfache Nutzbarkeit. Um dem enormen Zuwachs an multidimensionalen Daten zu begegnen, wurde beschlossen, ein semantikorientiertes Retrieval der Daten zu entwickeln. Aus diesen Anforderungen ist das System CERA (Climate and Environmental Data Retrieval and Archiving System) entwickelt worden. Dabei wurde ein Datenbankschema für die Metadaten der Klimasimulationen modelliert. Diese Metadaten enthalten technische Information über das Experiment und Information speziell für die Suche, wie z.B. Schlagworte. Dies ersetzt den dateiorientierten Zugriff durch ein Datenretrieval, das entsprechend der Semantik der gespeicherten Klimadaten formuliert werden kann. Neben den Metadaten, werden nach einer Weiterentwicklung auch aufbereitete Simulationsdaten direkt als BLOBs in Tabellen der Oracle Datenbank gespeichert, um einen schnellen Zugriff auf aktuelle Daten zu erhalten.

Aufgrund der begrenzten Datenbankkapazität, wurde eine Anbindung an das HSM-System DiskXtender der Firma Legato entwickelt. Diese Kopplung entspricht der Variante 2 der Abbildung 2.9. Grundlage hierfür bildet die physikalische Datenorganisation des Oracle-DBMS. In Oracle werden Tabellen und BLOBs in Tablespaces organisiert. Ist das Oracle-DBMS auf einem Dateisystem aufgesetzt (kein Raw-Device), wird ein Tablespace als eine, bzw. mehrere Dateien gespeichert. Um Daten auf Tertiärspeicher auszulagern, wird die Möglichkeit von Oracle genutzt, Tablespaces Online bzw. Offline schalten zu können. Online entspricht dem normalen Status: Daten innerhalb dieser Tablespace können direkt angefragt werden. Ein Tablespace, die Offline gesetzt wurde, ist deaktiviert und Anfragen können von Oracle nicht bearbeitet werden. Diese Offline gesetzten Tablespaces können jedoch durch ein darunter liegendes HSM-System auf Magnetbänder ausgelagert werden. Damit Oracle eine Anfrage auf diese ausgelagerten Daten ausführen kann, wurde ein Storage Broker entwickelt, der diese Tablespace vom Tertiärspeicher lädt und wieder Online schaltet. Danach kann die Anfrage wie gewohnt ausgeführt werden. Ein solcher Storage Broker wurde im CERA-System integriert. Weitere Details über das CERA-System sind bei [Laut95, LT01, Laut02] nachzulesen.

Systembewertung

Das CERA-System bietet durch die enthaltenen Metadaten, eine für den Wissenschaftler einfachere Möglichkeit des Datenretrievals. Ein gravierender Nachteil des CERA-Systems ist die hohe Granularität des Datentransfers zwischen DBMS und Tertiärspeichersystem, da nur komplette Tablespaces auf TS-Medien geschrieben, bzw. bei Anfragen zurückgeholt werden können. Im Normalfall werden mehrere Objekte einer Simulationsreihe in eine Tablespace geschrieben. Bei einer Bereichsanfrage auf eines dieser Objekte, muss zuerst die gesamte Tablespace vom Tertiärspeicher geladen werden. Durch das Laden zusätzlicher, nicht benötigter Daten (Objekte), werden die Transferzeit und die Netzwerkbelastung unnötig in die Höhe getrieben. Als weiterer Nachteil ist zu bewerten, dass es nicht möglich ist, auf Teilbe-

reiche von Objekten zuzugreifen, da das CERA-System nur als Datenlieferant komplette Objekte für weitere Anwendungen realisiert. Will ein Anwender in einer Applikation eine Anfrage formulieren, die einen Schnitt durch mehrere dieser Objekte bedeutet, so müssen vom CERA-System alle betroffenen Objekte (als BLOBs gespeichert) an die Anwendung übertragen werden, nachdem die Tablespace von TS-Medien geholt wurde. Weiterhin werden beim Speichern der Daten auf TS-Medien keine Mechanismen zur Optimierung der Datenzugriffe integriert. So werden die einzelnen BLOBs einer Tablespace einfach als linearisierter Bytestrom auf das TS-Medium geschrieben, ohne z.B. das Clustering multidimensionaler Daten auszunutzen. Das Update von Objekten ist beim CERA-System nicht vorgesehen. Dies würde bedeuten, dass aufgrund der Charakteristik von TS-Medien, eine gesamte Tablespace neu auf TS-Medium gespeichert werden muss, obwohl nur ein Objekt geändert wurde. Auch das Fehlen einer multidimensionalen Anfragesprache innerhalb des CERA-Systems, war ein Kriterium für das Deutsche Klimarechenzentrum, sich an dem Projekt ESTEDI zu beteiligen, um ein verbessertes Gesamtsystem zu erhalten.

2.4.3 Postgres

Ein weiters System, das betrachtet wird, ist das DBMS Postgres, das in den 80er Jahren an der Universität Berkeley entwickelt wurde. 1996 wurde dieses DBMS in PostgreSQL umbenannt und stetig durch die Open Source Initiative weiterentwickelt. Die aktuelle Version PostgreSQL 7.4 wurde im November 2003 freigegeben. Da der Quellcode von Postgres bereits sehr früh offen gelegt wurde, bot sich dieses System an, um eigene Modifikationen und Erweiterungen zu implementieren. In seiner Master Thesis realisierte Michael Olson bereits 1992 eine direkte Kopplung von Tertiärspeichersystemen an Postgres. Diese Arbeit schaffte es allerdings nicht, in den offiziellen Quellcode von Postgres bzw. PostgreSQL integriert zu werden. Allerdings bietet die Erweiterung von Olson einige interessante Ansätze und Konzepte, die hier näher betrachtet werden [Olso92].

Um die Anbindung an tertiäre Speichermedien zu realisieren, wurde Postgres um einen neuen Speichermanager (Storage Manager) und Gerätemanager (Device Manager) für eine Sony CD-ROM Jukebox und eine Exabyte Magnetband Jukebox erweitert (Abbildung 2.11). Um Daten zu lesen oder zu schreiben, kommuniziert der Datenmanager (Data Manager) in dieser Architektur nicht mehr direkt mit den Speichersystemen. Als weitere Schicht wurde der Speichermanager eingeführt, mit der die Speicherhierarchie verwaltet wird. Der Speichermanager hat die Information, welche Speichermedien für die Beantwortung einer Anfrage benötigt werden. Die unterschiedlichen Speichersysteme, bzw. -geräte, werden über eigene Gerätemanager an das System angebunden. Somit ist es möglich, durch die Implementierung eines neuen Gerätemanagers, neue Tertiärspeichergeräte anzubinden. Das zeigt einen Nachteil dieser Realisierung, da ein hoher Implementierungsaufwand notwendig ist, um neue Speichergeräte anzubinden.

Beim Einfügen neuer Datensätze in Postgres, kann angegeben werden, welches Speichermedium verwendet werden soll. Dabei können komplette Spalten einer Tabelle partitioniert

werden. Für jede dieser Partitionen kann ein beliebiger Speicherort gewählt werden. Der entsprechende Gerätemanager allokiert für die jeweiligen Partitionen Speicher in Einheiten von so genannten Extents. Welche somit die kleinste Zugriffsgranularität für Daten sind. Für den Gerätetreiber der Sony Jukebox wurde eine Extentgröße von 256 KByte gewählt, was dem 32fachem einer Datenbankseite (8 KByte) entspricht. Durch die Partitionierung besteht die Möglichkeit, Spalten mit hohem Speicheraufwand, wie zum Beispiel bei Bildern, auf günstige Speichermedien auszulagern. Welche Partitionen sich auf welchem Datenspeicher befinden, wird in einer Systemtabelle verwaltet. Dadurch wird bei Anfragen automatisch der richtige Gerätemanager ausgewählt und die entsprechenden Daten geladen. In Postgres ist es nicht notwendig, neue Einträge und geänderte Einträge sofort auf die jeweiligen Speichermedien zu propagieren. Dies vermeidet zwar einerseits unnötige Transferkosten, kann allerdings die Atomarität der einzelnen Schreibvorgänge verletzen [Olso92].

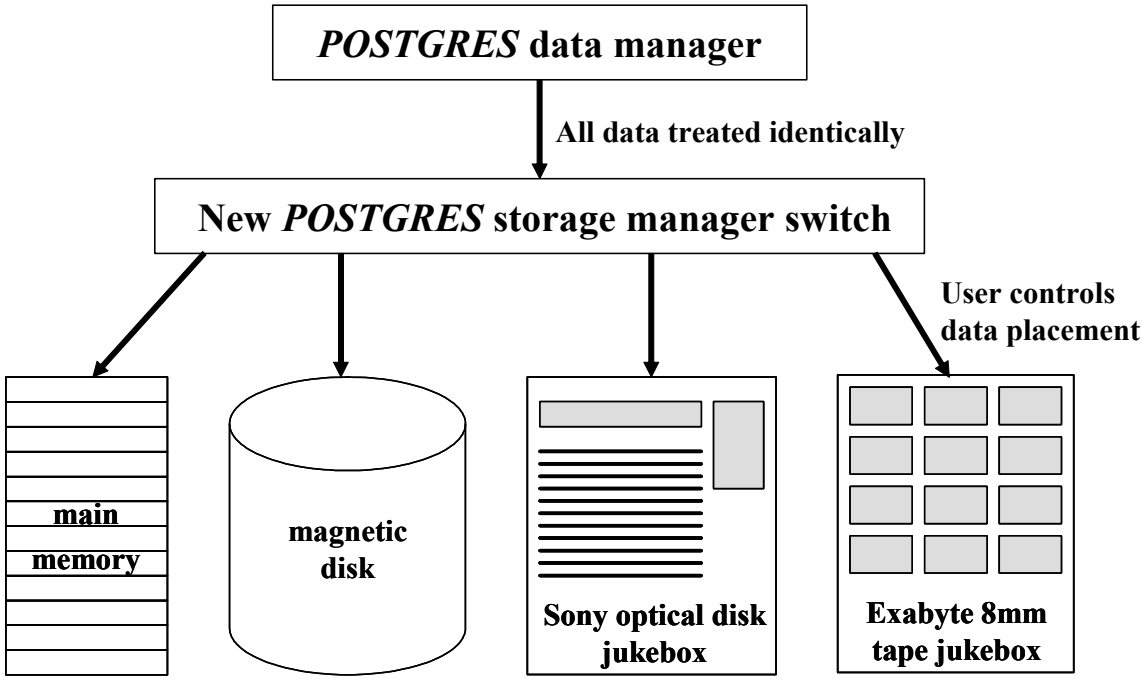


Abbildung 2.11: Architektur mit erweitertem Speichermanager [Olso92].

1993 wurde die Möglichkeit der Speicherung von LOBs in Postgres vorgestellt [SO93]. Diese Entwicklung ebnete den Weg, um multidimensionale Arrays in Postgres zu speichern. Vor allem Sunita Sarawagi nutzte den erweiterten Speichermanager von Postgres und entwickelte einige Konzepte um multidimensionale Array-Daten effizient zu speichern: Zu diesen Strategien zählen *Chunking*, Scheduling und Caching. Beim Chunking wird ein Objekt in mehrere gleichförmige Teilobjekte zerlegt und als einzelne LOB in der Datenbank gespeichert. Abbildung 2.12 zeigt beispielhaft das Chunking eines zwei und drei dimensionalen Objektes.

Dabei wird ersichtlich, dass durch Aufteilung der großen Objekte in mehrere LOBs, bei Anfragen (graue Queryboxen) sehr viel weniger Daten geladen werden müssen. Dies bedeutet

allerdings auf der anderen Seite einen Mehraufwand bei der Verwaltung dieser Objekte. Die Teilobjekte werden entsprechend der Reihenfolge entlang der ersten Dimension auf Magnetband geschrieben. Es muss klar sein, dass ein vom Anwender gewähltes Chunking nicht für alle Anfragenmuster gleich gut geeignet ist. Somit muss das Chunking mit Bedacht gewählt werden, da auch eine Verschlechterung der Zugriffsperformance eintreten kann. Mit Hilfe eines Schedulers wird versucht, die Ausführungsreihenfolge von Anfragen zu optimieren. Dabei wird durch Information über den Cacheinhalt und den benötigten Daten entschieden, welches LOB als nächstes geladen wird, bzw. welche Objekte aus dem Cache verdrängt werden. Nähere Informationen über die Arbeiten von Sarawagi sind bei [Sara96, SS94a, SS94b, Sara95a, Sara95b, Rein03b] zu finden.

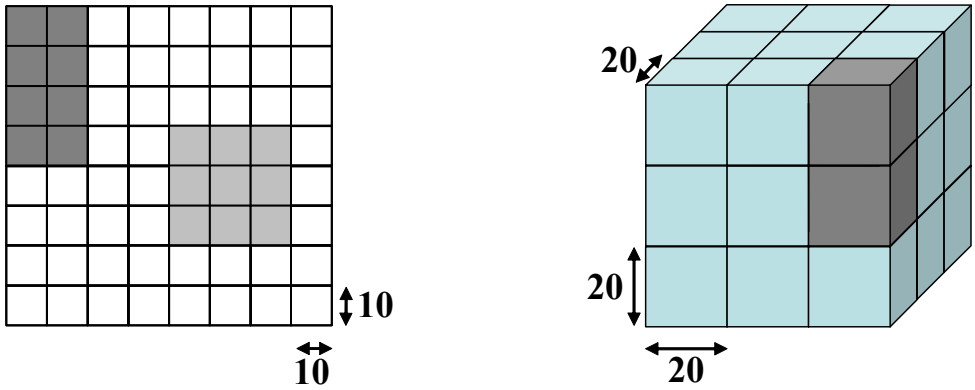


Abbildung 2.12: Mögliches Chunking eines zwei- bzw. drei dimensionalen Arrays.

Systembewertung

Vergleichen wir die hier realisierte Anbindung von Postgres an tertiäre Speichermedien, so entspricht dies der rechten Darstellung in Abbildung 2.9. Postgres ist über die unterschiedlichen Gerätemanager direkt an TS-Systeme gekoppelt. Dies erfordert einen hohen Implementierungsaufwand, um neue Speichergeräte an Postgres anzubinden, da für jedes Gerät ein separater Gerätemanager geschrieben werden muss. Die Arbeiten von Sarawagi ermöglichen es, multidimensionale Daten in einer aktiven Speicherhierarchie zu verwalten. Einen enormen Vorteil bietet das Konzept des Chunking von Objekten. Allerdings ist Chunking nicht für alle Zugriffsmuster optimal geeignet, da es nur gleichförmige Teilobjekte zulässt. Das allgemeinere Tiling (Kachelung)-Konzept würde mehr Möglichkeiten bieten (vergleiche Kapitel 2.5.3).

Beim Speichern der Chunks (LOBs) auf tertiären Speichermedien wird zwar eine „optimale“ Ordnung gewählt: Allerdings wird immer entlang einer Array-Achse geordnet und somit könnte das multidimensionale Clustering verbessert werden. Ein weiterer, vernachlässigter Punkt ist die Zugriffsgranularität auf die multidimensionalen Daten. Es wird keine unterschiedliche Granularität für die Speicherung auf Tertiärspeicher und im DBMS realisiert. Wird eine große Granularität gewählt, um besseres Zugriffsverhalten für Tertiärspeicher zu bekommen, ist diese nicht optimal für die interne Struktur von Postgres. Weiterhin wurde keine multidimensionale Anfragesprache integriert, um multidimensionale Daten direkt im

DBMS zu bearbeiten. Es können nur Teilbereiche eines Objektes aus der Datenbank geholt werden. Die Verarbeitung dieser Daten muss auf einer höheren Ebene geschehen.

Ein Vergleich der hier bewerteten Systeme mit der Eigenentwicklung HEAVEN folgt in Kapitel 5. Das DBMS RasDaMan und HSM-Systeme bilden die Grundlage von HEAVEN. Besonderheiten des multidimensionalen Array-DBMS RasDaMan werden in Kapitel 2.5 beschrieben.

2.5 Multidimensionales Array-DBMS RasDaMan

Wie im Kapitel 1 gezeigt, besteht ein sehr großer Bedarf, multidimensionale Array-Daten zu speichern und effizient anzufragen. Durch die Forschungsarbeiten im Projekt ESTEDI wurde bewiesen, dass gerade die HPC-Anwendungsgebiete sehr stark von den Vorteilen der Datenhaltung in DBMS profitieren können. Der Übergang von der Speicherung der Daten als Datei im Dateisystem, zur Speicherung in DBMS, bietet die bekannten Vorteile: Zum Beispiel flexible Zugriffsmöglichkeiten, Transaktionskonzepte, Recovery, Zugriffskontrolle, Sichten und die Vermeidung von Redundanz und Inkonsistenz. In diesem Kapitel wird das multidimensionale Array-DBMS RasDaMan (*Raster Data Management*) näher betrachtet. Dabei wird speziell auf Grundlagen eingegangen, die für die später folgende Tertiärspeicheranbindung von Bedeutung sind. Das DBMS RasDaMan wurde bei FORWISS innerhalb des europäischen ESPRI-20073 Projektes RASDAMAN in den Jahren 1995 bis 1998 entwickelt [BFRW97, BDFR98, BDFR99]. Auch nach der 1997 erfolgten Ausgründung von Active Knowledge GmbH, wurde RasDaMan stetig weiterentwickelt. RasDaMan ist das erste, kommerziell vertriebene DBMS für Rasterdaten (Array-Daten) beliebiger Dimensionalität. Im Laufe der Entwicklung von RasDaMan, entstanden bisher vier Dissertationen [Rits99, Furt99, Widm00, Dehm02], was den wissenschaftlichen Anspruch dieses Forschungsbereiches bestätigt. Das DBMS RasDaMan ist vornehmlich als Lösung für die Wissenschaft gedacht, um sehr große, multidimensionale Array-Daten zu verwalten und zu analysieren. Um eine Vorstellung über die prinzipiellen Unterschiede zwischen relationalen DBMS und Array-DBMS zu erhalten, werden diese Systeme in Tabelle 2.4 anhand ausgewählter Kriterien verglichen.

Die wichtigsten Unterschiede liegen vor allem im Datenbankvolumen, in der Größe der einzelnen Datenelemente und deren komplexen Struktur. Im direkten Zusammenhang zur unterschiedlichen Größe der einzelnen Datenelemente steht die Anzahl der erforderlichen Datenbankseiten bei der Speicherung auf Festplatte. Weiterhin ist bei Array-DBMS eine viel geringere Kardinalität der Objektmengen zu erkennen. Daraus ergeben sich spezielle Anforderungen an Array-DBMS. Es müssen kostenintensive Operationen und den damit verbundenen, überwiegend CPU-lastigen, Ressourcenverbrauch bewältigt werden. Bei relationalen DBMS ist der Ressourcenverbrauch durch *Ein-/Ausgabe*- (E/A) und CPU-Operationen gekennzeichnet. Im Gegensatz zur Anfragesprache SQL (Structured Query Language) bei relationalen DBMS ist für das effiziente Retrieval von Array-Daten eine eigenständige, multidimensionale Anfragesprache (*Multidimensional Query Language*; MQL) notwendig. Weitere Besonderheiten des multidimensionalen Array-DBMS RasDaMan werden in den folgenden Kapiteln vorgestellt. Zunächst folgt ein Überblick über die Architektur des DBMS RasDaMan.

Bewertungskriterien	Relationales DBMS	Array-DBMS
Datenbankvolumen	unterer bis mittlerer GByte-Bereich	GByte-Bereich bis PByte-Bereich
Größe einzelner Datenelemente	Tupel → sehr klein Byte bis KByte	Array-Daten → sehr groß MByte bis GByte
Komplexität der Datenelemente	einfache Struktur	komplexe Struktur
Speicherung eines Datenelementes auf Festplatte	n Objekte pro Datenbankseite	N Datenbankseiten pro Datenobjekt
Kardinalität der Objektmenge	$\leq 10^7$	≤ 200
Kosten einer Operation auf ein Datenelement	billig	teuer bis sehr teuer
Ressourcenverbrauch	E/A & CPU	überwiegend CPU
Anfragesprache	SQL	MQL

Tabelle 2.4: Vergleich: Relationales DBMS und Array-DBMS.

2.5.1 Systemarchitektur

Das multidimensionale Array-DBMS RasDaMan ist, wie in Abbildung 2.13 zu erkennen, als vierschichtige Architektur realisiert. Jede dieser Schichten kann auf unterschiedlichen Rechnern betrieben werden. Die erweiterte Systemarchitektur hinsichtlich der Kopplung mit tertiären Speichertechnologien, wird in Kapitel 3.1 erläutert.

Die oberste Schicht stellt die Client-Schicht dar. Sie erlaubt es, Anwendungen über ein Modul mit dem RasDaMan-Server und dem RasDaMan-Manager zu kommunizieren. Diese Kommunikation kann über die beiden Protokolle RPC (*Remote Procedure Call*) oder HTTP (*Hypertext Transfer Protocol*) erfolgen. Um das zu übertragende Datenvolumen zu reduzieren, ist eine Komponente zur Komprimierung, bzw. Dekomprimierung, vorhanden. Applikationen verfügen über die C++ bzw. JAVA Klassenbibliothek RasLib (*RasDaMan Library*), die Datendefinitionssprache RasDL (*RasDaMan Data Definition Language*) und die Datenmanipulationssprache RasML (*RasDaMan Data Manipulation Language*). RasML ermöglicht, Daten einzufügen, zu ändern und zu löschen und enthält die multidimensionale Anfragesprache RasQL (*RasDaMan Query Language*). Als Zielarchitektur für den RasDaMan Client kommen Unix, Linux und Windows Systeme in Frage.

Der *RasDaMan-Manager* (RasMgr) wird in der zweiten Schicht realisiert. Diese Schicht übernimmt die zentralen Verwaltungsaufgaben, Mehrbenutzerbetrieb, Zugriffskontrolle, Authentifizierung und die Zuweisung von Anfragen an unterschiedliche RasDaMan-Server. Durch die Möglichkeit der Verwendung mehrerer RasDaMan-Server, wird eine parallele Anfragebearbeitung (Inter-Query-Parallelität) ermöglicht. Eine nähere Betrachtung im Bereich der Parallelisierung folgt in Kapitel 3.7.3.

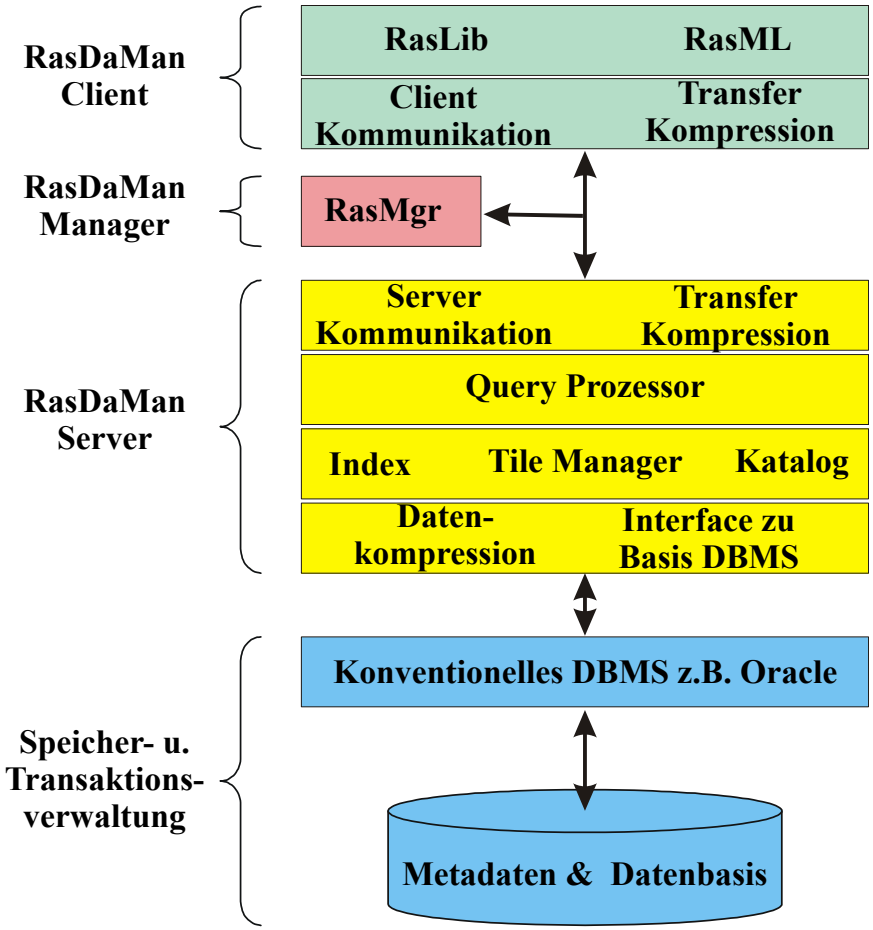


Abbildung 2.13: Systemarchitektur des DBMS RasDaMan.

Als Hauptschicht ist der RasDaMan-Server anzusehen, in der alle Anfragen abgearbeitet werden. Diese Schicht enthält zahlreiche gekapselte Module. Neben einem Modul zur Kommunikation mit dem RasMgr und dem Client kann die zu übertragende Datenmenge durch Komprimierung (Transferkompression) verdichtet werden. Die Anfragebearbeitung findet in der Komponente Query-Prozessor statt. Dabei werden zunächst mittels eines Parsers aus den Anfragen Anfragebäume generiert. Diese werden daraufhin optimiert und ausgeführt. Die Module der darunter liegenden Ebene, übernehmen die multidimensionale Indizierung, um ein schnelles Auffinden der Datenobjekte zu gewährleisten. Der Tile-Manager übernimmt die Verwaltung, der in einzelnen Kacheln (Tiles) zerlegten Datenobjekte. Für die Verwaltung der Metadaten ist das Katalog-Modul verantwortlich. Weiterhin kann durch ein Modul zur Datenkomprimierung, die Datenbasis der untersten Schicht komprimiert gespeichert werden. Die Kommunikation mit der untersten Schicht erfolgt über eine eigene DBMS-Schnittstelle und ist durch ESQL (*Embedded Structured Query Language*) verwirklicht. Die Zielarchitektur für den RasDaMan Server sind Unix- und Linux-Systeme.

Die unterste Schicht ist die Speicher- und Transaktionsverwaltung. RasDaMan verwendet hierzu ein konventionelles, relationales DBMS, das über eine Schnittstelle mit dem RasDaMan-Server verbunden ist. Gegenwärtig werden Oracle, IBM/DB2 und Informix von RasDaMan unterstützt. Durch die Verwendung fortschrittlicher und konventioneller Datenbank-

technologie, können große Datenmengen effizient verwaltet werden. Außerdem baut das Transaktionskonzept von RasDaMan, direkt auf den Konzepten konventioneller DBMS auf. Nach der Erläuterung der RasDaMan-Systemarchitektur wird auf das logische und physikalische Datenmodell eingegangen.

2.5.2 Logisches Datenmodell

In diesem Kapitel wird das von RasDaMan verwendete logische Datenmodell für Objekte beliebiger Dimensionalität näher beschreiben. Grundsätzlich setzt sich ein logisches Datenmodell aus einer Trägermenge und den darauf ausgeführten Operationen zusammen. Das hier definierte, logische Datenmodell lehnt sich an die Beschreibungen von [Baum99, Rits99] an. Das relationale Modell [Codd70] wird um den Datentyp multidimensionaler Objekte (Array-Daten) und den entsprechenden multidimensionalen Operationen erweitert. Eine Zusammenstellung der verwendeten Nomenklatur ist im Notationsverzeichnis auf Seite 235 aufgeführt.

Als Trägermenge gelten multidimensionale Array-Daten, die auch als Rasterdaten und als multidimensionale, diskrete[11] Daten (*Multidimensional Discrete Data*; MDD) bezeichnet werden. Diese MDD entstehen aus Sampling (Abtastung) und Diskretisierung von Phänomenen: Wie z.B. Temperatur oder Geschwindigkeit über einen gewissen multidimensionalen Raum. Diese Daten werden in einem diskreten Raum $\mathbb{Z}^d$ in einem Raster der Dimensionalität d gespeichert. Die Lage und Ausdehnung eines MDD α im multidimensionalen Raum, wird durch seine Domäne (Wertebereich) beschrieben. Abbildung 2.14 zeigt zur Veranschaulichung ein dreidimensionales MDD in einem diskreten Raum $\mathbb{Z}^3$.

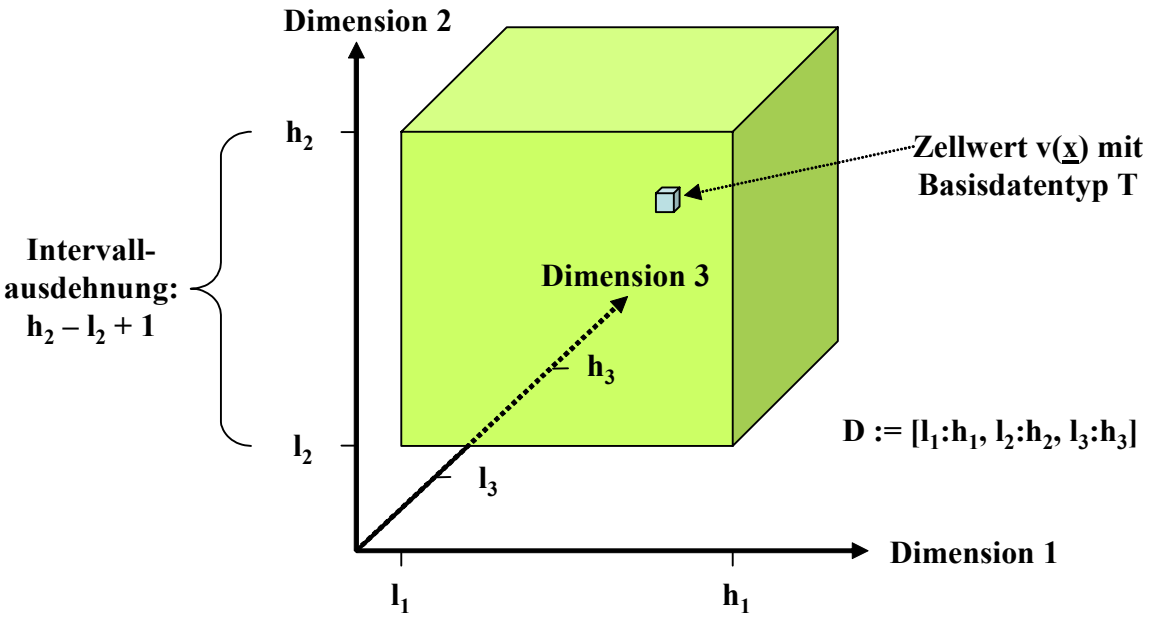


Abbildung 2.14: Lage und Ausdehnung eines MDD α im multidimensionalen Raum.

[11] Der lateinische Begriff „discretus“ bedeutet „getrennt“. Aus mathematischer Sicht entspricht eine diskrete Menge einer Menge mit einer Kardinalität $\leq \mathbb{N}_0$. Dies bedeutet, dass alle Punkte klar voneinander trennbar sind.

Die Domäne wird durch ein multidimensionales Intervall D der Dimensionalität d, mit den unteren Grenzen l_i und den oberen Grenzen h_i für jede Dimension i spezifiziert. Jedem Punkt (Vektor) $\underline{x}$ in diesem multidimensionalen Intervall D ist ein Zellwert $\nu(\underline{x})$ mit dem Basisdatentyp T zugeordnet. Das bedeutet, die Domäne ist die Menge aller Punkte $\underline{x}$ des multidimensionalen Intervalls D. Von dem genannten Basisdatentyp T ist der Typ (bzw. Datentyp) M eines MDD α abzugrenzen, da dieser durch das multidimensionale Intervall D und dem Basisdatentyp T beschrieben wird. Es folgt die formale Definition eines multidimensionalen Intervalls (bzw. Domäne) D, des Basisdatentyps T, des MDD α und des Typs M eines MDD. Eine weiterführende Behandlung des logischen Datenmodells wurde bei [Rits99] angestellt.

Definition 2.1: Multidimensionales Intervall (bzw. Domäne) D

Ein multidimensionales Intervall (bzw. Domäne) D der Dimensionalität d wird durch einen unteren $\underline{l}$ und einen oberen Begrenzungsvektor $\underline{h}$ definiert als:

$$D := \bigtimes_{i=1}^{d} \{ x \mid l_i \le x \le h_i, x \in \mathbb{Z} \} = [l_1:h_1] \times ... \times [l_d:h_d]$$

mit $\underline{l}, \underline{h} \in \mathbb{Z}^d$ und $l_i \le h_i$ für alle $i \in \{1, \ldots, d\}$

Für jede Dimension i wird das multidimensionale Intervall durch ein eindimensionales Intervall $x = [l_i:h_i]$ beschrieben. Die Ausdehnung dieses Intervalls lässt sich durch $h_i - l_i + 1$ ermitteln. Da es sich um ein abgeschlossenes Intervall handelt, wird in dieser Arbeit die Kurzschreibweise $[l_1:h_1, \ldots, l_d:h_d]$ für ein multidimensionales Intervall D verwendet. Die Funktion *low*(D) liefert die unteren Grenzwerte $[l_1, l_2, ..., l_d]$ eines d-dimensionalen Intervalls D. Die Funktion *high*(D) liefert die entsprechenden oberen Grenzwerte $[h_1, h_2, ..., h_d]$. Mit der Funktion *extent*(D) erhält man die Ausdehnung $[e_1, e_2, ..., e_d]$ eines Intervalls D, mit $e_i = h_i - l_i + 1$.

Nach der Definition eines multidimensionalen Intervalls wird der Basisdatentyp T eines MDD näher spezifiziert. Jeder einzelne Zellwert $\nu(\underline{x})$ eines MDD α hat diesen Basisdatentyp T, der aus atomaren und zusammengesetzten Datentypen bestehen kann.

Definition 2.2: Basisdatentyp T eines MDD

ζ ist eine Menge gegebener skalarer (atomarer) Datentypen, wie zum Beispiel unsigned integer $\mathbb{N}_0$, integer $\mathbb{Z}$, real number $\mathbb{R}$, boolean $\mathbb{B}$, usw. Die Menge aller Werte eines Basisdatentyps T ist somit definiert als:

$$T := \bigtimes_{i=1}^{n} \{ t_i \mid t_i \in \zeta \cup T \}; n \in \mathbb{N}; \zeta \in \mathbb{N}_0 \cup \mathbb{Z} \cup \mathbb{R} \cup \mathbb{B} ...$$

Somit können beliebig komplexe Basisdatentypen aus atomaren und zusammengesetzten Datentypen aufgebaut werden. Als Kurzschreibweise wird $\{type_1; \ldots; type_n\}$ eingeführt. Der Basisdatentyp {float; {char; char; char}} ist eine Beispielausprägung. Jeder atomare Daten-

typ, also ein einzelnes Element eines zusammengesetzten Datentyps, wird auch als Kanal bezeichnet. In der Praxis wird ζ durch die von einem DBMS unterstützten atomaren Datentypen bestimmt. Die Funktion *size*(T) liefert die Größe des Basisdatentyps in Byte.

Vor der Definition eines multidimensionalen Objektes α wird ausdrücklich darauf hingewiesen, dass die Begriffe multidimensionales Objekt, multidimensionales Array, Rasterdatenobjekt und MDD synonym verwendet werden.

Definition 2.3: MDD (multidimensionales Objekt) α

Ein MDD α mit dem multidimensionalen Intervall D und dem Basisdatentyp T ist eine Menge von Tupel (Elementen), der Form (Position $\underline{x}$, Wert $\nu(\underline{x})$), die durch die Abbildung f: D→T definiert werden:

$$\alpha := \left\{ \left(\underline{x},\ \nu\left(\underline{x}\right) \right) \mid \nu\left(\underline{x}\right) \in T,\ \underline{x} \in D \right\}$$

Vom Basisdatentyp T einer Zelle eines MDD ist der Typ eines MDD zu unterscheiden.

Definition 2.4: Typ (bzw. Datentyp) M eines MDD

Der Typ eines MDD α ist definiert aus dem multidimensionalen Intervall D und dem Basisdatentyp T als *type*(α) = <D, T>. Die Menge aller MDD mit dem multidimensionalen Intervall D und dem Basisdatentyp T sind von Typ M = <D, T>.

Eine Menge von MDD vom Typ M = <D, T> können als Kollektionen verwaltet werden. Vergleichen wir MDD und Kollektionen mit der Datenspeicherung in relationalen DBMS, so stehen MDD für einzelne Tupel einer Tabelle. Da Kollektionen gleichartige MDD zusammenfassen, kann eine Kollektion als Tabelle im relationalen Fall angesehen werden.

Definition 2.5: Kollektion C

Eine Kollektion C enthält ein oder mehrere MDD mit gleicher Domäne D und gleichem Basisdatentyp T und ist definiert als:

$$C \subset \{\alpha \mid type(\alpha) = <D, T>\}$$

Aufgrund der Kombination gleichwertiger MDD zu einer Kollektion C ist eine Kollektion vom Typ M = <D, T>.

2.5.3 Physikalisches Datenmodell

Das dem Array-DBMS RasDaMan zugrunde liegende physikalische Datenmodell erlaubt den effizienten Umgang mit großen Datenmengen. Die physikalische Ebene wird über eine defi-

nierte Schnittstelle angebunden und gewährleistet somit einen gewissen Grad an Datenunabhängigkeit. Durch eine gekachelte Abspeicherung der Array-Daten und der Definition einer Indexstruktur über diese Kacheln (Tiles), wird eine Navigation auf riesige Datenvolumina in Echtzeit erst möglich. Zunächst werden die Vorteile der Speicherung großer, multidimensionaler Array-Datenobjekte (MDD) in gekachelter Form besprochen. Anschließend erfolgen eine formale Definition der Kachelung (*Tiling*) eines MDD und eine Klassifizierung unterschiedlicher Möglichkeiten der Kachelung. Die Realisierung eines effizienten Zugriffes mittels Indexstrukturen wird in Kapitel 2.5.4 behandelt.

Normalerweise werden multidimensionale Arrays im Speicher eines Computers in linearisierter Form abgelegt. Dabei werden große Objekte auf mehrere Blöcke im Dateisystem, bzw. auf mehrere Speicherseiten in einem DBMS aufgeteilt. Die Linearisierung der Daten und die Aufteilung in Blöcke (1 bis 9) erfolgt entlang einer Vorzugsdimension, wie es die linke Seite der Abbildung 2.15 zeigt. Die Position eines bestimmten Punktes im multidimensionalen Objekt wird durch eine Linearisierungsfunktion definiert. Die Linearisierung eines kompletten MDD bei der Ablage auf Sekundärspeicher hat den Nachteil, dass die räumliche Nähe verloren geht. Die Nachbarschaft einer Koordinate wird auf dem Sekundärspeicher verstreut. Ausgenommen ist der direkte Vorgänger und Nachfolger einer Koordinate, da diese Nachbarschaft durch die Linearisierung erhalten bleibt. Bei einer Bereichsanfrage müssen die zugehörigen Daten, die auf Sekundärspeicher verstreut vorliegen, über die langsame, wahlfreie Zugriffsmethode (Random Access) ausgelesen werden. Wie die linke Seite der Abbildung 2.15 zeigt, müssen für eine Bereichsanfrage die Blöcke 1 bis 6 vom Sekundärspeicher geladen werden, obwohl nur geringe Teilbereiche dieser Blöcke benötigt werden. Der überschüssig gelesene Bereich der Blöcke wird als Verschnitt bezeichnet (vergleiche Definition 2.11).

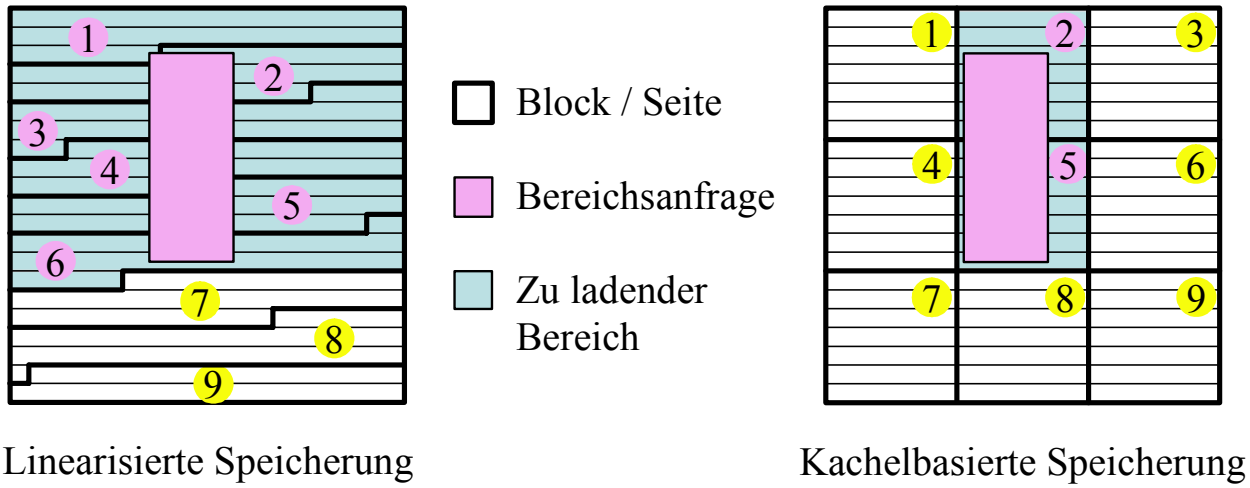


Abbildung 2.15: Vergleich unterschiedlicher Arten der Datenspeicherung.

Eine effizientere Möglichkeit bietet die Methode, multidimensionale Objekte in Teilbereiche, so genannte Kacheln, zu partitionieren und abzuspeichern (Abbildung 2.15, rechts). Die Parti-

tionierung eines MDD in disjunkte Sub-Arrays (Kacheln), wird als Kachelung bzw. Tiling bezeichnet. Jede Kachel weist ein multidimensionales Intervall auf, das festlegt, welcher Teilbereich eines MDD beschrieben wird. Bei dieser Speicherform bleibt die räumliche Nähe viel besser erhalten. Die Nachbarschaft einer Zelle wird zwar nach wie vor zerstreut, allerdings nur innerhalb einer Kachel. Bereichsanfragen können somit effizienter beantwortet werden. Bei kachelbasierter Speicherung, müssen für das Beispiel einer Bereichsanfrage (Abbildung 2.15, rechts) nur zwei Blöcke (2 und 5) geladen werden. Dem gegenüber stehen sechs geladene Blöcke bei einer linearisierten Speicherung. Es zeigt sich sehr deutlich, dass der Verschnitt bei der kachelbasierten Speicherung viel geringer ausfällt. Um den Speicherbedarf auf Sekundärspeicher minimal zu halten, werden für Kachelgrößen Vielfache (1, ..., n) von Blockgrößen, bzw. Speicherseiten einer Datenbank, gewählt. Dass Kachelung, durch die Möglichkeit der optimierten Zugriffe auf Teilbereiche multidimensionaler Objekte, zu einer Steigerung der Performance führt wurde bereits in einschlägiger Literatur [DKLP94, Lamb94, SS94a, SS94b] erkannt und bewiesen. Unterschiedliche Ausprägungen der Kachelung folgen nach einer formalen Definition einer multidimensionalen Kachel und der Kachelung eines MDD. Vorab werden die Funktionen *sdom*(α) und *dim*(α) eingeführt, die das multidimensionale Intervall (Domäne, Spatial Domain) D und die Dimensionalität d eines MDD α liefern.

Definition 2.6: Multidimensionale Kachel (Tile) τ

Eine Kachel (Tile) τ ist definiert als ein multidimensionales Teil-Array eines MDD α mit gleicher Dimensionalität d, gleichem Basisdatentyp T und besteht aus einer Menge von Tupel (Elemente), der Form (Position $\underline{x}$, Wert $\nu(\underline{x})$), die durch die Abbildung f: $D_\tau \rightarrow T$ definiert werden:

$$\tau := \left\{ \left(\underline{x},\ \nu\left(\underline{x}\right) \right) \mid \nu\left(\underline{x}\right) \in T,\ \underline{x} \in D_\tau \right\};\ \text{es gilt } dim(\alpha) = dim(\tau)$$

Eine Kachel τ stellt somit wieder ein MDD (vergleiche Definition 2.3) dar, wobei das multidimensionale Intervall $D_\tau \leq D_\alpha$ ist.

Definition 2.7: Datenvolumen (Größe) Δ_τ einer multidimensionalen Kachel τ

Das Datenvolumen Δ_τ in Byte, berechnet sich aus der multidimensionalen Ausdehnung D_τ einer Kachel τ und der Größe des dazugehörigen Basisdatentyps T:

$$\Delta_\tau = size(T) \prod_{i=1}^{d} extent(D_\tau)[i];\ \text{es gilt } D_\tau = sdom\left(\tau\right)$$

Definition 2.8: Kachelung (disjunkte Partitionierung) eines MDD α

Eine Menge von Kacheln τ_i wird als Kachelung (Tiling) von MDD α definiert, falls folgende Bedingungen erfüllt sind:

$$\bigcup_{i=1}^{n} sdom(\tau_i) = sdom(\alpha)$$

$$sdom(\tau_i) \cap sdom(\tau_j) = \varnothing \text{ für } i, j \in \{1, ..., n\}; i \neq j$$

Die disjunkte Menge aller Kacheln τ_i ergeben das komplette MDD α. Aufgrund der Kachelung eines MDD α werden bei Anfragen, wenn diese nicht exakt auf Kachelgrenzen treffen, überschüssige Bereiche der Kacheln gelesen. Dieser überschüssige Bereich wird als Verschnitt bezeichnet (Abbildung 2.15). Zunächst werden Anfragen auf multidimensionale Objekte genauer spezifiziert.

Definition 2.9: Anfrage q, Anfragebereich und Menge der Anfragen Q

Eine Anfrage q auf multidimensionale Daten, entspricht der Abbildung einer Menge von MDD α auf eine Menge von MDD α':

$$q_{(\alpha)} : \bigcup_{i=1}^{n} \alpha_i \rightarrow \bigcup_{j=1}^{m} \alpha_j'$$

wobei gilt $type(\alpha_i) = < D_i, T_i >$ für $i \in \{1, ..., n\}$; $type(\alpha_j') = < D', T' >$ für $j \in \{1, ..., m\}$

Die Domäne und der Datentype der eingehenden MDD α_i können unterschiedlich sein. Als Einschränkung von RasDaMan gilt, dass die Ergebnisobjekte α_j' sowohl die gleiche Domäne, als auch den gleichen Datentyp aufweisen müssen. Die Domäne des Anfragebereiches ist durch die Methode *sdom*(q) definiert. Die Menge Q der Anfragen bei Mehrbenutzerbetrieb ist definiert als:

$$Q := \bigcup_{i=1}^{n} q_i \text{ für } i \in \{1, ..., n\}$$

Q beinhaltet alle Anfragen, die gerade von dem vorhandenen RasDaMan-Server bearbeitet, bzw. die sich in der Warteschlange des RasMgr (RasDaMan Managers) befinden. Eine Behandlung von Multianfragen folgt in Kapitel 3.7.3.2.

Definition 2.10: Menge I_τ der von einer Anfrage q betroffenen Kacheln τ

Die für eine Anfrage q zu ladende Menge I_τ an Kacheln τ eines MDD α, werden durch die Methode *intersect*(q, α) ermittelt. Es werden die Kacheln bestimmt, die Daten im Schnittbereich der Anfrage *sdom*(q) beinhalten. Es gilt:

$$I_\tau := \bigcup_{i=1}^{n} \tau_i$$

für alle $sdom(\tau_i) \cap sdom(q) \neq \varnothing;\ i \in \{1, ..., n\};\ \tau \in \alpha$

Es gilt $sdom(\alpha) \geq sdom(I_\tau) \geq sdom(q)$. Die Anzahl der durch die Anfrage q betroffenen Kacheln ist durch $N_{I\tau} = |I_\tau|$ gegeben.

Definition 2.11: Verschnitt υ_τ bei einer Anfrage (Query) q

Der Verschnitt υ_τ bei einer Anfrage (Query) q, ist der prozentuale Anteil des überschüssig gelesenen multidimensionalen Raumes, der von der Anfrage geschnittenen Menge an Kacheln τ_i und ist definiert als:

$$\upsilon_\tau := \frac{\bigcup_{i=1}^{n} \left(sdom(\tau_i) \setminus sdom(q) \right)}{\bigcup_{i=1}^{n} sdom(\tau_i)}$$

$$\text{für alle } sdom(\tau_i) \cap sdom(q) \neq \emptyset; i \in \{1, ..., n\}$$

Durch den Verschnitt υ_τ kann eine Aussage über die Güte der Kachelung getroffen werden. Je geringer der Verschnitt υ_τ bei einer Anfrage q ausfällt, desto besser ist die gewählte Kachelung für diese Anfragemuster geeignet. In der Abbildung 2.15 (rechts) ist der bei einer Anfrage aufgetretene Verschnitt bei den Kacheln 2 und 5 zu erkennen.

Es gibt verschiedene Möglichkeiten, ein MDD disjunkt zu partitionieren. Abbildung 2.16 zeigt unterschiedliche Ausprägungen von gekachelten, zweidimensionalen Objekten. Dabei werden reguläre, ausgerichtete, irreguläre und arbiträre Strategien der Kachelung klassifiziert.

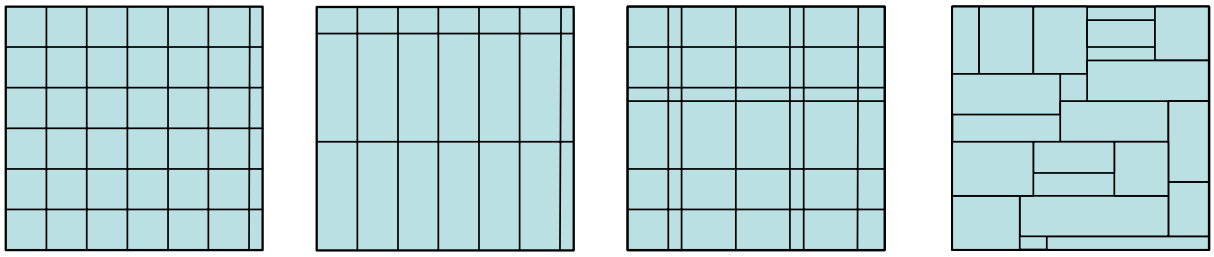


Abbildung 2.16: Unterschiedliche Ausprägungen von Kachelung eines 2D Objektes.

Welche Möglichkeit der Partitionierung verwendet werden sollte, hängt sehr stark von den zu erwartenden Anfragen ab. Ist keine Aussage hinsichtlich der zu erwartenden Anfragemuster zu treffen oder die Anfragemuster weisen keinerlei Gemeinsamkeiten auf, so ist die reguläre Kachelungsmethode vorzuziehen. Die Kacheln weisen in jeder Dimension die gleiche Ausdehnung auf. Randkacheln können von der gleichförmigen Ausdehnung abweichen. Diese einfachste Form der Kachelung (Tiling) wird auch als Chunking bezeichnet.

Treten häufig Bereichsanfragen auf, die das multidimensionale Intervall in einer Dimension einschränken, so kann durch eine ausgerichtete Kachelung der Verschnitt für eine typische Klasse von Anfragen reduziert werden (Abbildung 2.16). So wird beispielsweise in einer Zeitreihe von Satellitenbildern häufig ein bestimmter Zeitpunkt selektiert. Die Selektion eines ausgewählten Höhen- oder Breitengrades, ist dagegen nur für sehr spezielle Analysen von Bedeutung. Eine Kachelung mit geringer Ausdehnung in der Zeitdimension, reduziert somit den Verschnitt und damit E/A (Ein-/Ausgabe, bzw. *I/O*, Input/Output) für typische Anfragen und führt zu einer Verbesserung der Performanz. Bis auf die Kacheln der Randbereiche, ist das multidimensionale Intervall der Kacheln gleich. Die reguläre Kachelung kann als Spezialfall der ausgerichteten Kachelung angesehen werden.

Werden Anfragen gestellt, die das multidimensionale Intervall eines MDD in mehreren Dimensionen unterschiedlich einschränkt, so ist irreguläre Kachelung (Abbildung 2.16) zu wählen, um eine Steigerung der Performance zu erreichen. Hauptaugenmerk gilt auch hier wieder der Minimierung des Verschnitts und des E/A beim Laden der Daten. Bei dieser Art der Datenmodellierung, sind die einzelnen Kacheln nicht mehr gleichförmig aufgebaut. Wie bei regulärer und ausgerichteter Kachelung, verlaufen auch bei irregulärer Kachelung die Kachelgrenzen durchgehend entlang einer kompletten Dimension.

Durch die in Abbildung 2.16 gezeigte arbiträre Kachelung, lassen sich bevorzugte Anfragebereiche (Area of Interest) optimal modellieren. Kacheln werden bei Anfragen mit ausgedehnten Bereichen größer gewählt, als bei kleinen Bereichsanfragen. In der Praxis wird arbiträre Kachelung aufgrund sehr komplexer Einfügeprogramme und fehlender detaillierter Information über zukünftige Zugriffsmuster, meist nicht realisiert. Um eine vernünftige arbiträre Kachelung zu erhalten, ist eine automatisierte Anpassung der Kachelung aufgrund bereits gestellter Anfragen durchzuführen.

Wie die Erfahrungen aus dem Projekt ESTEDI gezeigt haben, werden in der Praxis überwiegend die reguläre und die ausgerichtete Kachelung eingesetzt. Es folgt eine formale Definition der Kachelungsstrategien:

Definition 2.12: Kachelungsstrategien eines MDD α

1. **Arbiträre Kachelung:** Generalisierte Form der Kachelung, welche den Bedingungen aus Definition 2.8 genügen.
2. **Irreguläre Kachelung:** Es gelten die Eigenschaften aus 1. und zusätzlich für alle $low(sdom(\tau_i))[k] = low(sdom(\tau_j))[k]$ gilt $extent(sdom(\tau_i))[k] = extent(sdom(\tau_j))[k]$; $i, j \in \{1, ..., n\}$; $i \neq j$; $k \in \{1, ..., d\}$. Kachelgrenzen verlaufen durch eine gesamte Dimension k (Abbildung 2.16).
3. **Ausgerichtete Kachelung:** Neben den Eigenschafen aus 2. gilt $extent(sdom(\tau_i)) = extent(sdom(\tau_j))$; $i, j \in \{1, ..., n\}$; $i \neq j$. Mögliche Ausnahme: $extent(sdom(\tau_i)) \leq extent(sdom(\tau_j))$ falls gilt $high(sdom(\tau_i))[k] = high(sdom(\alpha))[k]$ und $high(sdom(\tau_j))[k] \neq high(sdom(\alpha))[k]$; $k \in \{1, ..., d\}$. Obere Randkacheln eines

MDD α können kleiner ausfallen. Vergleiche Objektrand (oben, bzw. rechts) bei ausgerichteter Kachelung in Abbildung 2.16.

4. **Reguläre Kachelung:** Neben den Einschränkungen aus 3. gilt $extent(sdom(\tau_i))[k] = extent(sdom(\tau_i))[1]$; $i \in \{1, ..., n\}$; $k \in \{1, ..., d\}$.

Einzelne Kacheln weisen die kleinste Zugriffsgranularität auf. Die Größe der Kacheln wird entsprechend typischer Anfragen gewählt. Bei steigender Kachelgröße, beschleunigen sich im Allgemeinen die Zugriffe aufgrund geringerer Indexaktivitäten. Allerdings erhöht sich der Verschnitt, was zu erhöhtem Ladeaufwand führt. Bei Punktanfragen wird der höchste Verschnitt erreicht. In DBMS gibt es die Möglichkeit, Arrays (MDD und Kacheln) in BLOBs (Binary Large Objekts) zu speichern. Diese BLOBs stellen Byte-Container dar, um beliebig strukturierte Daten aufzunehmen. Die Daten werden linearisiert als eindimensionaler Bytestream entlang der ersten Dimension abgelegt. RasDaMan setzt zur Speicherung der BLOBs ein konventionelles RDBMS, wie zum Beispiel Oracle oder IBM/DB2 ein. Die BLOBs werden vom RDBMS in einer oder mehreren BLOB-Tablespaces organisiert. Die Tablespaces wiederum können als Raw-Device, oder als Dateien im Dateisystem gespeichert werden. Bei der vorliegenden Installation wurde das konventionelle DBMS Oracle in der Version 8.1.7 verwendet, welches auf einem Dateisystem (nicht Raw-Device) aufgesetzt ist. Weiterführende Betrachtungen über das physikalische Datenmodell von RasDaMan sind bei [Furt99] zu finden. Nach der Beschreibung des physikalischen Datenmodells, das RasDaMan zugrunde liegt, wird die Möglichkeit des effizienten Zugriffs auf die besprochenen Kacheln durch multidimensionale Indexierung behandelt.

2.5.4 Multidimensionale Indexierung

Um einen effizienten Zugriff auf Zellen zu erlauben, können Kacheln in einer räumlichen Indexstruktur verwaltet werden. In der Vergangenheit wurden verschiedene Techniken entwickelt, um multidimensionale, ausgedehnte Objekte in einem Index zu verwalten. Einen guten Überblick bietet [GG98]. Grundsätzlich sind mehrere Verfahren geeignet, einen effizienten Zugriff auf multidimensionale Objekte zu erreichen. Als Beispiele können der UB-Baum (*Universal B-Tree*) [Baye97] oder der R-Baum (*Rectangle Tree*) [Gutt84] aufgeführt werden.

Das DBMS RasDaMan realisiert den Zugriff auf multidimensionale, ausgedehnte Objekte (MDD) mittels eines R^+-Baumes [SRF87], einer Variante des R-Baumes. R-Bäume sind eine multidimensionale Generalisierung eines B-Baumes [BM72], bzw. eines B^+-Baumes[12] [Knut98]. Der R-Baum ermöglicht die Indexierung von d-dimensionalen, achsenparallelen Rechtecken. Somit lassen sich multidimensionale Kacheln gut durch einen R-Baum verwalten, da sie dieser Bedingung genügen. Die Kacheln können direkt als Datenrechtecke repräsentiert werden und müssen nicht wie anders geformte Objekte, durch kleinste umhüllende Rechtecke modelliert werden. Dabei werden, entsprechend der Größe einer Datenbankseite,

[12] Die Terminologie ist in der Literatur nicht durchgängig, da die ursprüngliche Entwicklung von Knuth [Knut73] nicht mit einem Namen versehen wurde. Hier wird die von Comer [Com79] vorgeschlagene Benennung B^+-Baum verwendet, um eine Verwechslung mit dem B*-Baum zu vermeiden.

ein oder mehrere räumlich benachbarte Datenrechtecke zu einer Datenregion zusammengefasst. Aufgrund der Kachelgröße bei RasDaMan, die ein Vielfaches von Datenbankseiten einnehmen, entspricht ein Datenrechteck genau einer Datenregion. Diese Datenregion sind die Blätter (Leaves) der Baumstruktur. Die inneren Knoten der Baumstruktur mit ihrer Verzeichnis-Region (Directory-Region; Bounding Box), fassen wiederum Regionen zusammen, die nach dem gleichen Prinzip wie Datenregionen organisiert sind.

Die Struktur eines R-Baumes gleicht einem, auf d Dimensionen verallgemeinerten, B^+-Baum und ist somit auch höhenbalanciert. Die Regionen einer Ebene des Baumes sind allerdings in der Regel nicht disjunkt. Der gesamte Datenraum wird durch die Directory-Region der Wurzel beschrieben. Die Directory-Regionen der einzelnen Baumebenen werden wiederum durch Directory-Regionen, bzw. Datenregionen auf Blattebene, detailliert. Jede Ebene ist eine vollständige und überlappende Sicht auf den Datenraum. Da ein Objekt nur einer Region zugeordnet ist, ist es aufgrund der Überlappung teilweise notwendig, mehrere Suchpfade von der Wurzel zu den Blättern zu verfolgen, bis das Objekt lokalisiert wurde. Durch eine Minimierung der Überlappung, lässt sich die Menge der Suchpfade reduzieren. Im Gegensatz zum R-Baum, sind beim R^+-Baum keine überlappenden Regionen erlaubt. Daher wird auf jeder Ebene des Baumes eine disjunkte und vollständige Repräsentation des gesamten Datenraumes erreicht. Die Überlappungen werden vermieden, da der R^+-Baum eine Partitionierung erlaubt, bei der Datenobjekte unterteilt werden dürfen. Dieses Konzept wird als Clipping bezeichnet. Ein Datenobjekt wird, entsprechend den Regionsgrenzen, unterteilt und mehreren Regionen zugeordnet. In Abbildung 2.17 wird zum Beispiel das Objekt 7 von der Directory-Region R1 und R2 referenziert. Bei geeigneten Kachelungs-Strategien (d.h. reguläre, ausgerichtete und irreguläre Kachelung), kann durch geschickte Wahl der Directory-Regionen, mehrfache Referenzierung vermieden werden. Wie Abbildung 2.17 zeigt, ist das bei arbiträr gekachelten Datenregionen meist nicht möglich.

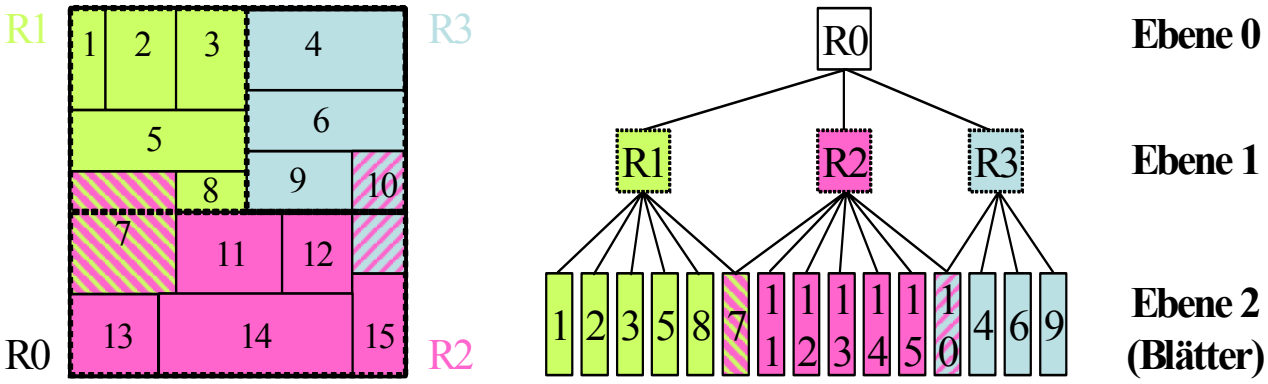


Abbildung 2.17: Ebenen eines R^+-Baumes bei der Indexierung eines 2D-Objektes.

Das Prinzip eines R^+-Baumes als räumliche Indexstruktur ist klar: Es wird die Domäne eines multidimensionalen Objektes in hierarchisch geschachtelte Sub-Domänen zerlegt. Durch

diese Zerlegung, muss bei einem Zugriff auf eine Koordinate (Punktanfrage), nicht die Domäne jeder Kachel durchsucht werden, um die Kachel zu finden, in dessen Domäne die Koordinate liegt. Stattdessen werden, ausgehend vom Wurzelknoten (Ebene 0), je Baumebene nur die Kindknoten mit der Domäne (bzw. Sub-Domäne) durchsucht, in der die Koordinate liegt. Ist man bei den Blattknoten angelangt, so hat man die Kachel mit der gewünschten Koordinate gefunden. R^+-Bäume weisen, da sie konzeptuell auf dem B-Baum basieren, eine sehr geringe Tiefe auf. Aus diesem Grund hat man eine gewünschte Kachel nach sehr wenigen Schritten gefunden. Bei einer Anfrage auf räumliche Gebiete (Bereichsanfrage), wird es meist notwendig, mehrere Äste des Baumes zu untersuchen, um alle Kacheln zu finden, die den gewünschten Bereich beinhalten. Nach der allgemeinen Beschreibung des R^+-Baumes folgt eine formale Definition:

Definition 2.13: R^+-Baum

Ein R^+-Baum hat die folgenden Eigenschaften [Gutt84, SRF87]:

1. Blattknoten haben die Form $(Id_1, D_1), ..., (Id_n, D_n)$,
   wobei Id_i (für alle $i \in \{1, ..., n\}$) ein Identifikator der einzelnen Objekte darstellt, um auf die einzelnen Objekte in der Datenbank zu verweisen. Das multidimensionale Intervall D_i beschreibt die räumliche Ausdehnung $[l_1:h_1, \ldots, l_d:h_d]$ eines d-dimensionalen Objektes.
2. Innere Knoten und Wurzelknoten haben die Form $(N_1, D_1), ..., (N_n, D_n)$,
   wobei N_i (für alle $i \in \{1, ..., n\}$) ein Zeiger auf einen Knoten ist, der eine Ebene tiefer liegt. Dabei ist die Wurzel (Ebene 0) die höchste Ebene und die Blätter entsprechen der niedrigsten Ebene. Das multidimensionale Intervall D_i beschreibt die d-dimensionale Domäne $[l_1:h_1, \ldots, l_d:h_d]$ dieses Knoten.
3. Alle Knoten besitzen maximal M Einträge. Der Wurzelknoten hat mindestens zwei Einträge, ausgenommen er ist zugleich ein Blattknoten.
4. Alle Blätter befinden sich auf der gleichen Ebene des Baums. Der Baum ist somit höhenbalanciert. Die Höhe eines R^+-Baumes beträgt $\lceil \log_m N \rceil$, wobei m ($m \leq M$) für die Anzahl der Einträge pro Knoten und N ($N > 1$) für die Anzahl der Objekte steht. Die Zeit- und E/A-Komplexität ist $O(\log_m N)$.
5. Für jeden beliebigen Eintrag (N_i, D_i) und (N_j, D_j) gilt:
   $D_i \cap D_j = \varnothing$ für alle $i, j \in \{1, ..., n\}; i \neq j$
   Diese Einschränkung legt die Überlappungsfreiheit der Regionen fest. Für alle inneren Knoten einer Ebene gilt $D_1 \cup ... \cup D_n = D$. Dies bedeutet, dass die disjunkte Menge der multidimensionalen Intervalle D_i das Gesamtintervall D ergibt.
6. Für jeden Eintrag (N, D) enthält der innere Knoten (nicht Wurzel- und Blattknoten) eines in N eingehängte Teilbaums, einen Eintrag D_i, wenn (und nur wenn) D_i in D enthalten ist ($D_i \cap D = D_i$). Eine Ausnahme bildet ein Eintrag D_j eines Wurzelknoten, da in diesem Fall D_j die Domäne D nur überlappen muss ($D_j \cap D \neq \varnothing$).

2.5.5 Operationen in RasDaMan

Operationen in RasDaMan können grundsätzlich in Operationen auf MDD und in Operationen auf Kollektionen (d.h. relationale Operationen) unterschieden werden. Zunächst werden Operationen auf MDD beschrieben.

2.5.5.1 Operation auf MDD

Auf multidimensionalen Objekten werden vier unterschiedliche Typen von Operationen definiert. Es sind geometrische Operationen, induzierte Operationen, Aggregatoperationen und Zelloperationen.

Geometrische Operationen

Geometrische Operationen sind eine Abbildung (<D, T> → <D', T>) eines MDD auf ein anderes MDD. Diese Operationen verändern weder den Typ, noch den Wert der Zellen eines MDD. Geometrische Operationen bewirken ausschließlich eine Änderung bezüglich der Domäne eines MDD:

- Trimming (Zuschneiden): Durch diese Operation wird die Domäne eines MDD auf einen Teilbereich eingeschränkt, also zugeschnitten. Die Dimensionalität des MDD bleibt erhalten. Diese Operation entspricht einer Bereichsanfrage (Range Query), häufig auch Area-of-Interest Anfrage genannt. Wird der eingeschränkte Bereich auf einen Punkt reduziert, so sprechen wir von Punktanfragen:
  trimm $_{D'}$: <D, T> → <D', T>, α → trimm(α, D')
  mit D = [l_1:h_1, ..., l_d:h_d] und D' = [l_1':h_1', ..., l_d':h_d'],
  wobei für alle i ∈ {1, ..., d} gilt: $l_i \leq l_i'$ und $h_i \geq h_i'$ und D' ⊂ D
  Die Zuschneidung eines MDD der Domäne [0:119, 0:63, 0:127] auf [50:99, 10:59, 20:69] (Abbildung 2.18, Nr. 1) reduziert zum Beispiel die Domäne der zweiten Dimension auf den Bereich 10 bis 59.
- Shift (Verschieben): Mit dieser Operation wird die Domäne eines MDD um einen gegebenen Vektor $\underline{x}$ im multidimensionalen Raum verschoben. Dimensionalität und Inhalt des MDD verändern sich nicht:
  shift $_{\underline{x}}$: <D, T> → <D', T>, α → shift(α, $\underline{x}$)
  wobei *low*(D') = *low*(D) + $\underline{x}$ und *high*(D') = *high*(D) + $\underline{x}$
- Section (Schnitt bzw. Fixieren einer Dimension): Durch diese Operation wird die Dimensionalität eines MDD um eine Dimension reduziert. Die zu reduzierende Dimension d sowie die Koordinate (Position, pos), an der der Schnitt erfolgen soll, sind als Eingabeparameter zu spezifizieren. Durch die Fixierung einer Dimension auf einen festen Wert wird aus einem MDD eine Hyperebene ausgeschnitten:
  section $_{d, pos}$: <D, T> → <D', T>, α → section(α, d, pos)
  mit D = [l_1:h_1, ..., l_d:h_d] und D' = [l_1:h_1, ..., l_{d-1}:h_{d-1}], wobei gilt: D' ⊂ D
  Der Schnitt [70, *:*, *:*] fixiert die erste Dimension auf den festen Wert 70 und reduziert somit eine dreidimensionale Domäne auf eine Ebene (Abbildung 2.18, Nr. 2). Eine Punktanfrage kann als Spezialfall von Section eingestuft werden, da durch Konkatenation, wie zum Beispiel [70, 30, 28], der Wert eines Punktes zurückgeliefert wird.

Induzierte Operationen

Induzierte Operationen sind arithmetische und logische Funktionen, die auf alle Zellwerte eines Arrays angewendet werden. Dabei können unär- und binärinduzierte Operationen unterschieden werden:

- Unäre induzierte Operation: Bei dieser Operation wird ein MDD auf ein anders MDD abgebildet, indem auf jede Zelle die definierte Operation ausgeführt wird:
  $ind_{\oplus}$: <D, T> → <D, T'>, $\alpha \rightarrow ind_{\oplus}(\alpha)$
  Beispiele für eine unäre, induzierte Operationen können die Absolutwertfunktion abs(α) und die Wurzelfunktion sqrt(α) aufgeführt werden.
- Binäre induzierte Operation: Die Operation bildet entweder zwei MDD auf ein anderes MDD oder ein MDD und einen Skalarwert (Konstante c) auf ein MDD ab:
  $ind_{\oplus}$: $<D, T_1> \times <D, T_2> \rightarrow <D, T'>$, $\alpha \times \beta \rightarrow ind_{\oplus}(\alpha, \beta)$ bzw.
  $ind_{\oplus}$: $<D, T> \times c \rightarrow <D, T'>$, $\alpha \times c \rightarrow ind_{\oplus}(\alpha, c)$
  Im ersten Fall müssen die Basistypen und die Domänen der beiden MDD übereinstimmen. Pro Koordinate der Domäne wird der entsprechende Zellwert aus dem ersten und dem zweiten Operanden geladen, die Operation angewendet und das Ergebnis in dem abgebildetem MDD gleicher Domäne abgelegt. Die Multiplikation zweier MDD $\alpha * \beta$ kann als Beispiele aufgeführt werden. Ein weiteres Beispiel ist der zellweise Vergleich (=, !=, <, ≤, > und ≥) eines MDD α mit einer Konstanten. In Abbildung 2.18 (Nummer 3) werden Bereiche gesucht, die eine höhere Temperatur als 286° Kelvin (12,85° Celsius) aufweisen. Das Ergebnis ist aus $\mathbb{B}$. Bereiche, die das Kriterium erfüllen, sind in Rot dargestellt.

Aggregatoperationen

Aggregatoperationen bilden ein MDD auf einen Skalarwert ab. Ist die Aggregatoperation für den Basistyp des MDD definiert, so werden alle Zellen des MDD zu einem Ergebniswert verdichtet. Beispiele hierfür sind Extremwertsuche, Summenbildung und Berechnung des Durchschnittswertes:

agg: <D, T> → <D', T'>, $\alpha \rightarrow agg(\alpha)$

Als Beispiel kann hier die Ermittlung der Durchschnittstemperatur von 286,519° Kelvin mittels avg_cells(α) eines entsprechenden MDD angeführt werden (Abbildung 2.18, Nr. 4). Der Existenzquantor ∃ und der Allquantor ∀ werden durch die Aggregatfunktion *some_cells* mit der Kurzform *some*, bzw. *all_cells* mit der Kurzform *all* dargestellt:

some: <D, $\mathbb{B}$> → <D', $\mathbb{B}$> mit dim(D') = 0. $\exists\, \alpha[\underline{x}] \in \alpha \mid \alpha[\underline{x}] = true \leftrightarrow some(\alpha) = true$.

all: <D, $\mathbb{B}$> → <D', $\mathbb{B}$> mit dim(D') = 0. $\forall\, \alpha[\underline{x}] \in \alpha \mid \alpha[\underline{x}] = true \leftrightarrow all(\alpha) = true$.

Im Bereich der Aggregatoperationen wird vom sonst durchgängigen logischen Datenmodell abgewichen. Das Ergebnis wird nicht als MDD, sondern als Skalarwert abgebildet. Nachteilig wirkt sich dabei aus, dass zum Beispiel bei der Extremwertbestimmung, die Information über die Lage dieser Extremwerte im multidimensionalen Raum verloren geht. Die Lokalisierung

des gefundenen Extremwertes war nur sehr uneffizient möglich, indem das komplette MDD an den Client übertragen wurde. Dieses MDD ist nur an der Position, bzw. an den Positionen des Extremwertes besetzt und enthält sonst Nullwerte. Die sonst bei Aggregatoperationen übliche Verdichtung eines MDD auf einen Ergebniswert entfällt somit komplett. Wird ein aggregierter Wert einer Extremwertbestimmung als MDD mit einer auf den Punkt des gefundenen Wertes eingeschränkten Domäne zurückgeliefert, so kann die Lage dieses Wertes im multidimensionalen Raum erhalten bleiben. Diese wurde innerhalb einer, im Rahmen dieser Dissertation betreuten Diplomarbeit [Milz03], als zusätzliche Möglichkeit implementiert. Bei einer solchen Extremwertsuche wird allerdings nur die Position des ersten auftretenden Extremwertes geliefert.

Abhilfe bieten hier Top-K Anfragen, die ebenso zu den Aggregatsoperationen zählen. Ziel dieser Anfragen ist es, nicht nur einen Extremwert zu finden, sondern die k extremsten Werte zu ermitteln. Die Suche nach der Verteilung von extrem hohen, bzw. extrem niedrigen Werten ist ein sehr häufig eingesetztes Analyseverfahren. Bisher war es in RasDaMan nur über Umwege möglich, die Top-K Werte und die Position dieser Extremwerte zu bestimmen. Es musste das gesamte Datenobjekt k-mal traversiert werden, was zu einer schlechten Performanz führte. Aus diesem Grund wurde im Projekt ESTEDI die Anforderung formuliert, eine effiziente Möglichkeit für die Durchführung von Top-K Anfragen zu schaffen. Die Entwicklung und Implementierung der Top-K Vektor-Anfragen für multidimensionale Rasterdaten erfolgte ebenfalls in der genannten Diplomarbeit [Milz03]. Die Bestimmung der Top-K Extremwerte erfordert nur noch einen kompletten Durchlauf des Datenobjektes und liefert zusätzlich die Koordinaten der Extremwerte zurück. Eine Top-K Anfrage ist durch diese Entwicklung sehr einfach und intuitiv zu stellen:

```
SELECT  TOP(50)  max_cells( a )  FROM  object AS a
```

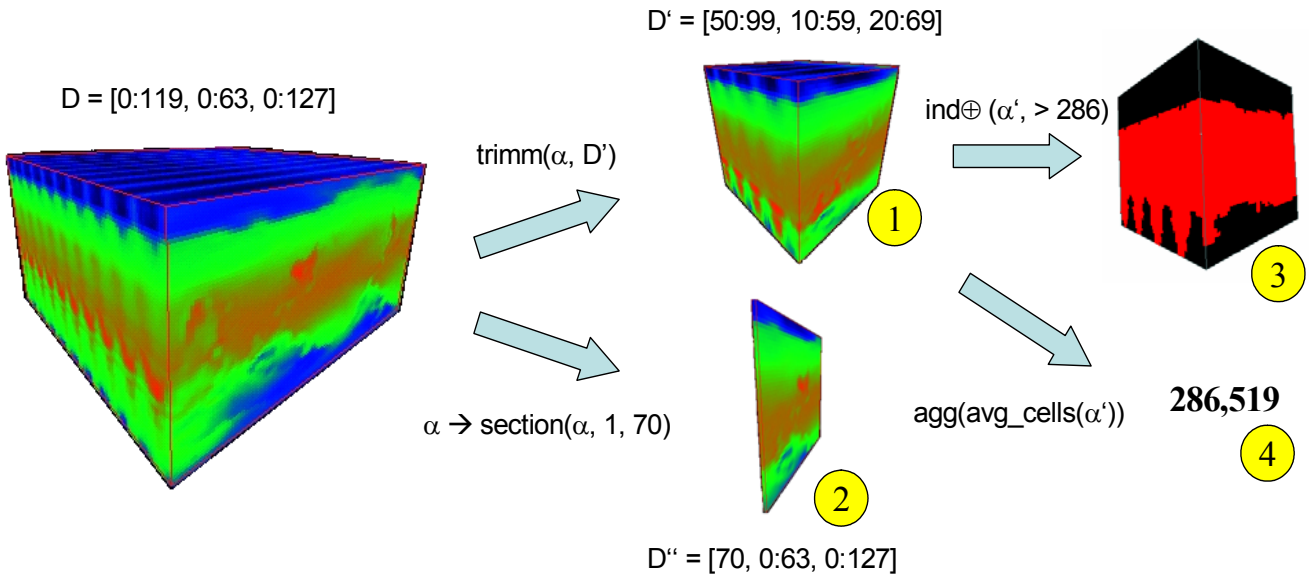


Abbildung 2.18: Beispiele für geometrische, induzierte und Aggregatoperationen.

Zelloperationen
Bei einer Zelloperation handelt es sich um eine Projektion auf einen Basistyp T. Dies bedeutet, dass der Basistyp eines MDD reduziert wird. Somit können einzelne Kanäle bei Anfragen selektiert werden:
cell: <D, T> → <D, T'>, α → cell(α, T'), mit T' $\subset$ T und |T'| $\leq$ |T|

Bei einem MDD α mit RGB-Werten können mit der Einschränkung α.red, Anfragen ausschließlich auf die Rot-Werte erfolgen. Die Anzahl der Zellen eines MDD ist durch |cell(α)| gegeben.

2.5.5.2 Operationen auf Kollektionen

Kollektionen C sind eine Menge (Set) von MDD. Es sei k die Kardinalität |C| einer Eingabekollektion von MDD, bzw. die Kardinalität $|C_1| * ... * |C_n|$ eines Tupel von MDD. Auf diese Mengen lassen sich unter anderen relationale Operationen durchführen. Im Datenmodell von RasDaMan sind die Operationen Applikation, Selektion, Kreuzprodukt und Zugriff auf einer Kollektion definiert:

- α_{op}[13] → Applikation: Ein Operatorbaum wird auf die Kollektion C angewendet. Das Ergebnis der multidimensionalen Operation op wird als Menge von MDD oder Skalarwerten zurückgegeben. Die Komplexität beträgt O(k * op).
- σ_{cond} → Selektion: Die Selektion durchläuft alle MDD einer Kollektion C und überprüft die Selektionsbedingung (Condition, bzw. cond), die wiederum eine multidimensionale Operation op darstellt. Genügt das MDD der Bedingung, so wird dieses MDD der Ergebnismenge hinzugefügt. Es ergibt sich eine Komplexität von O(k * op).
- x → Kreuzprodukt: Das kartesische Produkt (Kreuzprodukt) einer Menge n an Kollektionen C_i (für alle $i \in \{1, ..., n\}$) wird als (C_1 x … x C_n) gebildet und zurückgegeben. Das Kreuzprodukt enthält alle $|C_1| * … * |C_n|$ möglichen Kombinationen von Elementen aus C_1, …, C_n. Das Kreuzprodukt dreier Kollektionen mit zwei, fünf und acht MDD liefert 80 (2 * 5 * 8) dreier-Tupel an MDD. Bei n Kollektionen ergibt sich ein Komplexität von O($\prod k_i$).
- ω → Zugriff: Es werden alle *Objektidentifikatoren* (OID) der Elemente (MDD) einer Kollektion C geliefert. In RasDaMan wird von persistenten Objekten gesprochen, da diese noch nicht geladen wurden. Das eigentliche Laden der benötigten Elemente, bzw. Teilelemente erfolgt im Operatorbaum der Selektion, bzw. Applikation. Aus den persistenten Objekten werden dadurch transiente Objekte. Die Anfragebearbeitung erfolgt auf der Basis einer Stromverarbeitung (*Streaming*). Das bedeutet, die Objekte werden kontinuierlich, Schritt für Schritt geladen, um möglichst ohne Zeitverlust von RasDaMan verarbeitet zu werden. Die Komplexität eines Kollektionszugriffes beträgt O(k).

[13] Der Bezeichner α wird hier für eine Applikation verwendet, und ist eindeutig von der sonst verwendeten Notation für ein MDD zu unterscheiden.

Bei der Applikation und Selektion sind für die Komplexität die Kosten der Operation op ausschlaggebend. Bei induzierten und Aggregatoperationen ist die beteiligte Anzahl der Zellen |cell| bedeutend. So ergibt sich eine Komplexität für op von $O(op_s * |cell(\alpha)|)$ mit op_s als einzelne (single) Array-Operation. Die Komplexität einer geometrischen Operation beträgt O(1), da die Berechnung über Metadaten erfolgt. Die Operationen Selektion, Kreuzprodukt und Zugriff sind der relationalen Algebra entliehen. Eine allgemeine Erläuterung dieser Operationen und deren Anwendung in relationalen DBMS sind bei [KE99, HS00] zu finden.

2.5.6 Anfragesprache und -ausführung

Die Anfragesprache von RasDaMan gliedert sich in die Datenmanipulationssprache RasML und der Datendefinitionssprache RasDL. Die Datenmanipulationssprache enthält neben der Anfragesprache RasQL noch Insert-, Update- und Delete-Funktionalität.

Die deklarative Anfragesprache RasQL von RasDaMan basiert auf SQL92 (bzw. SQL 2 [AI92]) und weist somit eine SQL-ähnliche Syntax auf. RasQL wurde speziell für multidimensionale Operationen auf MDD entwickelt. Diese Anfragesprache geht auf [Baum99] zurück. Eine RasQL Anfrage hat die Form:

SELECT <Operation>
FROM <Kollektion> [as <Name>] [, <Kollektion> [as <Name>], …]
[WHERE <Selektionsbedingung>]

Neben SELECT Anfragen bietet, die RasDaMan-Manipulationssprache RasML auch INSERT, UPDATE und DELETE Konstrukte. Eine detaillierte Beschreibung der RasDaMan Anfragesprache RasQL ist bei [Rasd02] zu finden. Als Beispiel kann eine Anfrage wie folgt formuliert werden:

SELECT *max_cells*(A[0:199, 51:300, 100:249] * B)
FROM coll1 as A, coll2 as B
WHERE *avg_cells*(A) > *min_cells*(B)

Bei dieser Anfrage sind die beiden Kollektionen coll1 und coll2 beteiligt (FROM-Klausel). In unserem Beispiel soll die Kollektion coll1 aus mehreren MDD und coll2 nur aus einem MDD bestehen. Aus der Kollektion coll1 werden diejenigen MDD selektiert (WHERE-Klausel), bei denen der Durchschnittswert größer als der Minimalwert der Kollektion coll2 ist. Die Operationen *avg_cells*(A) und *min_cells*(B) sind Aggregatoperationen. Die resultierenden Objekte der Kollektion A werden, durch eine Trimming-Operation geometrisch beschnitten ([0:199, 51:300, 100:249]) und somit die Domäne (Spatial Domain, bzw. sdom) angepasst. Es folgt eine Multiplikation von coll1 (A) und coll2 (B) durch eine binäre, induzierte Operation. Die Domänen von A und B müssen identisch sein. Danach wird erneut mittels einer Aggregatoperation der Maximalwert ermittelt und als Ergebnis zurückgeliefert. Diese Anfrage wird intern von einem Parser in einen Anfragebaum umgewandelt. Der Operatorbaum dieser Beispielanfrage ist in Abbildung 2.19 zu erkennen.

Der Anfragebaum weist ein Kreuzprodukt x über die Kollektionen ω_A und ω_B auf. Zunächst bestehen die Tupel des Kreuzproduktes aus den OID der Elemente (MDD) der Kollektionen A und B. Das eigentliche Laden der Elemente erfolgt erst in den Operatorbäumen der Selektion, bzw. Applikation. (schraffierte Ovale in Abbildung 2.19). Auf jedes Tupel des Kreuzproduktes wird eine Selektion σ angewendet. Die Selektion erhält als Prädikat einen Operatorbaum, der auf jedem Tupel angewendet wird und zu „wahr“ oder „falsch“ evaluiert. Ist die Selektionsbedingung des Prädikates erfüllt, so wird der Typ an die Applikation α weitergeleitet. Hier wird ebenfalls ein Operatorbaum auf das Tupel angewendet und das Ergebnis der Anfrage berechnet. Anhand des Anfragebaumes lassen sich zwei Operatoren unterscheiden. Die linke Seite (Applikation α, Selektion σ und Kreuzprodukt x) folgt dem Iteratorkonzept, das bei relationalen DBMS weit verbreitet ist. Hier wird mittels Open-, Next- und Close-Aufrufen durch eine Menge von Objekten iteriert. Die Dreiecke der Abbildung 2.19 zeigen Operationen auf multidimensionale Array-Daten. Die Beschreibung der hier behandelten multidimensionalen Anfragesprache RasQL erhebt keinen Anspruch auf Vollständigkeit, sondern soll grundlegende Konzepte darstellen. Weiterführende Betrachtungen sind bei [Ritc99, Grae93] zu finden.

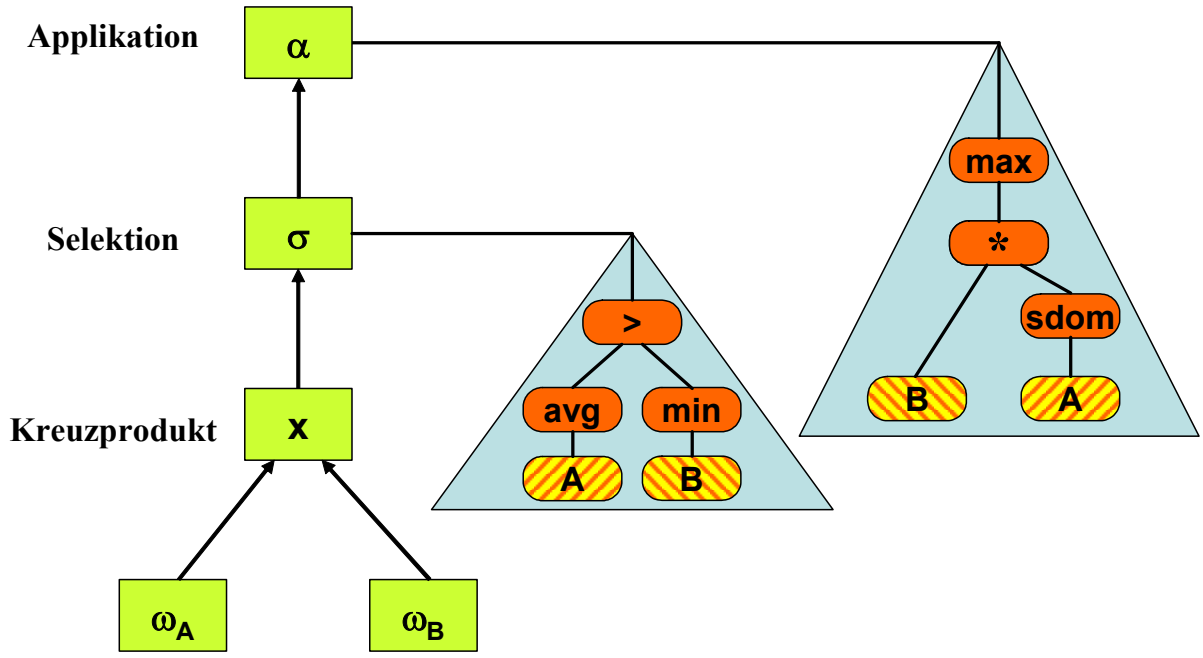


Abbildung 2.19: Beispiel eines Anfragebaums.

Eine Datendefinitionssprache RasDL ermöglicht es Anwendern, benutzerdefinierte Basisdatentypen T zu spezifizieren (vergleiche Definition 2.2). Diese stützen sich auf skalare Datentypen, wie zum Beispiel Integer, Float und Double. Sie werden in starker Anlehnung an die C++-Syntax definiert und einem neuen Typbezeichner zugewiesen. Diese Bezeichner können dadurch in der Anfragesprache genutzt werden.

If I have a thousand ideas and only one turns out to be good, I am satisfied.
Alfred Bernhard Nobel (1833 - 1896)

Kapitel 3 HEAVEN

Dieses Kapitel beschreibt das im Rahmen dieser Arbeit entwickelte System HEAVEN, eine hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array-DBMS. Dabei wird das multidimensionale Array-DBMS RasDaMan mit einem hierarchischen Speichermanagement-System verschmolzen und mit effizienter und optimierter Retrievalfunktionalität angereichert. Die Systemarchitektur von HEAVEN wird in Kapitel 3.1 dargestellt. Kapitel 3.2 beschreibt, wie die Anbindung eines hierarchischen Speichermanagement Systems an das Array-DBMS RasDaMan erfolgt. Nach einer allgemeinen Beschreibung folgt die konkrete Anbindung des hierarchischen Speichermanagement Systems Tivoli Storage Manager der Firma IBM. Kapitel 3.3 diskutiert die Vor- und Nachteile einer direkten Anbindung eines Bandlaufwerkes an RasDaMan. Geeignete Möglichkeiten der Optimierung bei Zugriffen auf TS-Medien werden in Kapitel 3.4 klassifiziert. Themen sind das im Rahmen dieser Arbeit neu entwickelte Super-Kachel-Konzept und Inter- bzw. Intra-Super-Tile-Clustering, die in Kapitel 3.5 behandelt werden. Weiterhin zeigt Kapitel 3.6 das Exportieren von Daten auf tertiäre Speichermedien und Kapitel 3.7 das Datenretrieval. Die Funktionsweise von Update- und Delete-Operationen werden in Kapitel 3.8 erklärt. Kapitel 3.9 erläutert die Realisierung der Caching-Funktionalität von Array-Daten. Mit Object-Framing wird eine Technik präsentiert, die zum einen eine Reduzierung der TS-Zugriffskosten mit sich bringt und andererseits erweiterte Möglichkeiten bezügliche der Anfragesprache von RasDaMan bietet (Kapitel 3.10). Abschließend folgen in Kapitel 3.11 die Vorteile der Nutzung eines erweiterten Systemkataloges zur Verwaltung vorberechneter Ergebnisse von Aggregatoperationen.

3.1 HEAVEN Systemarchitektur

Die Entscheidung über die Art der Anbindung von Tertiärspeichersystemen an das multidimensionale DBMS RasDaMan wurde vor allem durch die Anforderungen der am ESTEDI-Projekt beteiligten Hochleistungsrechenzentren geprägt. Wie in Kapitel 1.2 beschrieben, soll eine flexible und performante Infrastruktur für sehr große Datenvolumina entwickelt werden. Dabei ist es notwendig, vor allem den bestehenden Flaschenhals im Bereich des Retrieval und der Evaluierung dieser Rasterdaten zu beseitigen. Erreicht wird dies durch den Einsatz des

multidimensionalen DBMS RasDaMan, mit den in Kapitel 2.5 beschriebenen vielseitigen Möglichkeiten zur Datenanalyse und der automatisierten Anbindung an TS-Systeme. Bei der realisierten Lösung liegt ein besonderes Augenmerk in der Notwendigkeit, eine Vielzahl unterschiedlicher TS-Systeme, und somit auch unterschiedliche TS-Medien (z.B. DLT, LTO, AIT, DDS, MOD, DVD, usw.) zu unterstützen. Aufgrund der unterschiedlichen und nicht genormten Schnittstellen der TS-Systeme birgt diese Notwendigkeit ein gewisses Problem. Erleichterung schaffen in dieser Problematik HSM-Systeme mit ihrer Dateisystemschnittstelle. Wie in Kapitel 2.2.4 ausführlich beschrieben, erweitert ein HSM-System die lokale Festplatte um die Kapazität einer Vielzahl tertiärer Speichermedien. Aufgrund der automatisierten Migration der Daten durch die unterschiedlichen Hierarchiestufen der Sekundärspeicher und Tertiärspeicher, kann das HSM-System als „normales“ Dateisystem mit nahezu unbegrenzter Speicherkapazität angesehen werden. In Wirklichkeit unterteilt sich das HSM-System in einen begrenzten Festplattencache (meist als RAID-System realisiert), auf dem der Anwender arbeitet und einem angegliederten TS-System mit einer robotergesteuerten TS-Medienbibliothek. Solche HSM-Systeme werden in unterschiedlichen Ausprägungen bereits von Hochleistungsrechenzentren eingesetzt, was für die Verwendung dieser Systeme spricht. Darüber hinaus wurde die Verwendung und Anbindung der bestehenden HSM-Systeme an RasDaMan in einer initialen Bedarfsanalyse von den Mitgliedern des ESTEDI-Projektes als zwingend notwendig erachtet. Diese HSM-Systeme verfügen über ausgereifte und hoch entwickelte Technologien und unterstützen unterschiedliche TS-Medien. Deshalb sind HSM-Systeme hervorragend geeignet, eine Anbindung des DBMS RasDaMan an tertiäre Speichermedien zu bewerkstelligen. Um auch eine direkte Anbindung eines eigenständigen TS-Systems zu ermöglichen, wurde wie in Kapitel 3.3 beschrieben, die essentielle Funktionalität eines HSM-Systems in *HEAVEN* nachgebildet.

Bevor die in *HEAVEN* zugrunde liegende Systemarchitektur des multidimensionalen DBMS RasDaMan mit Tertiärspeicheranbindung vorgestellt wird, werden Bezeichner zur Beschreibung des aktuellen Systemzustandes eingeführt:

Definition 3.1: Objektmenge Θ_α, Θ_σ, und Θ_τ, bzw. Anzahl der Objekte $N_{\Theta\alpha}$, $N_{\Theta\sigma}$ und $N_{\Theta\tau}$

Die Objektmenge Θ_α bezeichnet die Menge aller MDD $\alpha \in \Theta_\alpha$, die von *HEAVEN* verwaltet und durch Angabe eines Objektidentifikators (OID) angefragt werden können. Die Anzahl der MDD ist durch $N_{\Theta\alpha} = |\Theta_\alpha|$ gegeben.

Die Objektmenge Θ_σ bezeichnet die Menge aller Super-Kacheln $\sigma \in \Theta_\sigma$, die von *HEAVEN* auf TS-Systemen verwaltet werden. Super-Kacheln sind die Zugriffsgranularität auf TS-Medien. Die Anzahl ist durch $N_{\Theta\sigma} = |\Theta_\sigma|$ gegeben, wobei gilt $N_{\Theta\sigma} \geq N_{\Theta\alpha}$.

Die Objektmenge Θ_τ bezeichnet die Menge aller Kacheln $\tau \in \Theta_\tau$, die von *HEAVEN* verwaltet werden. Kacheln sind die kleinste, verwaltbare Objektgranularität. Die Anzahl ist durch $N_{\Theta\tau} = |\Theta_\tau|$ gegeben, wobei gilt $N_{\Theta\tau} \geq N_{\Theta\sigma} \geq N_{\Theta\alpha}$.

Definition 3.2: Speicherinhalte Θ_P, Θ_S, Θ_{SC}, Θ_{Ton}, Θ_{Tnear} und Θ_{Toff}

Entsprechend der Zugehörigkeit einzelner Kacheln τ_i zu unterschiedlichen Speicherebenen, werden diese Speicherinhalte wie folgt bezeichnet ($i \in \{1, ..., N_{\Theta\tau}\}$; $j \in \{1, ..., N_{\Theta\sigma}\}$):

- Θ_P Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt im *Primärspeicher* (P) befinden. Die auf den Swap-Speicher[14] ausgelagerten Objekte werden dem Speicherinhalt Θ_P zugerechnet.
- Θ_{SR} Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt im *Sekundärspeicher des konventionellen DBMS von RasDaMan* (SR) befinden. Hier befinden sich nur Objekte, die nicht auf tertiäre Speichermedien exportiert werden.
- Θ_{SRC} Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt im *Sekundärspeicher des TS-Cache-Bereiches des konventionellen DBMS von RasDaMan* (SRC) befinden (Kapitel 3.9). Das sind auf TS-System exportierte Daten, die bei einer Anfrage auf Sekundärspeicher vorübergehend zwischengespeichert werden.
- Θ_{SHC} Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt im *Sekundärspeicher-Cache des HSM-Systems* (SHC) befinden. Dieser HSM-Cache wird als Online-Bereich des HSM-Systems bezeichnet (Abbildung 3.2). Die Kacheln sind entsprechend der Definition 3.6 als Super-Kacheln (Speicherobjekt) organisiert und zugreifbar, wobei gilt $\tau_i \in \sigma_j$. Daraus folgt $sdom(\tau_i) \subset sdom(\sigma_j)$ mit $\sigma_j \in \Theta_\sigma$.
- Θ_{Ton} Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt auf einem *Online-Medium*[15] *des HSM-Systems* (T_{on}) befinden. Ein Medium wird als Online-Medium bezeichnet, wenn es sich in einer Schreib-/Lesestation (Laufwerk) befindet. Die Kacheln sind entsprechend der Definition 3.6 als Super-Kacheln (Speicherobjekt) organisiert und zugreifbar, wobei gilt $\tau_i \in \sigma_j$. Daraus folgt $sdom(\tau_i) \subset sdom(\sigma_j)$ mit $\sigma_j \in \Theta_\sigma$.
- Θ_{Tnear} Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt im *Nearline-Speicherbereich des HSM-Systems* (T_{near}) befinden. Diese Kacheln sind entsprechend der Definition 3.6 als Super-Kacheln (Speicherobjekt) organisiert und zugreifbar, wobei gilt $\tau_i \in \sigma_j$. Daraus folgt $sdom(\tau_i) \subset sdom(\sigma_j)$ mit $\sigma_j \in \Theta_\sigma$.
- Θ_{Toff} Menge aller Kacheln $\tau_i \in \Theta_\tau$, die sich zum aktuellen Zeitpunkt im *Offline-Speicherbereich des HSM-Systems* (T_{off}) befinden. Diese Kacheln sind entsprechend der Definition 3.6 als Super-Kacheln (Speicherobjekt) organisiert und zugreifbar, wobei gilt $\tau_i \in \sigma_j$. Daraus folgt $sdom(\tau_i) \subset sdom(\sigma_j)$ mit $\sigma_j \in \Theta_\sigma$.

Die jeweiligen Indices (P, SR, SRC, SHC, T_{on}, T_{near}, T_{off}) bezeichnen dabei die entsprechenden Speicherorte. Allgemein gilt, dass jedes $\tau_i \in \Theta_\tau$ in mindestens einem dieser Speicherbereiche enthalten ist ($\tau_i \in \{\Theta_P \cup \Theta_{SR} \cup \Theta_{SRC} \cup \Theta_{SHC} \cup \Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}\}$). Die Menge der Objekte τ_i, die sich im Primärspeicher Θ_P befinden, werden auch als transiente Objekte

[14] Hintergrundspeicher auf der Festplatte, um Teile des Hauptspeichers bei Bedarf auslagern zu können.
[15] Bei HSM-Systemen wird sowohl ein Medium, das sich in einer Bearbeitungsstation befindet, als auch der Sekundärspeicher-Cache des HSM-Systems zum Online-Bereich gerechnet. Aufgrund des stark unterschiedlichen Zugriffsverhaltens wird in dieser Arbeit der Online-Bereich in Θ_{SHC} und Θ_{Ton} unterschieden.

bezeichnet. Die Objekte aller anderen Speicherebenen sind persistente[16] Objekte, da diese bei einer Anfrage von RasDaMan noch nicht im Hauptspeicher materialisiert wurden. Abbildung 3.1 unterscheidet die funktionale, physikalische und logische Darstellung der Speicherinhalte. Auf funktionaler Ebene kann das DBMS RasDaMan ($\Theta_{RasDaMan} = \Theta_P \cup \Theta_{SR} \cup \Theta_{SRC}$) und das HSM-System ($\Theta_{HSM\text{-}System} = \Theta_{SHC} \cup \Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}$) unterschieden werden. Im Bereich der physikalischen Datenhaltung können sich Kacheln im Primärspeicher (P), auf dem Sekundärspeicher von RasDaMan (SR oder SRC) und/oder des HSM-Systems (SHC), bzw. auf tertiären Speichermedien (T_{on}, T_{near} oder T_{off}) befinden.

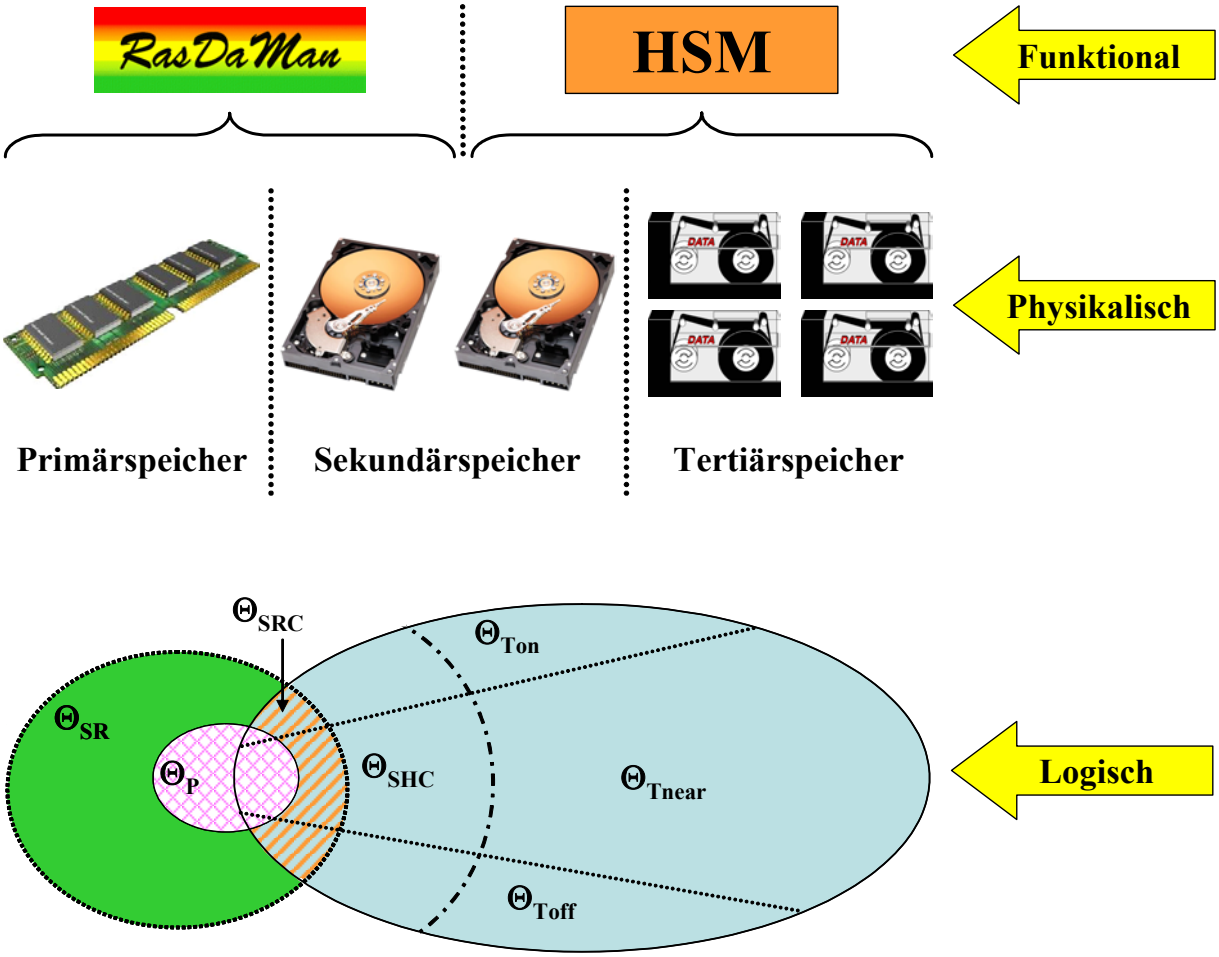


Abbildung 3.1: Funktionale, physikalische und logische Darstellung der Speicherinhalte.

Beim Übergang auf die logische Ebene, zeigen sich entsprechend der Definition 3.1 die Zusammenhänge der einzelnen Speicherinhalte und die möglichen Speicherorte der Kacheln. Es ist zu erkennen, dass einzelne Kacheln gleichzeitig in mehreren Speicherebenen enthalten sein können. So tritt zum Beispiel nicht selten auf, dass eine Kachel bei einer Anfrage von dem Speicherbereich Θ_{Ton} eines Magnetbandes in den HSM-Cache Θ_{SHC}, weiter in den RasDa-

[16] Persistent bedeutet dauerhaft, auf nicht flüchtigen Speicher geschrieben.

Man-Cache-Bereich Θ_{SRC} und schließlich zur Bearbeitung in den Primärspeicher Θ_P geladen wird. Die Kachel ist in jedem der genannten Speichereben zu finden. Zwischen den Speicherinhalten bestehen folgende Beziehungen:

- $\Theta_P \subset \{\Theta_{SR} \cup \Theta_{SRC} \cup \Theta_{SHC} \cup \Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}\}$
- $\Theta_{SR} \cap \Theta_{SRC} = \varnothing$
- $\Theta_{SR} \cap \{\Theta_{SHC} \cup \Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}\} = \varnothing$
- $\Theta_{SRC} \cap \{\Theta_{SHC} \cup \Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}\} \neq \varnothing$, falls gilt $\Theta_{SRC} \neq \varnothing$
- $\Theta_{SHC} \subset \{\Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}\}$, falls die Menge aller $\tau \in \Theta_{SHC}$ bereits von HSM-Cache auf Magnetband migriert wurde ($\tau \in \{\Theta_{Ton} \cup \Theta_{Tnear} \cup \Theta_{Toff}\}$).
- $\Theta_{Ton} \cap \{\Theta_{Tnear} \cup \Theta_{Toff}\} = \varnothing$
- $\Theta_{Tnear} \cap \Theta_{Toff} = \varnothing$

Nach der Definition der Speicherinhalte der einzelnen Hierarchieebenen, zeigt Abbildung 3.2 die in *HEAVEN* zugrunde liegende Systemarchitektur des multidimensionalen DBMS RasDaMan mit Tertiärspeicheranbindung.

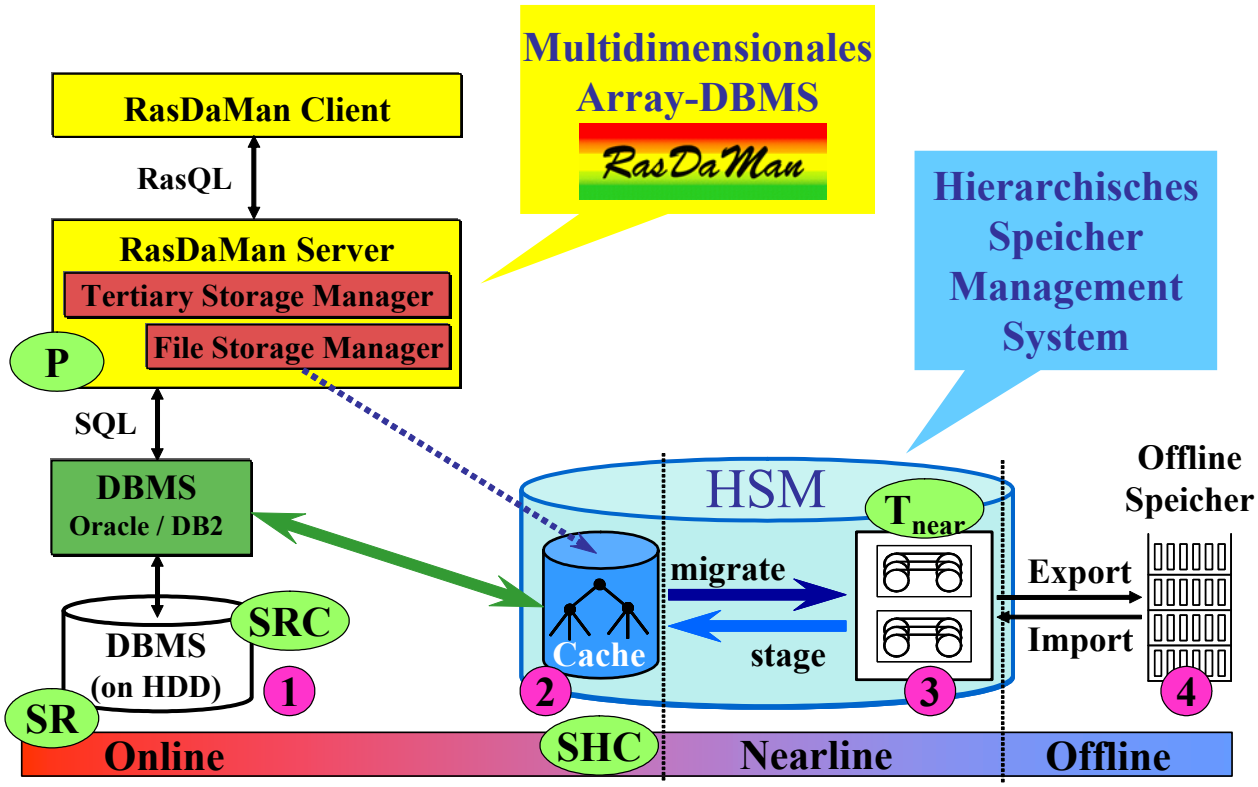


Abbildung 3.2: Systemarchitektur von RasDaMan mit Tertiärspeicheranbindung.

Die linke Seite veranschaulicht in vereinfachter Form, die Architektur des DBMS RasDaMan, mit einem RasDaMan-Client, einem RasDaMan-Server und einem darunter liegenden konventionellen DBMS (Kapitel 2.5.1, Abbildung 2.13). Das konventionelle DBMS wird von

RasDaMan zur Speicher- und Transaktionsverwaltung eingesetzt. In der originalen Version des DBMS RasDaMan, werden MDD in gekachelter Form als BLOBs in einer relationalen Tabelle dieses konventionellen DBMS auf Festplatte gespeichert (Abbildung 3.2, Nr. 1). Die dort befindlichen Daten liegen entweder im Speicherbereich Θ_{SR} für dauerhaft auf HDD gespeicherte Daten oder für zwischengespeicherte Daten in Θ_{SRC}. Die rechte Seite der Abbildung beschreibt die Komponenten eines HSM-Systems, mit einem Festplatten-Cache und einem automatisierten TS-System. Dabei können die drei Bereiche Online, Nearline und Offline unterschieden werden. Daten, die sich im HSM-Cache[17] befinden, werden als Online bezeichnet (Speicherbereich Θ_{SHC}), da auf sie sehr schnell zugegriffen werden kann (Abbildung 3.2, Nr. 2). Befinden sich Daten zum Beispiel auf Magnetbänder in der robotergesteuerten TS-Medienbibliothek[18], so spricht man von Nearline-Daten (Θ_{Tnear}). Auf diese Daten kann ohne zusätzliche Nutzerinteraktion zugegriffen werden (Abbildung 3.2, Nr. 3). Ausnahme hierzu bilden Medien, die sich aktuell in einer Schreib-/Lesestation befinden. Diese Medien werden dem Speicherbereich Θ_{Ton} zugerechnet. Ein Backup dieser Daten wird aus Gründen der nicht vorhandenen Kapazität, teilweise im Offline-Speicher gehalten (Abbildung 3.2, Nr. 4). Diese Daten sind nur manuell durch einen Administrator in das automatisierte System einzufügen und werden dem Speicherinhalt Θ_{Toff} zugeordnet.

Als zusätzliche Komponenten wurden in den Kern des RasDaMan-Servers der Tertiary-Storage-Manager und der File-Storage-Manager integriert (Abbildung 3.2, links). Diese beiden Komponenten übernehmen die gesamte Verwaltung und Funktionalität der Anbindung eines beliebigen HSM-System an RasDaMan. Bei der Ausführung einer Anfrage an RasDaMan, werden zunächst die betroffenen Kacheln ermittelt. Zur Durchführung dieser Aufgabe, verfügt der Tertiary-Storage-Manager über die notwendige Information (Metadaten), an welchem Speicherort (1, 2, 3 oder 4) sich die jeweiligen Kacheln befinden. Die multidimensionalen Objekte können permanent auf der Festplatte des konventionellen DBMS ($\tau \in \Theta_{SR}$) bzw. auf dem TS-Medium ($\tau \in \{\Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$) gespeichert sein. Eine weitere Möglichkeit bietet der realisierte TS-Cache innerhalb der Datenbank ($\tau \in \Theta_{SRC}$). Dort werden aus Gründen der Performance, die auf dem TS-Medium befindlichem Daten entsprechend gewisser Kriterien zeitweise zwischengespeichert. Die auf dem Tertiärspeicher befindlichen Daten, werden automatisch von einem oder mehreren TS-Medien über den HSM-Cache ($\tau \in \Theta_{SHC}$) in den TS-Cache ($\tau \in \Theta_{SRC}$) von RasDaMan geladen. Angeleitet vom Tertiary-Storage-Manager, übernimmt der File-Storage-Manager dabei den Import der Daten über die Dateischnittstelle des HSM-Systems. Nach dem Importieren der angefragten Daten, erfolgt die Anfragebearbeitung vom RasDaMan-Server, durch das Laden der Daten in den Hauptspeicher ($\tau \in \Theta_P$). Durch eine automatisierte Anfragebearbeitung über mehrere Speicherebenen, wird eine aktive Speicherhierarchie geschaffen.

HEAVEN bietet dem Anwender dabei eine logische Sicht auf die gespeicherten Daten, unabhängig von dem tatsächlichen physikalischen Speicherort. Durch das angegliederte HSM-

[17] Meist als RAID-System realisiert. Die Speicherkapazität liegt bei mehreren hundert GByte bis ca. 50 TByte.
[18] Die Speicherkapazität reicht von mehreren TByte (10^{12}) bis hin zu einigen PByte (10^{15}).

System, wird dem Anwender eine Transparenz in Bezug auf den Speicherort, der eingesetzten Speichergeräte und Speichermedien gewährleistet. Durch die im RasDaMan-Server angesiedelten Komponenten Tertiary-Storage-Manager und File-Storage-Manager ist es unerheblich, welches der von RasDaMan unterstützten, konventionellen DBMS Verwendung findet. Auch bei einer Ergänzung von RasDaMan um ein weiteres, konventionelles DBMS, wird dies nicht durch die gewählte Lösung zur TS-Anbindung beeinträchtigt. Mit der hier vorgestellten Systemarchitektur, können an RasDaMan nahezu alle HSM-Systeme problemlos angeschlossen werden. Diese Anbindung kann entweder mittels *Network File System* (NFS) oder *File Transfer Protocol* (FTP) erfolgen. Das NFS-Protokoll wurde von Sun Microsystems entwickelt. Es erlaubt Rechnern, nahtlos auf Dateien über das Netzwerk zuzugreifen, als wären sie lokal gespeichert. NFS ist weit verbreitet und kann somit als de facto Standard angesehen werden. Das FTP bietet keinen nahtlosen Zugriff wie NFS, ermöglicht allerdings Daten via TCP/IP (*Transmission Control Protocol over Internet Protocol*) zwischen Rechnern (auch netzwerkfremder Rechner) zu übertragen. Dadurch können auch HSM-Systeme eines entfernten Standortes, auch weltweit, bedient werden. Allerdings ist eine angemessene Internetverbindung notwendig, um akzeptable Übertragungsraten zu erhalten. Weiterhin erlaubt HEAVEN die gleichzeitige Anbindung unterschiedlicher HSM-Systeme, um eine gezielte und flexible Verteilung der Datenobjekte zu ermöglichen. Somit ist HEAVEN hochgradig erweiterbar. Vergleichen wir die in Abbildung 3.2 propagierte Art der Anbindung von DBMS an tertiäre Speichermedien, so entspricht das einer Kopplung über ein Dateisystem (Abbildung 2.9, Variante 1).

Alternative Realisierungsmöglichkeit
Neben der hier gezeigten Architektur, besteht die Möglichkeit, die Datenbasis des von RasDaMan verwendeten konventionellen DBMS, direkt in den Cache-Bereich eines HSM-Systems zu legen. Verglichen mit Abbildung 3.2, verschmelzen die Speicherorte 1 und 2. Diese Vorgehensweise wird, wie in Kapitel 2.4 beschrieben, zum Beispiel von Systemen wie StorHouse/RM der Firma FileTek angeboten. Solche Systeme wurden für die Verwaltung großer relationaler Datenvolumen entwickelt und optimiert. Das Einsatzgebiet umfasst im Allgemeinen das Archivieren von historischen Daten auf TS-Medien. Zum Beispiel eines Data Warehouses. Die operativen Daten werden nach wie vor auf Sekundärspeicher gehalten, um einen schnellen Zugriff zu gewährleisten. Besteht jedoch bei gewissen Anfragen die Notwendigkeit, auf TS-Medien ausgelagerte, historische Daten zuzugreifen, so erfolgt die Migration der Daten automatisiert. Allerdings sind diese Zugriffe nicht Bestandteil der Anfragemuster des „normalen“ Tagesgeschäftes.

Die von RasDaMan verwendeten BLOBs, zur Speicherung der multidimensionalen Kacheln, werden meist in den bestehenden Systemen unterstützt. Allerdings nur auf rudimentäre Art und Weise. Durch eine direkte Einbettung der Datenbasis, des von RasDaMan verwendeten konventionellen DBMS in den Cache-Bereich des HSM-System (SHC), gehen wichtige Möglichkeiten zur Verbesserung der Performance verloren. Da bei HEAVEN aufgrund der Datenvolumina überwiegend auch operative Daten des täglichen Bedarfes ausgelagert werden, ist ein besonderes Augenmerk auf die Optimierung der Zugriffe auf TS-Medien zu legen. Zu erwäh-

nen sind hier die Minimierung kostenintensiver Medienwechseloperationen und Spuloperationen durch geeignetes Clustering der Datenbasis und Optimierung der Anfragebearbeitung. Weiterhin kann durch Caching und eine zweckmäßige Wahl der Datengranularität, der auf TS-Medien gespeicherten multidimensionalen Daten, eine Verbesserung der Retrievalperformance erreicht werden. Ausführliche Betrachtungen dieser und weiterer Möglichkeiten werden in den folgenden Kapiteln behandelt.

Einen weiteren Nachteil birgt der Aufwand für die notwendige Einbettung der von RasDaMan unterstützten, konventionellen DBMS Oracle, IBM/DB2 und Informix in bestehende, unterschiedliche HSM-Systeme. Es sind bei jedem System Besonderheiten zu beachten, was einen hohen zeitlichen Aufwand bei jeder Integration bedeutet. Wird in einem Hochleistungsrechenzentrum bereits seit einigen Jahren ein bestimmtes HSM-System eingesetzt, das keine bestehende Lösung für die Kopplung mit konventionellen DBMS anbietet, so wird es erforderlich, das DBMS eigenständig an das HSM-System anzubinden. Um eine Vorstellung über bestehende Probleme bei der direkten Einbettung zu erhalten, wird dies am Beispiel von Oracle besprochen. Als Voraussetzung gilt, dass die auf TS-Medien auszulagernde Datenbasis als Dateien (nicht als Raw-Device) organisiert vorliegt. Generell erwartet Oracle, dass sich alle benötigten System-Tablespaces und Daten-Tablespaces als Dateien auf einer lokalen Festplatte befinden und somit schnell verfügbar und zugreifbar sind. Befinden sich Datendateien unter Kontrolle eines HSM-Systems, so wird diese Forderung nur noch bedingt erfüllt. Oracle überprüft zum Zeitpunkt des Startens der Datenbank, und zu bestimmten zeitlichen Kontrollpunkten, alle Datei-Header der Oracle-Dateien, um die Datenintegrität und ein mögliches Recovery zu verifizieren. Werden Datendateien auf TS-Medien durch das HSM-System ausgelagert, so bleibt im lokalen Dateisystem, anstelle der ausgelagerten Datei, nur ein Verweis (Stub-File oder Anchor) im ersten Datenblock der ursprünglichen Datei (Kapitel 2.2.4). Reicht die in einem solchen Verweis enthaltene Information nicht aus, um der periodischen, zeitlichen Kontrolle von Oracle zu genügen, werden diese Daten zeitaufwändig von TS-Medien geladen. Um dies zu vermeiden, lassen sich prinzipiell vier Strategien zur Auslagerung auf ein HSM-System unterscheiden [WT99]:

1. Read-Only Tablespaces mit Delayed-Open Option:
   Durch die Delayed-Open Option wird verhindert, dass Oracle beim Start und zu bestimmten zeitlichen Kontrollpunkten, die Datei-Header dieser Tablespaces überprüft. Nachteilig wirkt sich aus, dass fehlende und fehlerhafte Daten, erst zum Zeitpunkt einer Anfrage erkannt werden. Ein gravierender Nachteil ist, dass nur lesend auf diese Daten zugegriffen werden kann. Probleme bereitet auch, dass Oracle aus Gründen der Performance, einmal geöffnete Dateien nicht wieder selbsttätig schließt. Dies bedeutet, dass die Dateien auf Sekundärspeicher erhalten bleiben, solange die Oracle-Instanz aktiv ist. Nur wenige HSM-Systeme unterstützen es, eine solche Dateieinbindung zu lösen, um die Dateien nach dem Zugriff wieder zu verdrängen.
2. Online/Offline Tablespaces:
   In Oracle besteht die Möglichkeit, Tablespaces Online, bzw. Offline zu schalten. Normalerweise befinden sich Tablespaces im Zustand Online. Soll vermieden werden,

dass Oracle beim Start und zu bestimmten zeitlichen Kontrollpunkten die Datei-Header einer Tablesapce-Datei überprüft, kann in den Offline Modus gewechselt werden. Um Anfragen auf diese Daten zu ermöglichen, muss zunächst mit dem Kommando ALTER TABLESPACE ONLINE die Tablespace Online geschaltet werden. Nach dem Zugriff ist diese Tablespace wieder Offline zu setzen. Da Oracle nicht überprüft, ob ein Anwender gerade einen Zugriff auf Daten dieser Tablespace ausführt, muss dies eigenständig getestet werden, um Inkonsistenzen zu vermeiden. Das Online-, bzw. Offline-Schalten der Tablespaces bringt eine gewisse zeitliche Verzögerung mit sich. Ein Vorteil dieser Möglichkeit ist jedoch, dass Daten dieser Tablespace gelesen, geschrieben und geändert werden können.

3. Partitionen von Tabellen:
   Ab Oracle 8i ist es möglich, Zeilen einer Tabelle in mehrere Tablespaces zu speichern. Durch diese horizontale Partitionierung ist es möglich, die Granularität der auf TS-Medien auszulagernden Tablespace-Dateien zu reduzieren. Eine Partitionierung, entsprechend der Werte der Schlüsselattribute, wird durchgeführt. Die Tablespaces des partitionierten Datenraumes können entsprechend den ersten beiden Punkten behandelt werden.
4. Container für Binärdaten (LOB und BFILE):
   LOBs (Large Objects) mit einer Größe unter 4 KByte werden direkt in einer Tabelle und somit in dieser Daten-Tablespace abgelegt. Übersteigt die Größe eines LOBs 4 KByte, so wird in der Tabelle ein Verweis eingetragen und das LOB in einer eigenen LOB-Tablespace gespeichert. Aufgrund der Verwaltung durch Tablespaces gelten ebenso die Punkte 1 bis 3. Neben der Möglichkeit, LOBs in einer Tablespace zu verwalten, können LOBs als *BFILE*s (Binary Files) direkt im Dateisystem als eigene LOB-Dateien gespeichert werden. Diese BFILEs können durch Zeiger innerhalb einer Tabelle zugegriffen werden. Allerdings sind BFILEs nur Read-Only Objekte und werden bei der Transaktionsverwaltung nicht berücksichtigt.

Besteht die Notwendigkeit, auch schreibend auf TS-Medien ausgelagerte Daten zuzugreifen, so bietet sich grundsätzlich nur die zweite Strategie an. Wobei diese mit den Möglichkeiten zur Partitionierung, bzw. zur Speicherung von LOBs in Tablespaces kombiniert werden können. Allerdings sind diese Möglichkeiten nicht ausreichend, um einen effizienten Zugriff auf große, multidimensionale Daten zu erreichen. Diese Arte der Lösung ist speziell auf eine bestimmte Kombination aus DBMS und HSM-System ausgelegt. Möchte man neben Oracle auch weitere, von RasDaMan unterstützte DBMS, wie IBM/DB2 und Informix, an HSM-Systeme anbinden, so sind systemabhängige Besonderheiten zu beachten und eigene Lösungen zu realisieren.

Die Entscheidung zwischen den beiden hier beschriebenen Varianten, fiel eindeutig auf die erste Variante (Abbildung 3.2). Hauptgründe hierfür sind die flexiblere Einflussnahme zur Steigerung der Performance, bessere Kontrolle über die Datenorganisation auf TS-Medien und die Unabhängigkeit gegenüber der an RasDaMan angebundenen, konventionellen DBMS. Weiterhin können nahezu alle HSM-Systeme über die bestehende Dateisystem-

schnittstelle problemlos mit HEAVEN gekoppelt werden. Durch diese Wahl wurde vermieden, spezielle Lösungen für bestimmte Kombinationen aus DBMS und HSM-Systeme entwickeln zu müssen. Somit wurde mit der gewählten Architektur von HEAVEN, ein sehr flexibles und hochgradig erweiterbares System geschaffen. Nach der Vorstellung der gewählten Systemarchitektur, folgt die Beschreibung einer Anbindung eines konkreten HSM-Systems an RasDaMan.

3.2 Anbindung eines HSM-Systems

Als Beispielsystem wird hier der *Tivoli Storage Manager* (TSM) der Firma IBM vorgestellt. Dieses System wird vom *Leibniz-Rechenzentrum* (LRZ), dem Rechenzentrum für die Münchner Hochschulen, vor allem für die Zwecke Backup und Archivierung eingesetzt. Abbildung 3.3 zeigt die am LRZ verwendete Konfiguration unterschiedlicher Bandlaufsysteme (Stand Dezember 2002).

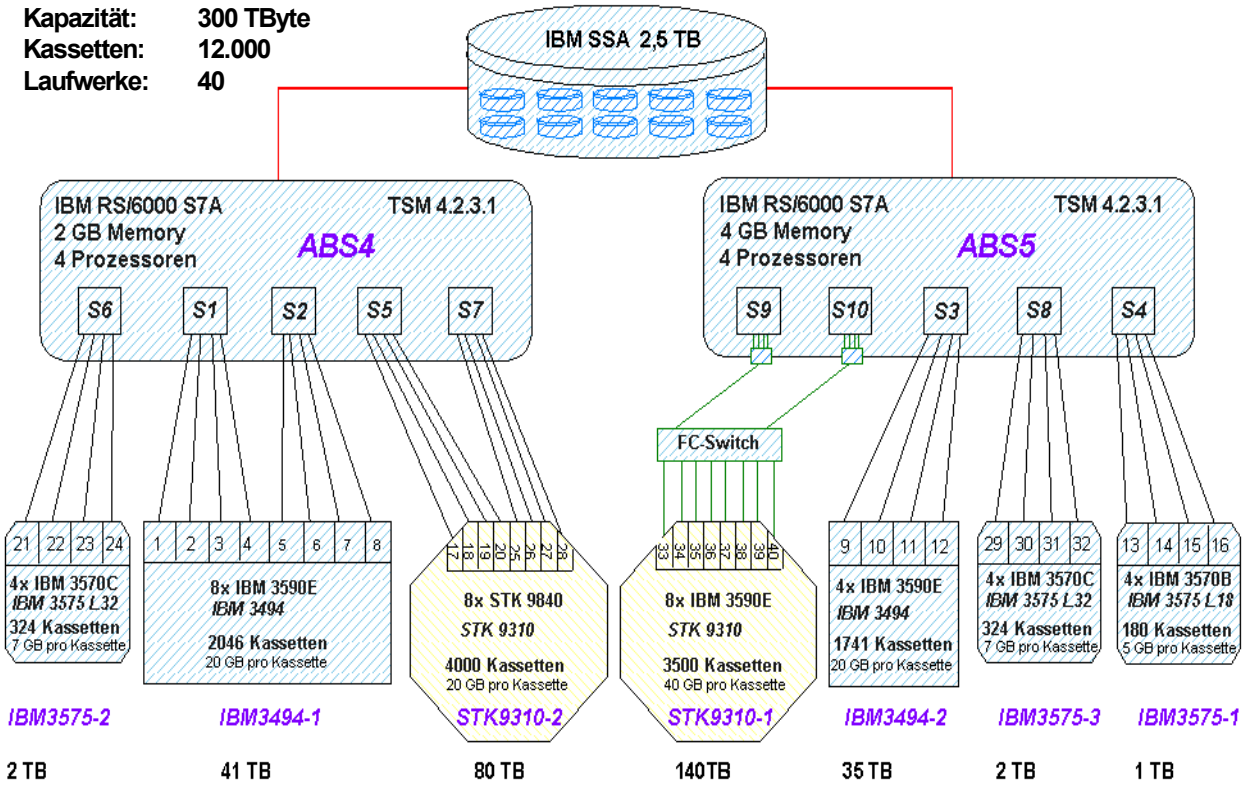


Abbildung 3.3: Konfiguration des TSM-Servers am LRZ München [LRZ03].

Eingesetzt werden 40 Bandlaufwerke der Firmen IBM und StorageTek (STK), die 12.000 Magnetbandkassetten mit einer Kapazität von insgesamt 300 TByte verwalten. Die Festplattenkapazität des Zwischenspeichers beträgt 2,5 TByte. Im November 2003 wurde durch weitere acht Bandlaufwerke der Marke LTO-2 eine neue Ausbaustufe mit 12.540 Kassetten und einer Gesamtkapazität von 408 TByte erreicht. Die Kapazität des Festplattenspeichers wurde

auf 5,5 TByte erweitert. Beim Backup sehr vieler Dateien sollte das Datenvolumen der Metadaten, die zur Verwaltung der gesicherten Dateien benötigt werden, nicht unterschätzt werden. Betrachtet man die in einem DBMS verwalteten Metadaten, der am LRZ gesicherten, bzw. archivierten Daten (über 1.249 Millionen), so weisen diese das beachtliche Volumen von 1 TByte auf (Stand Januar 2004).

Neben der Backup- und Archivierungsfunktionalität bietet TSM auch die Möglichkeit der HSM-Funktionalität. Die Komponente des HSM-Clients fand im bisherigen Nutzungsspektrum beim LRZ noch keine Verwendung und wurde erstmals in Zusammenhang mit HEAVEN eingesetzt. Eine ausführliche Beschreibung der Funktionsweise eines HSM-Systems ist in Kapitel 2.2.4 enthalten. Um die Auslagerung von Dateien einer lokalen Festplatte eines Rechners zu erreichen, wird auf diesem Rechner der HSM-Client des TSM installiert. Aus Gründen besserer Portierbarkeit des HSM-Clients auf unterschiedliche Betriebssysteme, wurde von IBM das Veritas Dateisystem (VxFS) vorausgesetzt (vergleiche Kapitel 2.2.4). Das bedeutet, dass nur Dateien, die auf einer Festplatte bzw. einer Partition mit Veritas-Dateisystem gespeichert werden, auf TS-Medien ausgelagert werden können. Die Anbindung von RasDaMan an den HSM-Client des TSM, erfolgt über die Schnittstelle des Veritas-Dateisystems. Der HSM-Client arbeitet als Hintergrundprozess (Dämon) und ermittelt in gewissen, einstellbaren Zeitabständen, Kandidaten für die Migration auf TS-Medien. Bei der Migration werden die Kandidaten vom lokalen Dateisystem in einen Zwischenspeicher (SHC_1) des TSM-Servers geladen, um Engpässe zu vermeiden. Bei der TSM-Implementierung fungiert das lokale Dateisystem als zusätzlicher Cache-Speicher (SHC_2) zwischen RasDaMan und den tertiären Speichermedien. Durch den TS-Cache (SRC) in RasDaMan und den beiden Cache-Bereichen SHC_1 und SHC_2 unter Kontrolle des HSM-Systems, ergibt sich eine dreistufige Caching-Hierarchie. Eine detailiertere Beschreibung folgt in Kapitel 3.9.
Die Migration der Dateien auf tertiäre Speicherbereiche (T_{on}, T_{near}, bzw. T_{off}), erfolgt durch den TSM-Server automatisch. Für den Anwender, bzw. einer Applikation, wie zum Beispiel RasDaMan, entsteht durch den gesamten Vorgang der Migration, vom lokalen Dateisystem zum TS-Medium, keine zeitliche Belastung. Erfolgt von einer Applikation ein Zugriff auf eine ausgelagerte Datei, wird eine Nachricht an den HSM-Dämon gesendet. Daraufhin wird die gewünschte Datei vom TS-System geladen und in das lokale Dateisystem gespeichert. Solange dieser Vorgang dauert, wird der eigentliche Systemaufruf blockiert. Nach der Beendigung des Rückholprozesses dieser Datei, erfolgen die Freigabe und ein ganz „normaler“ Dateizugriff.

Abbildung 3.4 zeigt beispielhaft die Speicherplatzverwendung des lokalen Veritas-Dateisystems einer SunBlade 150 und des serverseitigen TSM-Dateisystems als Tortendiagramm durch die grafische Benutzeroberfläche des HSM-Clients. Die Belegung des Speicherplatzes des Veritas-Dateisystems eines lokalen Rechners, wird im linken Diagramm dargestellt. Es sind Anteile des freien Speicherplatzes, dauerhaft gespeicherter (residenter) Dateien, vormigrierte (premigrated) Dateien und Dateiverweise (Stubs) aufgeführt. Bei vormigrierten Dateien wurde bereits eine Kopie auf TS-Medien geschrieben. Sie verweilen allerdings aus Gründen der Performance noch im lokalen Dateisystem (SHC_2). Treten im lokalen

Dateisystem Platzprobleme auf, so wird der Festplattenspeicherplatz vormigrierter Dateien freigegeben und durch einen Dateiverweis ersetzt. Das mittlere Tortendiagramm gibt Informationen über das Speichervolumen der bereits migrierten Dateien und des noch zur Verfügung stehenden Speicherplatzes. Jedem HSM-Knoten kann ein bestimmtes Kontingent (Quota) an Speichervolumen auf dem TSM-Server zugewiesen werden. In diesem Beispiel beträgt das Kontingent ca. 215 GByte (220.206 MByte). Eine Übersicht über die Verwaltungseinträge des Dateisystems der residenten, vormigrierten und migrierten Dateien des lokalen Dateisystems, ist in der rechten Darstellung zu finden. Diese Verwaltungseinträge werden bei Unix, bzw. Linux Betriebssystemen als *Inodes* (Index Nodes) realisiert. In der Dateiverwaltung beschreibt ein Indexknoten genau eine Datei. Neben den Informationen um alle Blöcke dieser Datei zu adressieren, sind Informationen über Eigentümer, Zugriffsrechte, Typ und Größe der Datei enthalten.

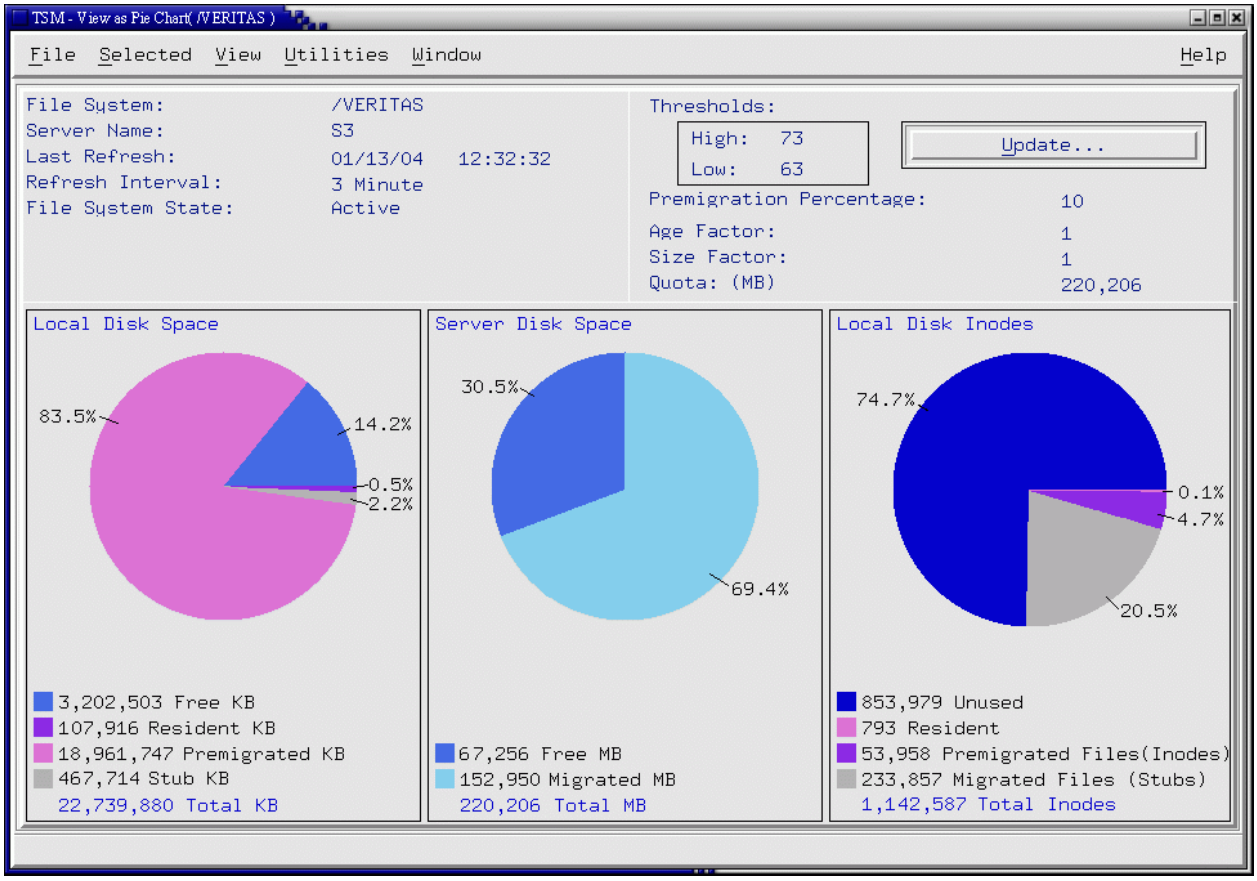


Abbildung 3.4: Speicherplatzverwendung des lokalen und des serverseitigen Dateisystems.

Optionen, die der Anpassung des HSM-Clients an spezielle Bedürfnisse dienen, können in den Konfigurationsdateien *dsm.sys* und *dsm.opt* eingetragen werden. Dazu zählt zum Beispiel die Möglichkeit, beim Migrieren der Daten gleichzeitig ein Backup zu erstellen. Weiterhin kann eingestellt werden, wie viele Hintergrundprozesse Aufträge quasi-parallel ausführen. Eine detaillierte Beschreibung ist bei [IBM03a, IBM03b] zu finden. Neben dem hier beschriebenen HSM-System, können andere HSM-Systeme, wie zum Beispiel der Storage Archive Manager (LSC Incorporation), FileServ (Adic Incorporation), DiskXtender (Legato

Systems Incorporation) und StorHouse Storage Manager (FileTek Incorporation) auf ähnliche Weise über die Dateisystemschnittstelle mit RasDaMan gekoppelt werden [Sun02, Adic03, Lega03a, CB00, CB01, CKK00]. Entsprechend einer grundlegenden Forderung aus dem ESTEDI-Projekt, wurde bei der Realisierung von HEAVEN auf eine einfache und unkomplizierte Anbindung von RasDaMan an eine Vielzahl konventioneller HSM-Systeme geachtet.

3.3 Anbindung eines Bandlaufwerkes

Neben der beschriebenen Verwendung beliebiger HSM-Systeme für die Anbindung tertiärer Speichermedien an RasDaMan, wurde im Rahmen der Entwicklung von HEAVEN, auch eine Möglichkeit geschaffen, Bandlaufwerke direkt zu verwenden. Dies wurde durch die Eigenentwicklung des *Tape-Controller-Toolkits* (TCT) realisiert. Das TCT wurde hauptsächlich in Hinblick auf die in Kapitel 4 folgende Performance Evaluierung von HEAVEN entwickelt. Ein wesentlicher Vorteil von TCT, gegenüber einem im operativen Betrieb befindlichen HSM-Systems, ist der exklusive Zugriff auf die TS-Medien. Damit werden Messergebnisse nachvollziehbar und reproduzierbar und müssen nicht aufgrund konkurrierender Nutzung gemeinsamer Ressourcen mehrmals wiederholt werden. Konventionelle HSM-Systeme bieten zum Beispiel auch nicht die Möglichkeit, detaillierte Messungen der Ladezeit, der Positionierzeit und der Medienwechselzeit durchzuführen. Es kann nur die gesamte Zeit eines Retrievalvorgangs ermittelt werden. Auch die Dauer der Migration einer Datei auf ein TS-Medium kann nicht ermittelt werden, da ein HSM-System keine Statusmeldung nach dem Schreiben der Datei auf ein TS-Medium liefert. Mit TCT können diese detaillierten Messungen ohne Probleme durchgeführt werden. Dies erleichtert die Überprüfung entwickelter Konzepte, da Faktoren, wie zum Beispiel konkurrierende Zugriffe, ausgeschlossen werden können.

Aus technischer Sicht, bildet das TCT die Funktionalität eines HSM-Systems nach. Allerdings ist eine engere Kopplung zwischen RasDaMan und TCT vorhanden. Dies bedeutet, dass RasDaMan seine Import- oder Export-Aufträge direkt an TCT über eine definierte Schnittstelle absetzt. Zum Transferieren der Daten von RasDaMan zum TCT, wird das Dateisystem als Festplatten-Cache (entspricht dem SHC) verwendet. TCT selbst läuft als Hintergrundprozess (Dämon), um beim Schreiben der Daten auf TS-Medien RasDaMan nicht zu blockieren. RasDaMan ist dadurch in der Lage, während dieses Prozesses parallel weitere Aufträge zu bearbeiten. Verglichen mit Abbildung 2.9 (Variante 2), entspricht die Anbindung tertiärer Speichermedien mittels TCT an RasDaMan einer Kopplung über einen TS-Cache, da Daten von RasDaMan direkt in den Festplatten-Cache des TS-Systems geschrieben werden. Mit Hilfe von *wxWindows*[19] wurde eine grafische Oberfläche zur Nutzerinteraktion erstellt. Abbildung 3.5 präsentiert diese grafische Oberfläche des TCT, mit einer Auswahl auf TS-Medien exportierter Dateien.

Um die Funktionsweise von HEAVEN mit einer größeren Datenmenge zu testen, wurden 48 Kollektionen mit einem gesamten Speichervolumen von 1.052 GByte (1,05 TByte) in Ras-

[19] wxWindows ist eine plattformübergreifende (open source) C++-Programmierbibliothek, um grafische Benutzeroberflächen zu erstellen (http://www.wxwindows.org).

DaMan eingefügt, auf TS-Medien exportiert und diverse Anfragen an das System gestellt. Jede dieser Kollektionen beinhaltet 17 MDD einer vierdimensionalen Klimasimulation des Deutschen Klimarechenzentrums, mit einer Größe von je 1,32 GByte. Zur Speicherung wurden Magnetbänder der Firma Sony, des Typs DLTtape IV[20], mit einer nativen Speicherkapazität von 20 GByte verwendet. Die aktivierte Hardware-Kompression des Quantum DLT-4000 Laufwerkes erreichte eine Komprimierungsrate von 3,37. Dadurch fanden jeweils 3 Kollektionen auf einem Magnetband Platz. Die benötigte Anzahl an TS-Medien beträgt 16 Stück, bei einem Speichervolumen von 1,05 TByte. Bei dem beschriebenen Szenario beträgt die festplattenseitige Datenbankgröße von RasDaMan 40 GByte. In einer realen Umgebung sollte diese Speichergröße allerdings sehr viel größer ausgelegt werden. HEAVEN zeigte keinerlei Probleme bei steigendem Datenvolumen hinsichtlich Stabilität. Ebenso waren keine Einbrüche in der Performance bei einem Datenvolumen von 1,05 TByte spürbar. Auch bei 100 TByte oder mehr sind keine Performanceeinbrüche zu erwarten. Eine gute Skalierbarkeit wird mit HEAVEN erreicht, was als entscheidende Grundvoraussetzung für den Einsatz im HPC-Bereich gilt. Bei einer aktuellen Magnetbandtechnologie (z.B. Super-DLT 600) mit 300 GByte nativer Speicherkapazität und gleicher Komprimierungsrate würde ein Magnetband zur Speicherung von 1,05 TByte[21] ausreichen.

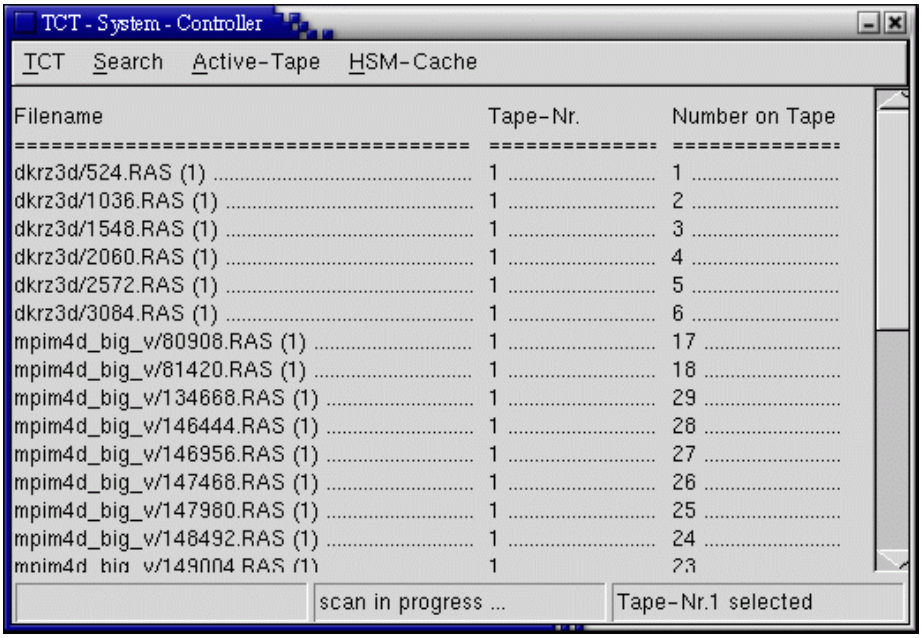


Abbildung 3.5: Die grafische Oberfläche des Tape-Controller-Toolkit (TCT).

Ein Nachteil der direkten Anbindung eines Bandlaufwerkes über TCT an RasDaMan, ist die Produktabhängigkeit. Für jedes Bandlaufwerk sind produktspezifische Einstellungen notwendig, um eine funktionierendes System zu realisieren. Durch die Anpassung gewisser Optionen lassen sich die meisten TS-Systeme mit SCSI-Schnittstelle an RasDaMan anbinden. Als

[20] Magnetbandtechnologie aus dem Jahr 1994. Aktuelle Technologie: Super-DLT 600 (mit 300 GByte nativ).
[21] Die Speicherkapazitäten im Bereich des HPC liegen in Regionen von mehreren hundert TByte (10^{12}) bis hin zu einigen PByte (10^{15}).

Beispiel ist hier ein Ausschnitt der TCT-Konfigurationsdatei (*TCTconfig.h*) für den Fünffach-Wechsler DLT-LXLS der Firma Overland Data mit einem integrierten Quantum DLT4000 Laufwerk aufgeführt:

```
//Schnittstelle für das Magnetbandlaufwerk:
#define TAPE_INTERFACE_N     "/dev/rmt/0cn"

//Konfigurationseinstellungen für die Positionierung
#define FORWARD_METHOD       0
#define FORWARD_MULTIPLIER   1
#define FORWARD_ADDEND       0

#define FWCORR_METHOD        0
#define FWCORR_VALUE         0

#define BACKWARD_METHOD      0
#define BACKWARD_MULTIPLIER  1
#define BACKWARD_ADDEND      1     // Voreinstellung: 0
#define BKCORR_METHOD        0
#define BKCORR_VALUE         0

#define REWIND_ACTIVE        0     // Voreinstellung: 1
#define EOM_ACTIVE           1

#define REWCORR_METHOD       0
#define REWCORR_VALUE        0

#define EOMCORR_METHOD       0
#define EOMCORR_VALUE        0

#define LOAD_CORRECTION      0     // Voreinstellung: 1
```

Zunächst ist es erforderlich, die systemabhängig Schnittstelle (TAPE_INTERFACE_N) des angeschlossenen Gerätes zu definieren. Bei Solaris ist es zum Beispiel die Schnittstelle /dev/rmt/0cn und bei Linux /dev/nst0. Speziell bei Spuloperationen ist es meist erforderlich Anpassungen vorzunehmen, da verschiedene Bandlaufwerke unterschiedlich reagieren. Soll zum Beispiel auf eine Datei zurückgespult werden, die n Positionen zurückliegt, ist es bei einigen Geräten erforderlich, n + 1 Positionen zurückzuspulen. Dies kann durch Ändern der Option BACKWARD_ADDEND von 0 auf 1 erreicht werden. Die weiteren Einstellungen sollen nur auf die Komplexität der Anbindung eines neuen TS-Systems hinweisen. Um das Ermitteln der richtigen Parameter zu erleichtern, wurde ein Skript erstellt, das durch verschiedene Schreib-, Lese- und Positionierungstests auf korrekte Parameter schließen lässt.

3.4 Optimierungspotential bei TS-Zugriffen

Nach Darstellung der HEAVEN Systemarchitektur und der Anbindung unterschiedlicher Systeme an RasDaMan folgt nun eine Diskussion über ein mögliches Optimierungspotential bei einem Datenretrieval von TS-Medien. Bestrebungen in dieser Richtung, sind aufgrund der vorhandenen großen Zugriffslücke (ca. 10^5) in Bezug auf Performance zwischen Sekundär- und Tertiärspeicher besonders sinnvoll. Bei unseren Betrachtungen gehen wir von einem Magnetband als TS-Medium aus, da im HPC-Bereich überwiegend diese Speichermedien

eingesetzt werden. Allerdings können die hier vorgestellten Überlegungen mit leichten Abwandlungen auf andere tertiäre Speichermedien (z.B. MO, CD, DVD) übertragen werden. Als Kriterium für den erzielten Performanzgewinn dient meist der so genannte *Speed-Up*:

Definition 3.3: Speed-Up (Beschleunigung, bzw. Performancegewinn)

Speed-Up beschreibt den Performanzgewinn, bezüglich der Ausführungszeit eines Systems mit optimierten Komponenten, gegenüber einem nicht optimierten System:

$$\text{Speed-Up} = \frac{\text{Laufzeit ohne Optimierung}}{\text{Laufzeit mit Optimierung}}$$

Also der Faktor, um den sich die Ausführungszeit einer Anfrage durch eine Optimierung gegenüber der ursprünglichen Version reduziert. Eine Steigerung der Performance lässt sich vor allem durch vier grundsätzliche Bestrebungen erreichen. Diese sind entsprechend des Optimierungspotentials absteigend sortiert und werden in den nachfolgenden Kapiteln näher erläutert:

Vermeidung, bzw. Reduzierung von Zugriffen auf TS-Medien:
Die größte Steigerung der Performance kann durch das Vermeiden, bzw. das Reduzieren von Zugriffen auf TS-Medien erreicht werden. Diese einleuchtende und zunächst trivial klingende Aussage bietet einige interessante Möglichkeiten der Optimierung. Beispielsweise kann ein für multidimensionale Array-Daten angepasstes Caching realisiert werden, um Zugriffe auf tertiäre Speichermedien zu vermeiden (Kapitel 3.9). Das entspricht einer Verlagerung der Anfragebeantwortung auf eine schnellere Speicherebene, innerhalb der vorhandenen Speicherhierarchie. Ein anderer Weg ist, das Laden der Daten nicht zu verlagern, sondern das zu ladende Datenvolumen zu reduzieren. Möglichkeiten hierfür sind die bereits in Kapitel 2.5.3 behandelte Kachelung von großen Objekten. Wie sich in Kapitel 3.5 zeigen wird, ist es sinnvoll, eine dem Speicherort (Festplatte oder TS-Medium) angepasste Datengranularität zu wählen. Eine wichtige Rolle in diesem Zusammenhang spielt auch eine für zukünftige Anfragemuster optimierte, Datenmodellierung. Dies kann durch die Wahl einer geeigneten Kachelungsstrategie erreicht werden. Als weitere Entwicklung wird Object-Framing als ein neues Analyseverfahren vorgestellt, das ebenfalls das zu ladende Datenvolumen reduziert (Kapitel 3.10). Weiterhin kann durch einen erweiterten Systemkatalog, der vorberechnete Aggregatsoperationen (z.B. Extremwert oder Durchschnitt) enthält, Zugriffe auf TS-Medien eingespart werden (Kapitel 3.11). Die Ergebnisse kostenintensiver Berechnungen, wie zum Beispiel der Extremwertbildung, werden in der Datenbank abgelegt und können bei weiteren Anfragen berücksichtigt werden. Die hier erwähnten Möglichkeiten schließen sich gegenseitig nicht aus, sondern können selbstverständlich beliebig kombiniert angewendet werden.

Minimierung von Medienwechsel:
Bei der Optimierung von TS-Zugriffen ist vor allem auf die Minimierung von Medienwechsel zu achten. Aufgrund der hohen Kosten von Medienwechsel sollte versucht werden, bei An-

fragen diese zu vermeiden. Zu den Kosten (12s – 40s) für das Einlegen eines Mediums in eine Schreib-/Lesestation, addiert sich noch die Positionierungszeit zum Auffinden des gewünschten Datensatzes. Diese ist stark abhängig von der Lage des Datenobjektes auf dem Magnetband. Im ungünstigsten Fall (ca. 160s)[22] muss vom Beginn des Mediums bis zum Ende positioniert (gespult) werden. Steht für den Zugriff keine freie Schreib-/Lesestation zur Verfügung, gibt es weiterhin zusätzliche Kosten für das Rückspulen (bis ca. 160s)[22] und das Auswerfen (10s – 20s) eines anderen tertiären Speichermediums. Auch ist die Verzögerung (10s – 30s) durch die Bewegung eines Roboterarmes zu berücksichtigen. Das ergibt im ungünstigsten Fall eine Verzögerung von ca. 410 Sekunden, bis das gewünschte Datenobjekt vom TS-Medium geladen werden kann. Diese Zahl zeigt deutlich, dass unnötige Medienwechsel unbedingt vermieden werden müssen. Durch geeignete Datenverteilung lassen sich Medienwechsel reduzieren. Es sollte versucht werden, logisch zusammengehörige Datenobjekte auf das gleiche TS-Medium zu speichern (Kapitel 3.5 und 3.6). Bei Anfragen werden diese oft gemeinsam angefasst. So können zum Beispiel alle MDD einer Kollektion auf ein gemeinsames Medium gespeichert werden. Werden trotz dieser Bestrebungen bei einer Anfrage, Datenobjekte von unterschiedlichen TS-Medien benötigt, so ist es sinnvoll, die Anfragewarteschlange (Query-Queue) vor der Anfragebearbeitung zu sortieren (Kapitel 3.7 und 3.8). Dies bedeutet, dass zunächst alle Datenobjekte, die sich auf einem TS-Medium befinden, geladen werden und anschließend die Datenobjekte, die auf einem anderem Medium gespeichert wurden. Dadurch reduzieren sich die Sprünge zwischen unterschiedlichen Medien. Der Vollständigkeit halber wird die Datenkompression erwähnt. Durch die Kompression von Daten finden mehr logisch zusammengehörige Datenobjekte auf einem TS-Medium Platz. Bei einer Anfrage steigt dadurch die Wahrscheinlichkeit, dass sich die Daten auf dem gleichen Band befinden. Eine weitere Strategie, die bei TS-Systemen mit mehreren Schreib-/Lesestationen Anwendung findet, ist die Bestrebung, ein Magnetband nach einem Zugriff, möglichst lange in der Station zu belassen. Der Grund hierfür ist die Annahme, dass in naher Zukunft ein weiterer Zugriff auf das gleiche Medium stattfinden wird. Dieses Verfahren wird auch als „*Lazy Eject*“ (träges Auswerfen) bezeichnet.

Minimierung von Operationen zum Positionieren:
Die neueste Generation der Bandlaufwerke erreicht Übertragungsraten von bis zu 36 MByte/s (Super-DLT 600, bei unkomprimierten Daten). Bei einer vom Hersteller angegebenen mittleren Kompressionsrate von 2 steigt dieser Wert auf 72 MByte/s. Vergleichen wir die Übertragungsraten von tertiären Speichermedien mit den Datentransferraten von Festplatten (30 bis 60 MByte/s), so ist leicht zu erkennen, dass der eigentliche Flaschenhals beim Retrieval der Daten von Magnetbändern nicht die Übertragungsrate, sondern die zeitlich aufwändigen Operationen des Bandwechsels und der Positionierung sind. Abbildung 3.6 zeigt den zeitlichen Verlauf der Produktentwicklung der DLT, bzw. Super-DLT Bandtechnologie bezüglich der mittleren Positionierungszeit[23], der Übertragungsrate (nativ) und der Kapazität (nativ).
Die in Abbildung 3.6 dargestellten Werte entstammen diversen Datenblättern der einzelnen Entwicklungsstufen [Quan03a, Quan03b, Quan96, GWM01, Degr01]. Eine zeitliche Produkt-

[22] Maximale Spulzeit eines aktuellen Magnetbandes, z.B. ein Super-DLT 600.
[23] Zeit, die benötigt wird, den Schreib-/Lesekopf eines Magnetbandes von Beginn bis zur Mitte zu positionieren.

entwicklung anderer Bandtechnologien, wie LTO und AIT verläuft ähnlich. Betrachten wir den Verlauf der Übertragungsrate und der Kapazität eines Magnetbandes, so zeigen diese einen quadratischen Anstieg. Dieser Anstieg geht konform mit der Verdopplung der Datendichte des magnetischen Materials alle 15 bis 18 Monate und ist äquivalent zum Moore'schen Gesetz[24]. Im Gegensatz zu dieser sehr positiven Entwicklung, steht der lineare Anstieg der mittleren Positionierungszeit. Erklären lässt sich dieser Trend durch eine Verlängerung der Magnetbänder. Eine Erhöhung der Datendichte reicht oft nicht aus, um die gewünschten Kapazitäten zu erlangen. Der Verlauf der Positionierungszeit ist als sehr negativ einzustufen. Eine fallende Kurve wäre wünschenswert, ist allerdings bei zukünftigen Entwicklungen nicht zu erwarten. So zeigt Abbildung 3.6 deutlich, dass die Zugriffszeit beim Retrieval von Daten, die sich auf Magnetbänder befinden, sehr stark von der Positionierungszeit dominiert wird. Das motiviert Bestrebungen, Operationen zum Positionieren auf TS-Medien zu minimieren.

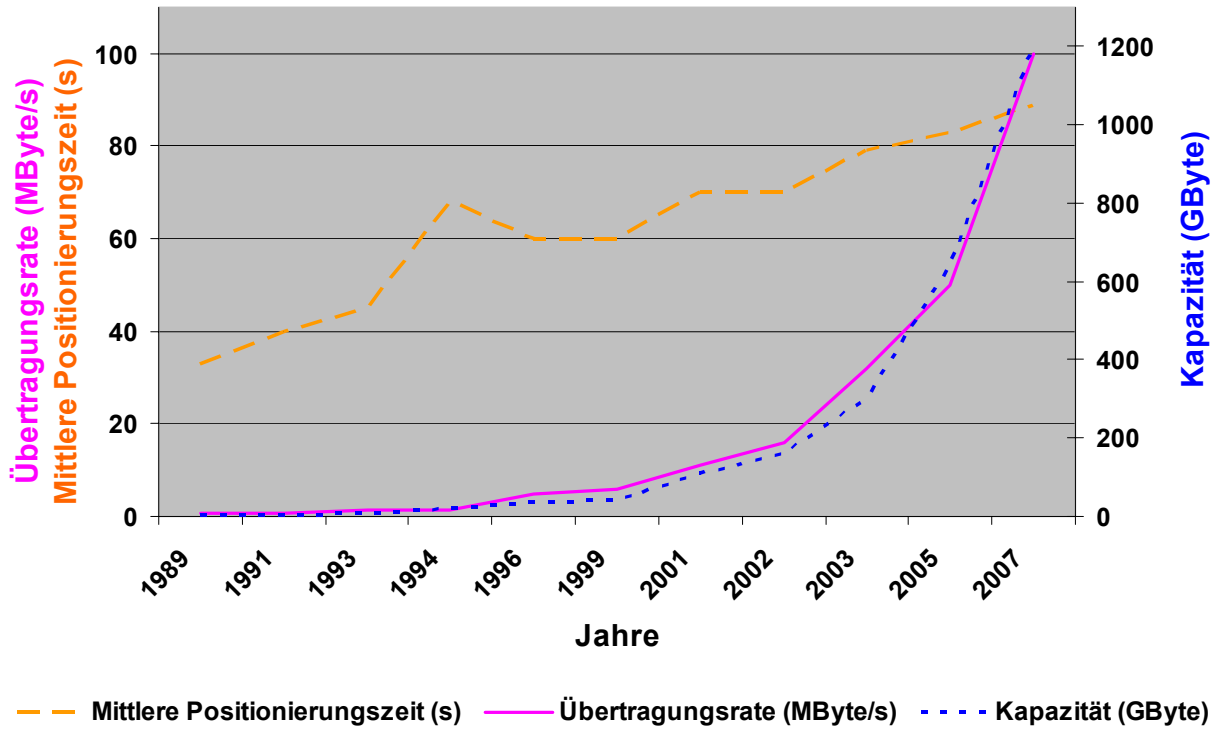


Abbildung 3.6: Verlauf der Entwicklung der DLT-, bzw. der Super-DLT-Bandtechnologie.

Eine sehr viel versprechende Möglichkeit, bietet das Clustering von multidimensionalen Array-Daten (Kapitel 3.5 und 3.6). Dabei wird versucht, die räumliche Nähe der Daten auch auf dem Magnetband beizubehalten. So kann vermieden werden, dass die Daten auf dem Magnetband „zufällig" verteilt vorliegen. Gerade bei Bereichsanfragen werden räumlich benachbarte Daten angefragt, die durch das Clustering auf dem TS-Medium in räumlicher Nähe liegen. So werden aufwändige Positionierungsoperationen minimiert. Bei Bereichsanfragen ist die Sortierung der Anfragewarteschlage eine weitere, wesentliche Möglichkeit der

[24] Die Beobachtung, dass sich durch den technischen Fortschritt die Komplexität von integrierten Schaltkreisen alle 18 Monate verdoppelt, heißt das Moore'sche Gesetz. Entdeckt von Gordon Moore 1965 [Moor65].

Optimierung (Kapitel 3.7 und 3.8). Die für eine Anfrage benötigten Datenobjekte werden in der gleichen Reihenfolge angefordert, wie sie in linearisierter Form auf das Magnetband geschrieben wurden. Dadurch erreicht man mit der Kombination aus Clustering und Sortierung der Anfragewarteschlange, eine optimale Performance durch minimale Positionierungszeiten. Auch eine dem jeweiligen Speicherort angepasste Datengranularität vermindert kostspielige Positionierung. Da gerade die Positionierungszeit bei einem Zugriff auf das Magnetband gegenüber der Übertragungsrate dominiert, ist es sinnvoll, die Datengranularität bei der Speicherung von Daten auf dem TS-Medium gegenüber der Festplatte zu erhöhen. Dadurch wird die gute Übertragungsrate ausgenutzt und die Anzahl der kostspieligen Positionierungsoperationen reduziert. Um dies zu erreichen, wurde in dieser Arbeit das Super-Kachel-Konzept entwickelt (Kapitel 3.5). Selbstverständlich spielt bei der Reduzierung der Positionierungszeit eine geeignete Datenmodellierung eine wichtige Rolle, wie auch schon bei den Bestrebungen zur Vermeidung von TS-Zugriffen (Kapitel 2.5.3).

Nutzung von Parallelität:
Wie bereits in Kapitel 1.2 beschrieben, war im Projekt ESTEDI neben der Tertiärspeicheranbindung das zweite zentrale Forschungsgebiet die Parallelisierung der Anfragebearbeitung für multidimensionale Array-Daten. Daher lag es nahe, diese Konzepte auch für HEAVEN zu nutzen, um eine Performancesteigerung bei der Anfragebearbeitung zu erreichen. Zunächst erfolgt mit Abbildung 3.7 eine Klassifizierung der parallelen Anfragebearbeitung.

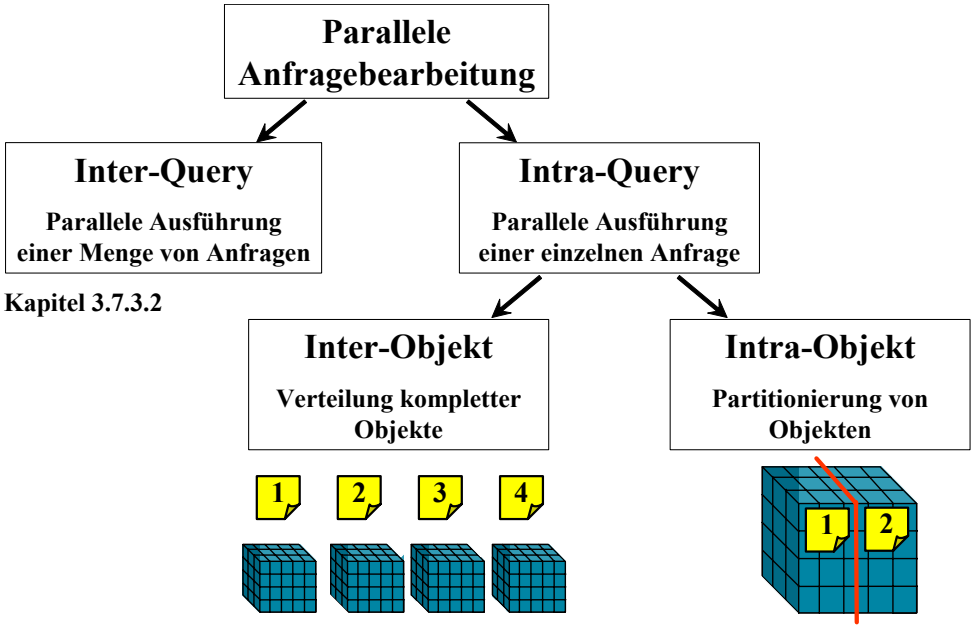


Abbildung 3.7: Klassifizierung der parallelen Anfragebearbeitung (nach [Rahm94]).

Bei der in Abbildung 3.7 gezeigten Inter-Query-Parallelität erfolgt eine parallele Verarbeitung auf der Granularität von Anfragen. Dadurch ist es möglich, eine Menge von Anfragen durch parallele Instanzen zu bearbeiten. Diese Art der Parallelisierung im Zusammenspiel mit HEAVEN wird in Kapitel 3.7.3.2 behandelt. Neben der parallelen Verarbeitung ganzer Anfra-

gen kann eine Parallelisierung innerhalb einer Anfrage (Intra-Query) erfolgen. Bei der Inter-Objekt-Parallelisierung werden einzelne Datenelemente durch parallele Instanzen verarbeitet. Im Gegensatz dazu wird bei der Intra-Objekt-Parallelisierung ein einzelnes Datenelement aufgeteilt und die Teilelemente parallel bearbeitet. Diese Art der Parallelität ist für relationale Daten in den seltensten Fällen sinnvoll. Meist ist ein relationales Tupel so klein, dass eine parallele Bearbeitung keinen Gewinn bringt. Als Ausnahme ist die parallele Verarbeitung von großen binären Objekten (LOB) in Relationen zu nennen [JM98, JM99, CO98, OCL99]. Da die Verarbeitung dieser Daten mit benutzerdefinierten Funktionen (*User Defined Functions*; UDF) erfolgt, ist eine Datenabhängigkeit nur schwer zu bestimmen. Das macht eine vernünftige Datenpartitionierung nur bedingt möglich. Im Gegensatz dazu ist bei RasDaMan die Möglichkeit der Intra-Objekt-Parallelität gut geeignet um eine beschleunigte Anfragebearbeitung zu erreichen. Für Array-Daten sind die Operationen und die Eingabeströme eindeutig einzuordnen. So ist die Intra-Objekt-Parallelisierung einfacher zu verwirklichen. Diese Thematik wird ausführlich in der Dissertation von Karl Hahn mit dem Titel „Parallele Anfrageverarbeitung für multidimensionale Array-Daten“ besprochen [Hahn05].

Neben der Verarbeitungsparallelität bei der Bearbeitung einer Anfrage, kann E/A-Parallelität durch parallele Hardware-Strukturen (mehrere Schreib-/Lesestationen) genutzt werden. Dafür gibt es spezielle Konzepte: Wie zum Beispiel „Lazy Eject“ und RAIT (*Redundant Arrays of Independent Tapes*). Analog zu RAID (Redundant Arrays of Independent Disks) für Festplatten dient RAIT bei TS-Systemen zur Erhöhung der Betriebssicherheit, der Datensicherheit und der Transferrate. Die Möglichkeiten solcher fehlertoleranten Systeme sind standardisiert und zu verschiedenen Levels zusammengefasst. So bietet beispielsweise RAID 1, bzw. RAIT 1 (Mirroring), eine redundante Datenhaltung auf mindestens zwei identische Medien. Neben der hohen Ausfallsicherheit kann beim Retrieval durch die Nutzung paralleler Hardware-Strukturen gleichzeitig mit zwei Lesestationen auf Daten zugegriffen werden. Nähere Angaben zu RAIT, bzw. RAID sind bei [Cher94, PGK88, GMW95, Mass97, HMD02] zu finden.

Nach der Darstellung der unterschiedlichen Einsparmöglichkeiten folgt eine Auflistung der in HEAVEN realisierten Optimierungen, mit Verweisen auf die entsprechenden Kapitel:

- Vermeidung, bzw. Reduzierung von Zugriffen auf TS-Medien:
  - Angepasste Datengranularität und Datenmodellierung (Kapitel 2.5.3 und 3.5).
  - Caching multidimensionaler Array-Daten (Kapitel 3.9).
  - Minimierung des vom TS-Medium zu ladenden Datenvolumens mittels Object-Framing (Kapitel 3.10).
  - Systemkatalog für vorberechnete Werte von Aggregatoperationen (Kapitel 3.11).
- Minimierung von Medienwechsel:
  - Geeignete Datenverteilung multidimensionaler Array-Daten (Kapitel 3.5 und 3.6).
  - Anpassung/Sortierung der Anfrage-Queue (Kapitel 3.7 und 3.8).
  - Kompression und Verfahren wie das „Lazy Eject“.
- Minimierung von Operationen zum Positionieren:
  - Angepasste Datengranularität und Datenmodellierung (Kapitel 2.5.3 und 3.5).

- Clustering multidimensionaler Array-Daten (Kapitel 3.5 und 3.6).
- Anpassung/Sortierung der Anfrage-Queue (Kapitel 3.7 und 3.8).

➢ Nutzung von Parallelität:

- Parallele Anfragebearbeitung in RasDaMan (Kapitel 3.7.2, 3.7.3 und [Hahn05]).
- Parallele Hardware-Strukturen und RAIT, bzw. RAID.

3.5 Datengranularität und Clustering bei TS-Speicherung

In diesem Kapitel wird beschrieben, wie durch eine der jeweiligen Speicherhierarchie angepassten Daten-, bzw. Zugriffsgranularität die bestehende Zugriffslücke zwischen Sekundärspeicher und Tertiärspeicher verringert werden kann. Weiterhin wird untersucht, wie räumliche Eigenschaften multidimensionaler Daten genutzt werden können, um effizient Zugriffe auf TS-Medien zu realisieren. Um dies zu erreichen, ist es sinnvoll, die Eigenschaften von RasDaMan und TS-Systemen hinsichtlich der jeweiligen Speichercharakteristika zu betrachten. Wie bereits in Kapitel 2.5.3 beschrieben, werden MDD in RasDaMan in gekachelter Form als BLOBs in einem relationalen DBMS gespeichert. Das hat den Vorteil, dass bei einer Bereichsanfrage nicht das gesamte MDD, sondern nur der gewünschte Ausschnitt (inkl. Verschnitt) geladen werden muss. Somit skaliert die Dauer der Anfrage mit der Größe der Bereichsanfrage und nicht mit der Objektgröße. Diese Eigenschaft war im HPC-Bereich bisher unüblich.

Die Granularität der in RasDaMan gespeicherten Kacheln ist für Festplattenzugriffe optimiert und beträgt ein Vielfaches einer physikalischen Datenbankseite. Bei der verwendeten Installation des Oracle-DBMS in der Version 8.1.7, wurde die Größe einer Datenbankseite auf den voreingestellten Wert von 8 KByte gesetzt. Normalerweise liegt die Größe einer Kachel zwischen 16 KByte und 1 MByte. Diese Granularität ist allerdings für TS-Zugriffe nicht besonders gut geeignet, da die Übertragungszeit von der Positionierungszeit stark dominiert wird. Bei einer Übertragungsrate von 36 MByte/s (Super-DLT 600), benötigt das Retrieval einer Kachel mit der Größe von 256 KByte nur 6,9 Millisekunden. Dem gegenüber steht eine mittlere Positionierungszeit von 76 Sekunden. Diese Diskrepanz zwischen den Kosten in der Datenübertragung und der Positionierung kann durch eine größere Datengranularität verringert werden. Abbildung 3.8 zeigt für unterschiedliche Speichermedien den Anteil Ψ der Positionierungszeit t_p gegenüber der Gesamtzeit t_g bei steigender Zugriffsgranularität. Dabei gilt: $0 \leq \Psi \leq 1$.

Mit steigender Zugriffsgranularität reduziert sich der Anteil der Positionierungskosten Ψ. Wie erwartet, zeigt die Kurve einer modernen Festplatte, mit einer mittleren Zugriffszeit (Positionierungszeit) von 6 ms und einer Transferrate von 50 MByte/s, einen ähnlichen Verlauf wie ein Magnetbandspeicher. Allerdings bei weit niedrigerer Seitengröße (Zugriffsgranularität). Bei einem Zugriff auf die Festplatte mit einer typischen Seitengröße von 8 KByte[25] oder 16 KByte, dominiert die Positionierungszeit mit über 95 % gegenüber der gesamten Bearbeitungszeit (Random Access). Um den gleichen Anteil bei Magnetbandsystemen zu erreichen,

[25] Standardeinstellung der logischen Blockgröße bei Solaris 9 mit *Unix File System* (UFS). Typische Seitengrößen variieren zwischen 2 KByte und 256 KByte, je nach Betriebssystem und verwendetem Dateisystem.

ist bei modernen Technologien, wie zum Beispiel einem Super-DLT 600 oder LTO-Ultrium 4600, eine Zugriffsgranularität von ca. 80 MByte bis 150 MByte zu wählen (obere Kurven Abbildung 3.8). Bei älteren Magnetbandlaufwerken (DLT 4000 und AIT 3) liegt die Granularität zwischen 6 MByte und 16 MByte. Stellen wir nun die in RasDaMan vorgeschlagenen Kachelgrößen von 16 KByte bis 1 MByte, der Zugriffsgranularität moderner Bandlaufwerke gegenüber, so wird es notwendig, eine zusätzliche Datengranularität einzuführen. Im Idealfall wird aus einer großen Anzahl an Zugriffen auf kleine Datenobjekte (z.B. Kacheln), mit den für jedes Datenobjekt anfallenden Positionierungskosten (Random Access), eine kleine Anzahl an Zugriffen auf größere, für TS-Medien optimierte Datenobjekte. Das bedeutet, dass neben der Abschwächung der Dominanz der Positionierungskosten gegenüber der Übertragungsrate, bei einem Zugriff auch die Anzahl der notwendigen Operationen zur Positionierung erheblich reduziert wird.

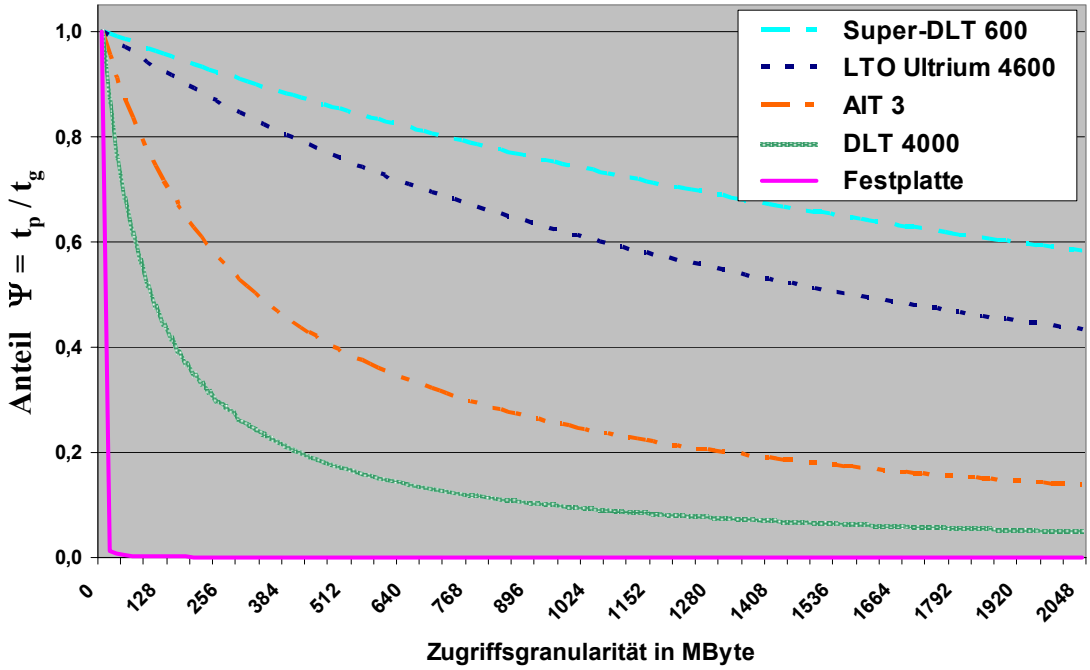


Abbildung 3.8: Anteil Ψ der Positionierungszeit t_p gegenüber Gesamtzeit t_g bei steigender Zugriffsgranularität.

Die gewählte Datengranularität bei TS-Medien sollte allerdings auch nicht zu hoch gewählt werden: Der Vorteil von RasDaMan, nur gewünschte Teilbereiche von MDD zu laden, muss erhalten bleiben. Auch die Kosten für den Einfügeprozess des zusätzlichen Datenvolumens, in den speziell für TS-Daten eingerichteten Festplatten-Cache-Bereich (Kapitel 3.9) des von RasDaMan verwendeten relationalen DBMS, sind nicht zu unterschätzen. Das führt zu einem angestrebten Kompromiss zwischen der Ausnutzung der guten Übertragungsrate von TS-Systemen und der Bestrebung, möglichst nur das für die Anfrage relevante Datenvolumen zu übertragen. Neben der in RasDaMan verwendeten Datengranularität einer Kachel, wird die Super-Kachel als neue, größere Datengranularität für TS-Medien eingeführt. Eine Super-Kachel ist ein Container, der eine, bzw. eine Vielzahl an Kacheln aufnimmt. Eine Super-

Kachel entspricht demzufolge einem Speicherobjekt auf TS-Medien. Die Ermittlung der optimalen Größe einer Super-Kachel behandelt Kapitel 3.5.4. Im Anschluss zur technischen Beschreibung folgen die formale Definition einer Super-Kachel und deren Eigenschaften:

Definition 3.4: Multidimensionale Super-Kachel (Super-Tile) σ

Eine Super-Kachel (Super-Tile) σ ist definiert als ein multidimensionales Teil-Array eines MDD α, mit gleicher Dimensionalität d, gleichem Basisdatentyp T und besteht aus einer Menge von Tupel (Elementen) der Form (Position $\underline{x}$, Wert $\nu(\underline{x})$), die durch die Abbildung f: $D_\sigma \rightarrow T$ definiert werden. Die Super-Kachel σ ist somit wieder ein MDD, wobei das multidimensionale Intervall $D_\sigma \leq D_\alpha$ ist. Eine Super-Kachel σ ist definiert als eine disjunkte Menge von Kacheln τ_i, wobei gilt $D_\sigma \geq D_\tau$:

$$\sigma := \left\{\left(\underline{x},\ \nu\left(\underline{x}\right)\right) \mid \nu\left(\underline{x}\right) \in T,\ \underline{x} \in D_\sigma\right\};\ \text{es gilt } dim(\sigma) = dim(\tau) = dim(\alpha)$$

$$sdom(\sigma) = \bigcup_{i=1}^{n} sdom\left(\tau_i\right)$$

$$sdom\left(\tau_i\right) \cap sdom\left(\tau_j\right) = \varnothing \text{ für } i, j \in \{1, ..., n\};\ i \neq j$$

Eine ausführliche Beschreibung, welche Kacheln zu einer Super-Kachel kombiniert werden, folgt in Kapitel 3.5.2, bzw. Kapitel 3.5.3.

Definition 3.5: Datenvolumen (Größe) Δ_σ einer multidimensionalen Super-Kachel σ

Das Datenvolumen Δ_σ in Byte, berechnet sich aus der Summe, der in einer Super-Kachel σ enthaltenen Menge an Kacheln τ_i:

$$\Delta_\sigma = \sum_{i=1}^{n} \Delta_{\tau_i}$$

für alle $\tau_i \cap \sigma \neq \varnothing;\ i \in \{1, ..., n\}$

Definition 3.6: Super-Kachelung (Super-Tiling) eines MDD α

Eine Menge von Super-Kacheln σ_k wird als Super-Kachelung (Super-Tiling) von MDD α definiert, falls folgende Eigenschaften gelten:

$$\bigcup_{k=1}^{m} sdom\left(\sigma_k\right) = sdom\left(\alpha\right)$$

$$sdom\left(\sigma_k\right) \cap sdom\left(\sigma_l\right) = \varnothing \text{ für } k, l \in \{1, ..., m\};\ k \neq l$$

Entsprechend der ersten Eigenschaft, ergibt die Menge aller Super-Kacheln σ_k das komplette MDD α. Die zweite Eigenschaft fordert eine disjunkte Menge der Super-Kacheln eines MDD α. Diese Einschränkung gilt, falls die in den Super-Kacheln enthaltenen Kacheln τ_i entspre-

chend einer regulären, ausgerichteten oder irregulären Kachelungsstrategie modelliert wurden (Kapitel 2.5.3). Aus Performancegründen wird bei einer arbiträren Kachelungsstrategie eine Überlappung der Super-Kacheln σ_k bei der physikalischen Speicherung zugelassen. Das entspricht einer konjunkten Partitionierung der Super-Kacheln σ_k eines MDD α und bedeutet, dass einzelne Kacheln redundant in mehreren Super-Kacheln gespeichert werden. Allerdings findet aus der Sicht des Datenmodells und den Operationen eine Partitionierung dieser Kacheln an Super-Kachel-Grenzen statt. Somit ist die Forderung nach Disjunktheit erfüllt. Zu der angestrebten Lösung gibt es zwei Varianten, die eine „echte" disjunkte Partitionierung realisieren. Jedoch weisen beide entscheidende Nachteile auf. Eine Vorgehensweise ist es, die entsprechende Kachel nur einer Super-Kachel zuzuschreiben, was sich allerdings bei einer Bereichsanfrage negativ auswirken kann, wenn eine zusätzliche Super-Kachel geladen werden muss, da genau der Bereich der Kacheln in einer bereits geladenen Super-Kachel fehlt. Eine andere Möglichkeit würde bedeuten, die Kachel an den Super-Kachel-Grenzen zu zerteilen um die einzelnen Partitionen in den Super-Kacheln abzulegen. Das wiederum bedeutet, dass die eindeutige Identifizierung der Kacheln durch Objektidentifikatoren (OIDs) ausgehebelt wird. Eine weiterführende Erläuterung folgt in Kapitel 3.5.2. Ebenso, wie bei der Kachelung kann die Super-Kachelung regulär, ausgerichtet, irregulär oder arbiträr ausgeprägt sein (Abbildung 2.16, Definition 2.12).

Definition 3.7: Menge I_σ der von einer Anfrage q betroffenen Super-Kacheln σ

Die für eine Anfrage q zu ladende Menge I_σ an Super-Kacheln σ eines MDD α, werden durch die Methode *intersect*(q, α) ermittelt. Es werden die zur Bearbeitung notwendigen Super-Kacheln bestimmt, die explizit von Tertiärspeicher (SHC, T_{on}, T_{near}, T_{off}) geladen werden müssen und Daten im Schnittbereich der Anfrage *sdom*(q) beinhalten. Es gilt:

$$I_{\sigma(q,\alpha)} = \bigcup_{i=1}^{n} \{\sigma_i \mid sdom(\sigma_i) \cap sdom(q) \neq \varnothing \wedge \sigma_i \in \alpha \wedge \sigma_i \in \alpha \wedge \sigma_i \in \{\Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\} \wedge \sigma_i \notin \{\Theta_P, \Theta_{SR}, \Theta_{SRC}\}\}$$

Es gilt $sdom(\alpha) \geq sdom(I_\sigma)$ und $sdom(\alpha) \geq sdom(q)$. Die Anzahl der durch die Anfrage q betroffenen Super-Kacheln ist durch $N_{I\sigma} = |I_\sigma|$ gegeben. Für reguläre und ausgerichtete Super-Kachelung erfolgt die Bestimmung von $N_{I\sigma}$ und die Berechnung der Anzahl $N_{I\sigma encl}$ der komplett (enclosed) in der Domäne der Anfrage liegenden Super-Kacheln wie folgt (Abbildung 3.9):

$$N_{I\sigma} = \prod_{i=1}^{d} \left\lceil \frac{extent(sdom(q))[i] + \left(low(sdom(q))[i] \bmod extent(sdom(\sigma))[i]\right)}{extent(sdom(\sigma))[i]} \right\rceil$$

$$N_{I\sigma encl} = \prod_{i=1}^{d} \left\lfloor \frac{extent(sdom(q))[i] - \left(extent(sdom(\sigma))[i] - \left(low(sdom(q))[i] \bmod extent(sdom(\sigma))[i]\right)\right)}{extent(sdom(\sigma))[i]} \right\rfloor$$

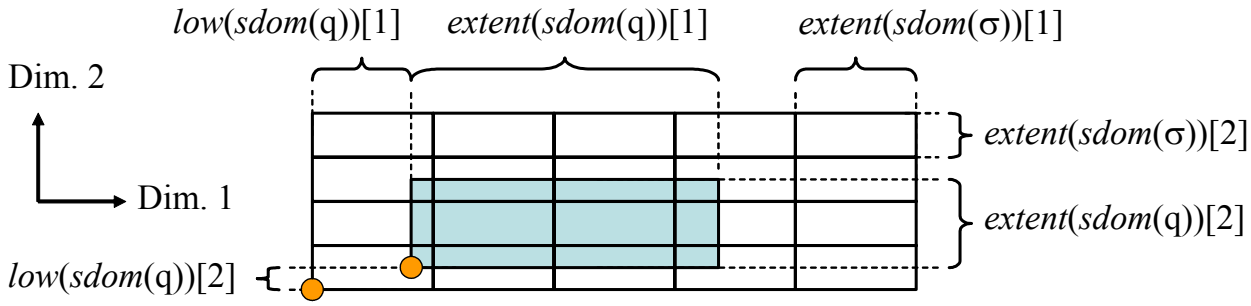


Abbildung 3.9: Berechnung der Anzahl der von einer Anfrage betroffenen Super-Kacheln.

Für das Beispiel aus Abbildung 3.9 wird für $N_{I\sigma} = 12$ und für $N_{I\sigma encl} = 2$ ermittelt, dabei gilt $N_{I\sigma} \geq N_{I\sigma encl}$. Die Methode *low*(*sdom*(q))[i] bezieht sich auf den Ursprung der Domäne des MDD α. Durch die Einführung der Datengranularität einer Super-Kachel erhöht sich gegenüber der kachelbasierten Speicherung meist der Verschnitt bei Bereichsanfragen, wenn diese nicht exakt auf die Grenzen der Super-Kacheln treffen. Der Verschnitt, der bei der Anfragebearbeitung hinsichtlich der Super-Kachel auftritt, ist definiert als:

Definition 3.8: Verschnitt υ_σ, bzw. $\upsilon_{\sigma\setminus\tau}$ bei einer Anfrage q an Super-Kachel partitionierte MDD α

Der Verschnitt υ_σ bei einer Anfrage (Queries) q, ist der Anteil des überschüssig gelesenen multidimensionalen Datenraumes, der von der Anfragen q geschnittenen Menge an Super-Kacheln σ_i, die explizit von Tertiärspeicher (SHC, T_{on}, T_{near}, T_{off}) geladen werden müssen. Der Verschnitt $\upsilon_{\sigma\setminus\tau}$ bezeichnet den durch die Super-Kachelung, gegenüber der Kachelung auftretenden zu viel gelesenen multidimensionalen Datenraum:

$$\upsilon_\sigma := \frac{\bigcup_{i=1}^{n} \left(sdom(\sigma_i) \setminus sdom(q) \right)}{\bigcup_{i=1}^{n} sdom(\sigma_i)}$$

$$\upsilon_{\sigma\setminus\tau} := \frac{\bigcup_{i=1}^{n} sdom(\sigma_i) \setminus \bigcup_{j=1}^{m} sdom(\tau_j)}{\bigcup_{i=1}^{n} sdom(\sigma_i)}$$

für alle $sdom(\sigma_i) \cap sdom(q) \neq \emptyset$; $sdom(\tau_j) \cap sdom(q) \neq \emptyset$; $sdom(\sigma_i) \cap sdom(\tau_j) \neq \emptyset$

$\sigma_i \in \{\Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$; $\sigma_i \notin \{\Theta_P, \Theta_{SR}, \Theta_{SRC}\}$; $i \in \{1, ..., n\}$; $j \in \{1, ..., m\}$

An dieser Stelle soll angemerkt werden, dass der eventuell auftretende zusätzliche Verschnitt $\upsilon_{\sigma\setminus\tau}$, der durch die höhere Granularität der Super-Kacheln gegenüber dem Verschnitt der Kacheln υ_τ (Definition 2.11) entsteht, keine Auswirkungen auf die spätere Anfragebearbeitung im RasDaMan-Server hat. Erreicht wird dies durch die Rückgewinnung der einzelnen

Kacheln aus den angefragten Super-Kacheln. Die Kacheln werden in einem speziell für TS-Daten eingerichteten Festplatten-Cache-Bereich SRC des von RasDaMan verwendeten relationalen DBMS, gespeichert. Durch diese Methode ist der RasDaMan-Server in der Lage, Anfragen auf der Basis von Kacheln zu beantworten. Der Verschnitt υ_σ und $\upsilon_{\sigma\setminus\tau}$ beinhaltet nur den überschüssig gelesenen Datenraum, der beim expliziten Laden der Super-Kacheln von Tertiärspeicher auftritt. Aus diesem Grund kann der Verschnitt bei Multi-Anfragen, die TS-Daten betreffen, gegenüber dem Verschnitt bei Einzelanfragen geringer ausfallen. Dies tritt ein, wenn mehrere Anfragen gleiche Super-Kacheln benötigen. Bereits von einer Anfrage geladene Super-Kacheln, werden im Festplatten-Cache-Bereich SRC zwischengespeichert und müssen nicht erneut von TS-Medien geladen werden (Kapitel 3.9). Nach der formalen Definition folgen Möglichkeiten zur Nutzung räumlicher Eigenschaften von MDDs.

3.5.1 Clustering multidimensionaler Daten

Clustering ist in Verbindung mit Indexierung eine sehr viel versprechende Möglichkeit, performantes Datenretrieval zu realisieren. Durch Clustering wird versucht, die Anzahl der Zugriffe, bzw. der wahlfreien Zugriffe, zu reduzieren. Somit werden teure Operationen zur Positionierung vermieden. Die wesentliche Zielsetzung von Clustering ist es, Datenobjekte, die normalerweise gemeinsam angefragt werden, auf physikalischer Ebene möglichst in räumlicher Nähe zu speichern. Es wird dabei die logische, räumliche Nähe bei der physikalischen Speicherung von Daten genutzt und somit eine Überführung einer logischen Ordnung in eine physikalische Ordnung vorgenommen. Die Anzahl wahlfreier Zugriffe wird reduziert und möglichst durch sequentielle Zugriffe ersetzt. Das spielt besonders bei der Speicherung von Daten auf langsamen TS-Medien zur Steigerung der Performance eine entscheidende Rolle. Allerdings ist es nicht möglich, ein perfektes Clustering für alle auftretenden Anfragetypen zu finden. Das bedeutet, dass ein gewähltes Clustering für eine Klasse von Anfragen sehr gut geeignet ist. Für andere Anfragen kann es unter Umständen eine schlechtere Performance als bei der herkömmlichen Datenverteilung zeigen.

Eine wesentliche Forderung bei der Speicherung multidimensionaler Daten (MDD), ist die Aufrechterhaltung der räumlichen Nähe der einzelnen Punkte (Array-Zellen) im Datenraum, da typischerweise Bereichsanfragen gestellt werden. Bei der Umsetzung der Kachelung von MDD wurde dies berücksichtigt. So wird bei der Kachelung durch die disjunkte Partitionierung des Datenraumes, eine Menge benachbarter Punkte in den multidimensionalen Kacheln gespeichert. Abbildung 3.10 zeigt die Nachbarschaft einzelner Punkte im ein- bis dreidimensionalen Raum (1D, 2D und 3D). Jeder Punkt (ausgenommen Randpunkte) in einem multidimensionalen Objekt der Dimensionalität d hat 3^d-1 direkte, räumliche Nachbarn. Bei steigender Dimensionalität erhöht sich die Anzahl der räumlichen Nachbarn enorm. Zum Beispiel beim Übergang von zwei auf drei Dimensionen, erhöht sich die Anzahl von 8 auf 26 räumliche Nachbarn. Wird die Kachelung eines MDD betrachtet, so stehen Kacheln ebenso in räumlicher Nachbarschaft zueinander. Bei regulärer, ausgerichteter und irregulärer Kachelungsstrategie, hat jede Kachel eines MDD der Dimensionalität d maximal 3^d-1 räumliche Nachbarn (Abbildung 3.10). Randkacheln verfügen über entsprechend weniger räumliche Nachbarn. Bei

arbiträrer Kachelungsstrategie können, abhängig von Lage und Domäne der benachbarten Kacheln, weniger oder mehr räumliche Nachbarn vorhanden sein (Abbildung 2.16).

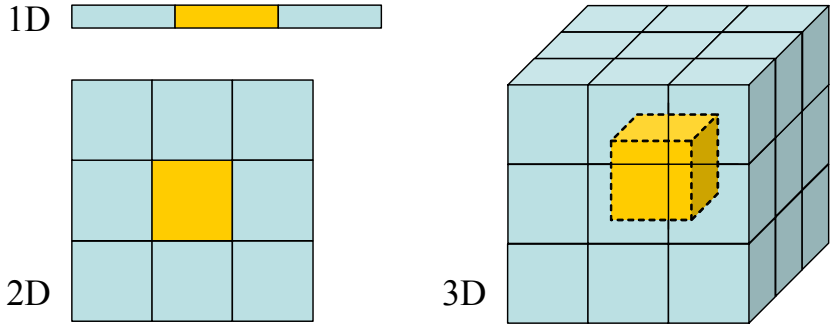


Abbildung 3.10: Direkte Nachbarschaft einer Zellen im ein- bis dreidimensionalen Raum.

In dieser Arbeit werden die Möglichkeiten des Clustering genutzt, um ein effizientes Retrieval von Daten, die auf TS-Medien gespeichert wurden, zu realisieren. Im Zusammenhang mit dem Super-Kachel-Konzept werden zwei grundlegende Arten von Clustering unterschieden, die wie folgt definiert sind:

Definition 3.9: Intra-Super-Tile-Clustering

Beim *Intra-Super-Tile-Clustering* wird die räumliche Nachbarschaft einer Menge von Kacheln τ_i eines MDD α bei der Verschmelzung von Kacheln zu einer Super-Kachel σ aufrechterhalten.

Definition 3.10: Inter-Super-Tile-Clustering

Beim *Inter-Super-Tile-Clustering* wird die räumliche Nachbarschaft einer Menge von Super-Kacheln σ_i eines MDD α bei der physikalischen Speicherung aufrechterhalten.

Bei aktuellen Speichermedien, wie zum Beispiel Festplatte und Magnetband, ist es notwendig, n-dimensionale Daten physikalisch auf eine Dimension abzubilden. Dadurch reduzieren sich die direkten räumlichen Nachbarn auf zwei Kandidaten. Abbildung 3.11 vergleicht unterschiedliche Clustering-Methoden bei einem Datenzugriff auf Magnetband. Dargestellt ist ein auf Magnetband gespeichertes MDD mit den Kacheln τ_1 bis τ_{12}. Jeweils vier Kacheln werden zu einer Super-Kachel kombiniert, was der kleinsten Zugriffsgranularität bei TS-Medien entspricht. Das optimale Clustering ist durch eine aufsteigende, lexikalische Ordnung der Kachelbezeichner gegeben. Eine Bereichsanfrage q umfasst die Kacheln τ_7, τ_8, τ_9 und τ_{10}, welche in aufsteigender Ordnung geladen werden. Als Kriterien für die Güte von Clustering-Methoden, gelten das zu ladende Datenvolumen, die Anzahl und die Dauer der benötigten Positionierungs- und Ladeoperationen.
Im ersten Fall wurde kein Clustering vorgenommen und alle Datenobjekte zufällig auf Magnetband verteilt. Um der Bereichsanfrage q mit $I_\tau = \{\tau_7, \tau_8, \tau_9, \tau_{10}\}$ zu genügen, müssen alle

drei Super-Kacheln vom Magnetband geladen werden. Aufgrund der Abarbeitungsreihenfolge der Anfrage, sind drei Positionierungsoperationen notwendig. Der zweite Fall realisiert Intra-Super-Tile-Clustering und fasst räumlich benachbarte Kacheln zu Super-Kacheln zusammen (Kapitel 3.5.2 und 3.5.3). Allerdings werden die Super-Kacheln nicht geordnet auf Magnetband geschrieben. Das zu übertragende Datenvolumen reduziert sich auf zwei Super-Kacheln. Es verbleiben je zwei Operationen zum Positionieren und zum Laden der Datenobjekte. Im dritten Fall wurde Inter-Super-Kachel-Clustering (Kapitel 3.6) verwendet und Kacheln, wie in Fall 1, zu Super-Kacheln kombiniert. Allerdings werden diese Super-Kacheln nach der ersten Kachel geordnet auf Magnetband geschrieben. Im Vergleich zur nicht geclusterten Speicherung, sind jedoch nur noch je zwei Operationen zum Positionieren und zum Laden notwendig. Dabei werden direkt aufeinander folgende Ladeoperationen (2a und 2b) als ein sequentielles Laden betrachtet. Das zu ladende Datenvolumen konnte, wie bei Fall 2, reduziert werden. Zu einem sehr guten Ergebnis führt die Kombination aus Inter- und Intra-Super-Tile-Clustering (Fall 4). Es ist nur noch je eine Operation zum Positionieren und zum Laden der Super-Kacheln notwendig.

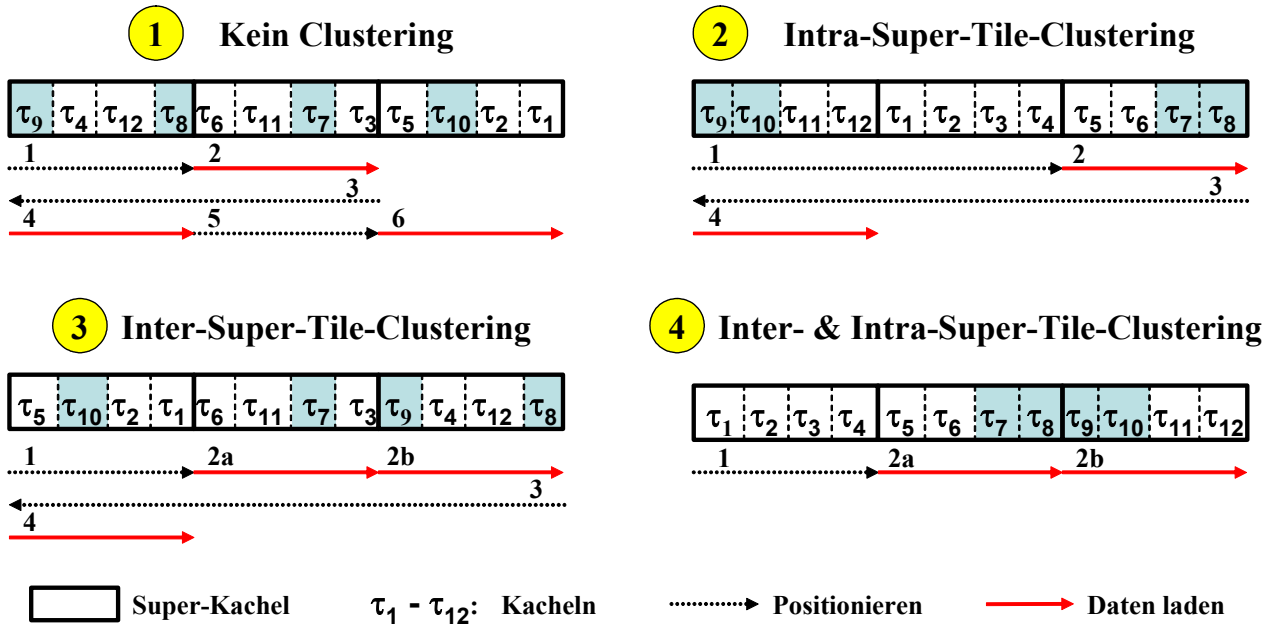


Abbildung 3.11: Vergleich unterschiedlicher Clustering-Methoden (Fall 1 bis 4).

Bei den bisherigen Betrachtungen wurde nur das Einlagern der Super-Kacheln von Magnetband in den Festplatten-Cache-Bereich (SRC) von RasDaMan betrachtet. Die für eine Anfragebearbeitung zusätzlich benötigten Operationen, zum Laden und Positionieren der einzelnen Kacheln von der Festplatte (SRC) in den Primärspeicher (P), wurden in Abbildung 3.11 nicht eingezeichnet. Wird angenommen, dass die Kacheln aus den Super-Kacheln in der gegebenen Ordnung auf die Festplatte geschrieben werden, so sind in den Fällen 1 und 3 mehrere Positionierungsoperationen (wahlfreie Zugriffe) erforderlich, da die benötigten Kacheln verteilt vorliegen. Im Fall 2 können jeweils zwei Kacheln und im Fall 4 sogar alle Kacheln sequentiell von der Festplatte gelesen werden.

Zusammenfassend lässt sich verallgemeinernd sagen, dass durch Clustering, wahlfreie Zugriffe eingespart werden können, bzw. durch sequentielle Zugriffe ersetzt werden. Auch das zu ladende Datenvolumen kann reduziert werden. Besonders bei einem Datenretrieval von tertiären Speichermedien, lässt sich durch ein geeignetes Clustering eine enorme Steigerung der Performance durch Einsparung von Positionierungsoperationen erreichen. Im folgenden Kapitel wird aufgezeigt, wie Intra-Super-Tile-Clustering realisiert wurde. Die Umsetzung von Inter-Super-Tile-Clustering folgt in Kapitel 3.6.

3.5.2 Super-Tile-Algorithmus (STAR)

Dieses Kapitel beschreibt den Super-Tile-Algorithmus (*Super-Tile-Algorithm*, bzw. STAR), der speziell für HEAVEN entwickelt wurde, um Zugriffskosten beim Datenretrieval von tertiären Speichermedien zu minimieren. Das verlangt neben der Einführung der Super-Kachel als weitere Speichergranularität für TS-Medien eine geschickte Verschmelzung einzelner Kacheln zu diesen Super-Kacheln.

Bei der Speicherung von MDD auf TS-Medien wird die Super-Kachel als optimierte Speichergranularität gewählt. Die einzelnen Kacheln sollen dabei entsprechend ihrer räumlichen Lage innerhalb des Datenraumes zu Super-Kacheln kombiniert werden. Um dieses Intra-Super-Tile-Clustering zu erreichen, nutzt STAR Eigenschaften, welche durch die in RasDaMan realisierte, multidimensionale Indexstruktur bereits vorgegeben sind. Wie bereits in 2.5.4 behandelt, verwendet RasDaMan einen R^+-Baum zur Indexierung der MDD. Der R^+-Baum ermöglicht die Indexierung von d-dimensionalen, achsenparallelen Rechtecken. Daher lassen sich multidimensionale Kacheln gut durch diese Indexstruktur verwalten. Einzelne Kacheln werden direkt als Datenrechtecke interpretiert. Aufgrund des Speichervolumens einer Kachel (16 KByte bis 1 MByte), das ein Vielfaches an Datenbankseiten einnimmt, entspricht ein Datenrechteck genau einer Datenregion. Blattknoten enthalten Objektidentifikatoren, die auf die jeweiligen Datenobjekte (Kacheln, bzw. BLOBs) in der Datenbank verweisen. Die multidimensionale Domäne der Blattknoten, entspricht dabei der vereinigten Domäne der referenzierten Kacheln. Innere Knoten der Baumstruktur mit ihrer Verzeichnis-Region, fassen wiederum Regionen zusammen, die nach dem gleichen Prinzip wie Datenregionen organisiert sind. Die einzelnen Regionen einer Ebene dürfen sich beim R^+-Baum nicht überlappen. Von der Wurzel des R^+-Baumes aus betrachtet, wird die Domäne eines multidimensionalen Objektes durch die einzelnen Ebenen in hierarchisch geschachtelte, disjunkte Sub-Domänen zerlegt. Abhängig von der Hierarchieebene werden mehr oder weniger räumlich benachbarte Datenregionen überdeckt.

Diese Eigenschaften der hierarchischen Schachtelung kann für die Generierung der Super-Kacheln genutzt werden. Dabei wird entsprechend des gewünschten Datenvolumens einer Super-Kachel ($\geq$ 50 MByte) eine Hierarchieebene im R^+-Baum gewählt und dieser innere Knoten als Super-Kachel-Knoten (Super-Tile-Node) ϕ bestimmt. Super-Kachel-Knoten werden im Folgenden auch als Super-Knoten bezeichnet. Alle von der Sub-Domäne eines Super-Knoten überdeckten Datenelemente, werden zu einer Super-Kachel kombiniert. Die enthalte-

nen Kacheln befinden sich in räumlicher Nachbarschaft zueinander und realisieren somit das Intra-Super-Tile-Clustering.

Die linke Seite der Abbildung 3.12 zeigt beispielhaft den R^+-Baum eines zweidimensionalen Objektes mit 16 regulär gekachelten Datenobjekten. Knoten verschiedener Super-Kacheln können dabei auf unterschiedlichen Ebenen des R^+-Baumes liegen. Als generelle Einschränkung des entwickelten Super-Tile-Algorithmus gilt, dass nur ein Super-Knoten im Pfad von der Wurzel bis zu den Blättern des Baumes enthalten sein darf. In unserem Beispiel der Abbildung 3.12 (linke Seite) haben wir fünf Super-Kachel-Knoten (ϕ_1 bis ϕ_5). Dabei liegen die Knoten ϕ_1 bis ϕ_3 auf der Ebene 2 und die beiden Knoten ϕ_4 und ϕ_5 auf der Ebene 3 (Blattknoten). Für die erste Super-Kachel mit Super-Kachel-Knoten ϕ_1 werden vier Datenobjekte (Nr. 1, 2, 5 und 6) zusammengefasst. Die fünfte Super-Kachel mit Super-Kachel-Knoten ϕ_5 enthält dagegen nur zwei Datenobjekte (Nr. 12 und Nr. 16).

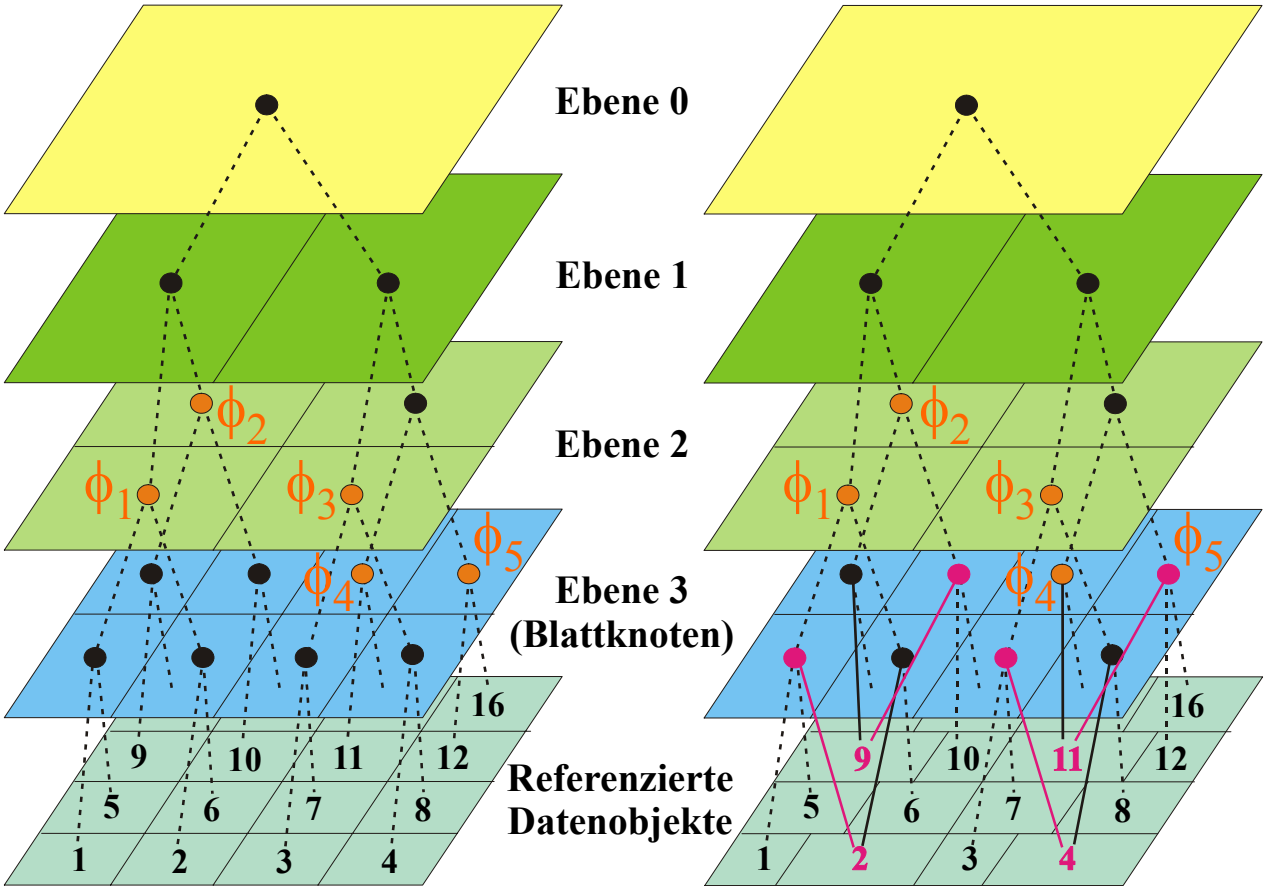


Abbildung 3.12: Lage der Super-Kachel-Knoten ϕ_1 bis ϕ_5 innerhalb eines R^+-Baumes.

Besonderheiten bei arbiträrer Kachelung

Eine Besonderheit tritt auf, wenn MDDs in RasDaMan mit der arbiträren Kachelungsstrategie modelliert und eingefügt werden. In diesem Fall ist es meist nicht möglich, Verzeichnis-Regionen durchgehend auf Datenregionen, bzw. Kachelgrenzen zu legen. Bei regulärer, aus-

gerichteter oder irregulärer Kachelungsstrategie, ist das einfach zu realisieren: Kachelgrenzen verlaufen immer entlang einer Dimension durch das gesamte Objekt (Definition 2.12, Abbildung 2.16). Bei arbiträrer Kachelung gilt diese Einschränkung nicht. Was dazu führt, dass Datenregionen eine Überlappung mit mehreren Verzeichnis-Regionen aufweisen können. In diesem Fall werden Datenregionen partitioniert (Clipping). Eine Datenregion wird, entsprechend den Regionsgrenzen, unterteilt und mehreren Regionen zugeordnet, bzw. von diesen referenziert.

Die rechte Seite der Abbildung 3.12 zeigt ein Beispiel eines zweidimensionalen MDD mit arbiträrer Kachelung. Es ist zu erkennen, dass einige physikalische Datenobjekte (Nr. 2, 4, 9 und 11) von mehreren Verzeichnis-Regionen der Ebene 3 referenziert werden (durchgezogene Linien). Die Super-Knoten liegen, wie bei der regulären Kachelung, an der gleichen Stelle des R^+-Baumes. Bei näherer Betrachtung der Unterbäume dieser Super-Knoten, ist zu erkennen, dass ϕ_4 und ϕ_5 der Ebene 3 jeweils einen Verweis auf die Datenregion 11 enthalten. Um das Problem zu lösen gibt es mehrere Möglichkeiten. Eine tatsächliche, physikalische Partitionierung dieser Objekte an den Super-Kachel-Grenzen wird nicht realisiert. Das würde die modellierte Kachelungsstrategie aufbrechen. Auch eine eindeutige Identifizierung über Objektidentifikatoren (OIDs) dieser partitionierten Datenobjekte kann in diesem Fall nicht mehr garantiert werden. Daher stellt sich die Frage, welcher Super-Kachel (Nr. 4 oder 5) das Datenobjekt 11 zugeschrieben werden soll? Aus Gründen der Performance wurde entschieden, dass das Datenobjekt 11 in beiden Super-Kacheln redundant gespeichert wird. Gerade bei einer Speicherung auf langsamen TS-Medien, ist es wenig sinnvoll, das Datenobjekt, zum Beispiel der Super-Kachel ϕ_4, mit der höheren Überdeckung zuzuorten. Dadurch wird es notwendig, bei einer Bereichsanfrage, die nur die Domäne der Super-Kachel ϕ_5 betrifft, zusätzlich ϕ_4 zu laden, da das Datenobjekt 11 nur einmal in ϕ_4 gespeichert wurde. Um ein effizienteres Retrieval zu realisieren, nimmt man den zusätzlichen Speicheraufwand (16 KByte bis 1 MByte) der redundanten Speicherung von Kacheln in Kauf[26]. Betrachten wird hingegen die Super-Knoten ϕ_1 bis ϕ_3, so ist zu erkennen, dass nicht zwangsweise eine redundante Speicherung notwendig ist. Die Domäne der Super-Knoten der Hierarchieebene 2 umfasst vollständig die doppelt referenzierten Datenobjekte.

Zusammenfassend lässt sich sagen, dass bei regulärer, ausgerichteter oder irregulärer Kachelung eine disjunkte Partitionierung des Datenraumes eines MDD vorgenommen wird. Bei arbiträrer Kachelung ist auf physikalischer Ebene eine konjunkte Partitionierung[27] mit redundanten Datenobjekten wahrscheinlich. Aus der Sicht des Datenmodells liegt aufgrund der logischen Partitionierung der Kacheln an Super-Kachel-Grenzen ebenfalls eine disjunkte Partitionierung vor, auch wenn auf physikalischer Ebene Kacheln redundant gespeichert werden. Anzumerken ist noch, dass die arbiträre Kachelung, aufgrund der komplexen Einfügeprogramme in der Praxis nicht verwendet wird. Nach der allgemeinen Beschreibung von STAR folgt eine formale Definition:

---

[26] Aufgrund der sehr geringen Praxisrelevanz der arbiträren Kachelung wird auf eine weiterführende Betrachtung des entstehenden Overheads bei der Speicherung verzichtet.

[27] Bei einer konjunkten Partitionierung können sich die einzelnen Partitionen überlappen. Das bedeutet, es gibt redundante Teilbereiche in den Partitionen.

Definition 3.11: Super-Tile-Algorithmus (STAR)

Der Super-Tile-Algorithmus weist folgende Eigenschaften auf:

1. Eine Menge von Super-Knoten ϕ eines R^+-Baumes, mit den Domänen $D_{\phi i}$, definieren das gesamte MDD α, wobei Super-Knoten auf unterschiedlichen Ebenen des R^+-Baumes liegen können:
   $$sdom(\alpha) = \bigcup_{i=1}^{n} D_{\phi_i}, \text{ wobei gilt: } D_{\phi_i} \cap D_{\phi_j} = \varnothing \text{ für } i, j \in \{1, ..., n\}; i \neq j$$
2. Auf dem Pfad von den Blättern bis zur Wurzel darf nur ein Super-Knoten enthalten sein.
3. Die Domäne D_σ einer Super-Kachel wird definiert durch die Domäne D_ϕ eines Super-Knotens des R^+-Baumes:
   $D_\sigma = D_\phi$
   Somit ergibt die Menge aller Super-Kacheln σ das komplette MDD α (Definition 3.6).
4. Alle Datenobjekte (Kacheln), die von D_ϕ, bzw. D_σ überdeckt werden, bilden eine Super-Kachel (Definition 3.4).

Um die entwickelten Konzepte besser evaluieren zu können, wurde das Visualisierungsprogramm *RasDa-Visualizer* implementiert. So ist es möglich, die Speicherstrukturen in der Granularität von Kacheln und Super-Kacheln zu zeigen. Abbildung 3.13 zeigt die Lage der Super-Kacheln (rechts oben) und Kacheln (rechts unten) in einem dreidimensionalen Objekt.

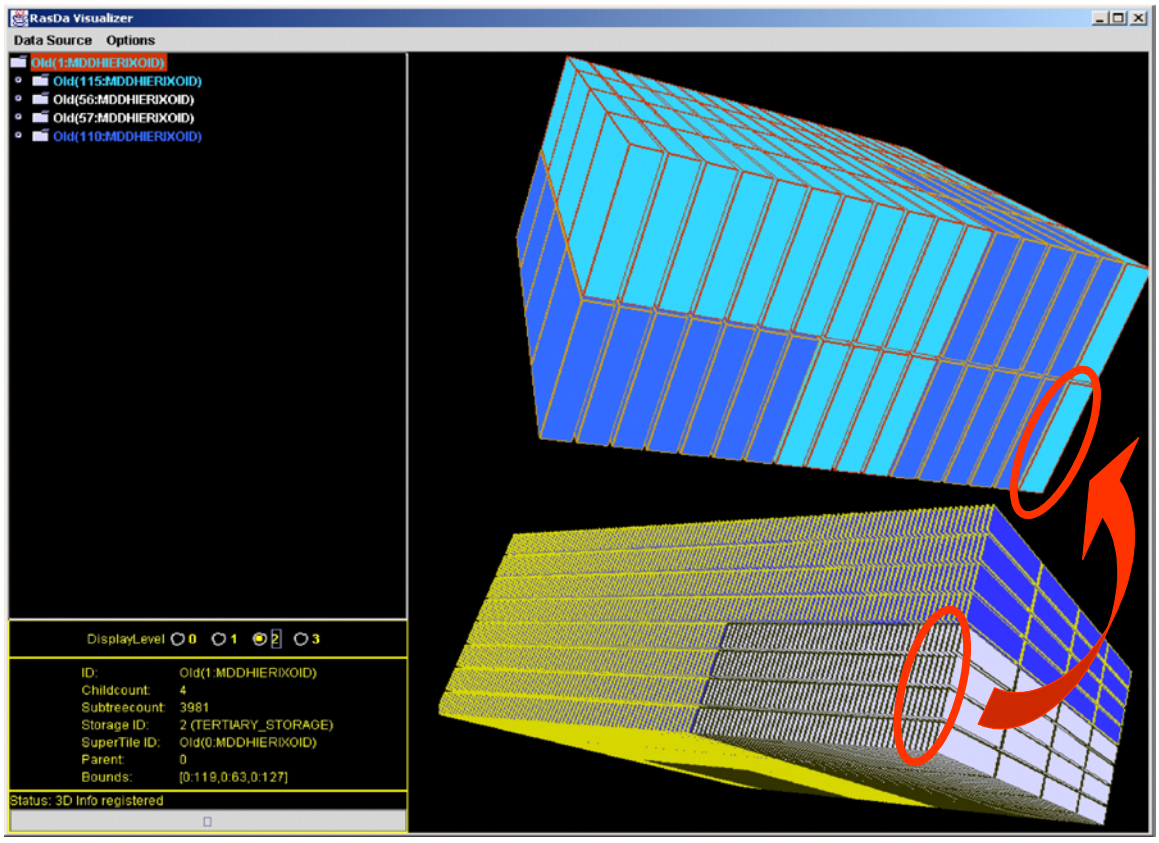


Abbildung 3.13: 3D-Objekt, dargestellt in Super-Kachel- und Kachel-Granularität.

Es ist zu erkennen, wie räumlich benachbarte Kacheln zu Super-Kacheln zusammengefasst wurden. Super-Kacheln, die sich auf TS-Medium ($\sigma \in \{\Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$) befinden, werden hellblau und Super-Kacheln, die sich im Cache-Bereich von RasDaMan ($\sigma \in \Theta_{SRC}$) befinden, dunkelblau dargestellt. Mit RasDa-Visualizer, können weiterhin für ein- bis dreidimensionale MDD, die Domänen der einzelnen Ebenen des R^+-Baumes angezeigt werden.

Schwächen des STAR-Konzepts

Wie bereits beschrieben, werden Knoten im R^+-Baum mit besonderen Eigenschaften zu Super-Knoten erhoben. Alle Datenobjekte (Kacheln), die sich in der Sub-Domäne dieses Super-Knotens befinden, werden zu einer Super-Kachel zusammengefasst. Die Auswahl der Super-Knoten innerhalb des R^+-Baumes ist von folgenden drei Kriterien abhängig:

1. Angestrebtes Datenvolumen der Super-Kachel Δ_σ
2. Vorhandenes Datenvolumen der Kacheln Δ_τ
3. Anzahl m der Einträge pro Indexknoten (R^+-Baum, mit $m \leq M$; Definition 2.13)

Die Höhe h_ϕ des Unterbaumes (Sub-Tree) eines Super-Knotens ϕ, kann somit wie folgt bestimmt werden:

$$\lfloor h_\phi \rfloor = \log_m \frac{\Delta_\sigma}{\Delta_\tau}$$

Zur Vereinfachung wurde angenommen, dass die Kachelgröße und die Anzahl m der Einträge pro Knoten im gesamten Baum gleich sind. Durch den logarithmischen Einfluss bei der Bestimmung der Super-Knoten, ist die angestrebte Super-Kachel-Größe Δ_σ in den meisten Fällen nicht exakt zu erreichen. Die tatsächliche Super-Kachel-Größe Δ^*_σ lässt sich ermitteln durch:

$$\Delta^*_\sigma = \Delta_\tau \, m^{h_\phi}$$

Abbildung 3.14 zeigt den prozentualen Anteil der tatsächlichen Super-Kachel-Größe gegenüber der angestrebten Super-Kachel-Größe bei Kachelgrößen zwischen 16 und 1024 KByte. Es ist zu erkennen, dass die angestrebten Super-Kachel-Größen zwischen 100 MByte und 500 MByte, jeweils nur für eine bestimmte Kachelgröße exakt erreicht wird. Eine ansteigende Wunschgröße einer Super-Kachel, erfordert eine zunehmende Kachelgröße, um diese genau zu erreichen. Greifen wir eine angestrebte Super-Kachel-Größe von 200 MByte heraus, so erhalten wir diese bei einer Kachelgröße von 128 KByte. Obwohl es in der Praxis nicht erforderlich ist, die angestrebte Super-Kachel-Größe exakt zu erreichen, ist die Anwendung von STAR problematisch. Mit STAR kann nur für jeweils einen sehr kleinen Bereich von Kachelgrößen eine akzeptable Annäherung an die gewünschte Super-Kachel-Größe ermöglicht werden. Um zum Beispiel eine Super-Kachel-Größe von 200 MByte zu mindestens 80 % zu erhalten, darf die Kachelgröße nur zwischen 103 KByte und 128 KByte variieren. Außerhalb dieses Bereiches sind, wie in Abbildung 3.14 ersichtlich, meist nur sehr geringe Annäherun-

gen an die gewünschte Super-Kachel-Größe möglich. Wie lässt sich der unterschiedlich starke Anstieg der Kurven, bis zum Erreichen des exakten Datenvolumens der Super-Kachel, gegenüber dem danach folgenden schwachen Anstieg erklären? Bis zum Erreichen der geforderten Super-Kachel-Größe, werden Kacheln von einem im R^+-Baum höher[28] liegenden Super-Knoten überdeckt. Eine steigende Kachelgröße wirkt sich auf alle überdeckten Kacheln aus und führt somit zu einem steileren Anstieg der Kurve. Wurde die gewünschte Super-Kachel-Größe mit dem Super-Knoten überschritten, wird ein Knoten eine Ebene tiefer zum neuen Super-Knoten erhoben. Steigende Kachelgrößen wirken sich nicht mehr so gravierend aus.

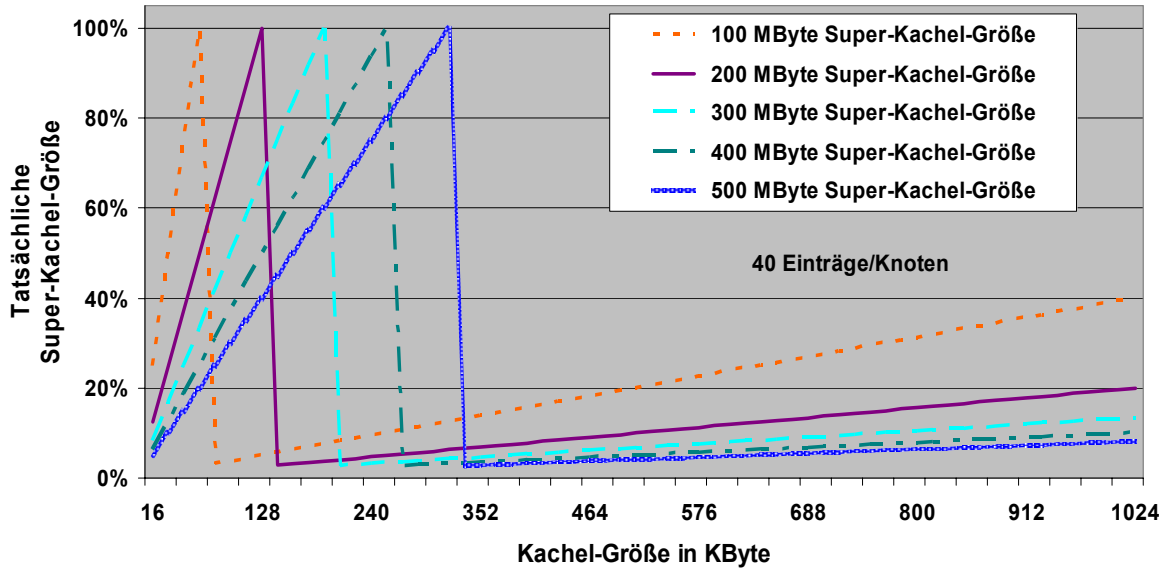


Abbildung 3.14: Tatsächlich erreichte Super-Kachel-Größe, abhängig von der Kachelgröße, bei unterschiedlich, angestrebten Super-Kachel-Größen (40 Einträgen pro Index-Knoten).

Neben der Abhängigkeit der tatsächlichen Super-Kachel-Größe von der verwendeten Kachelgröße, besteht mit der Anzahl der Einträge pro Indexknoten ein weiterer Einflussfaktor. In Abbildung 3.14 wurden 40 Einträge pro Indexknoten angenommen. Bei der in RasDaMan implementierten Variante des R^+-Baumes, liegt die durchschnittliche Anzahl der Einträge pro Indexknoten zwischen 20 und 60 Einträgen. Allerdings kann, im Gegensatz zum B-Baum, beim R^+-Baum kein Mindestfüllgrad der Indexknoten garantiert werden. Die Anzahl der Einträge eines Indexknoten ist vom Einfügevorgang der Objekte in den Index abhängig. Einen Einfluss auf den Füllgrad eines Indexknoten, hat auch die Dimensionalität des zu indexierenden Objektes. Für eine weitere Dimension, muss die entsprechende, zusätzliche Domäne im Indexknoten mit abgespeichert werden. Bei einer steigenden Anzahl von Einträgen pro Indexknoten verschieben sich die in Abbildung 3.14 dargestellten Kurven der unterschiedlichen, angestrebten Super-Kachel-Größen nach links. Die Form der Kurven verändert sich in gleicher Weise, wie bei verringerter Wunsch-Super-Kachel-Größe. Bei weniger Einträgen pro Indexknoten, findet eine entsprechende Verschiebung nach rechts statt.

[28] Diese Beschreibung bezieht sich auf die in dieser Arbeit gewählte Form der Darstellung eines R+-Baumes (Abbildung 3.12). Die höchste Ebene ist der Wurzelknoten und somit die Ebene 0. Analog dazu werden die Blattknoten als die tiefste Ebene bezeichnet.

Eine extreme Abhängigkeit der tatsächlichen Super-Kachel-Größe von der verwendeten Kachelgröße und der Anzahl der Einträge pro Indexkonten, ist natürlich nicht erwünschenswert. Um diesen Nachteil des entwickelten Algorithmus zu beseitigen wurde STAR dahingehend erweitert und verbessert.

3.5.3 Erweiterter Super-Tile-Algorithmus (eSTAR)

Um eine bessere Annäherung an die angestrebte Super-Kachel-Größe zu erreichen, wurde der erweiterte Super-Tile-Algorithmus (*extended STAR*, bzw. *eSTAR*) entwickelt. Die grundlegende Idee hinter eSTAR ist, nicht nur die von einem Super-Knoten überdeckten Kacheln zu einer Super-Kachel zu kombinieren, sondern die von mehreren Super-Knoten überdeckten Kacheln zu verschmelzen. Durch den logarithmischen Einfluss des R^+-Baumes beim Übergang in die nächste Hierarchieebene, kommt es sehr oft vor, dass die vom Vaterknoten überdeckten Kacheln, die geforderte Super-Kachel-Größe bereits überschreiten, allerdings der Kindknoten nur einen sehr kleinen Bereich an Kacheln überdeckt. Dadurch wird nur eine sehr geringe Annäherung an die gewünschte Super-Kachel-Größe erreicht (Abbildung 3.14). Betrachtet man die Baumstruktur, so fällt auf, dass neben dem ermittelten Super-Knoten, Geschwisterknoten (sibling node) vorhanden sind, welche die gleichen Kriterien erfüllen. Durch eine geschickte Kombination dieser Geschwisterknoten und den darunter enthaltenen Kacheln, kann eine gute Annäherung an die angestrebte Super-Kachel-Größe erreicht werden. Die Abbildung 3.15 zeigt dieses Prinzip des eSTAR.

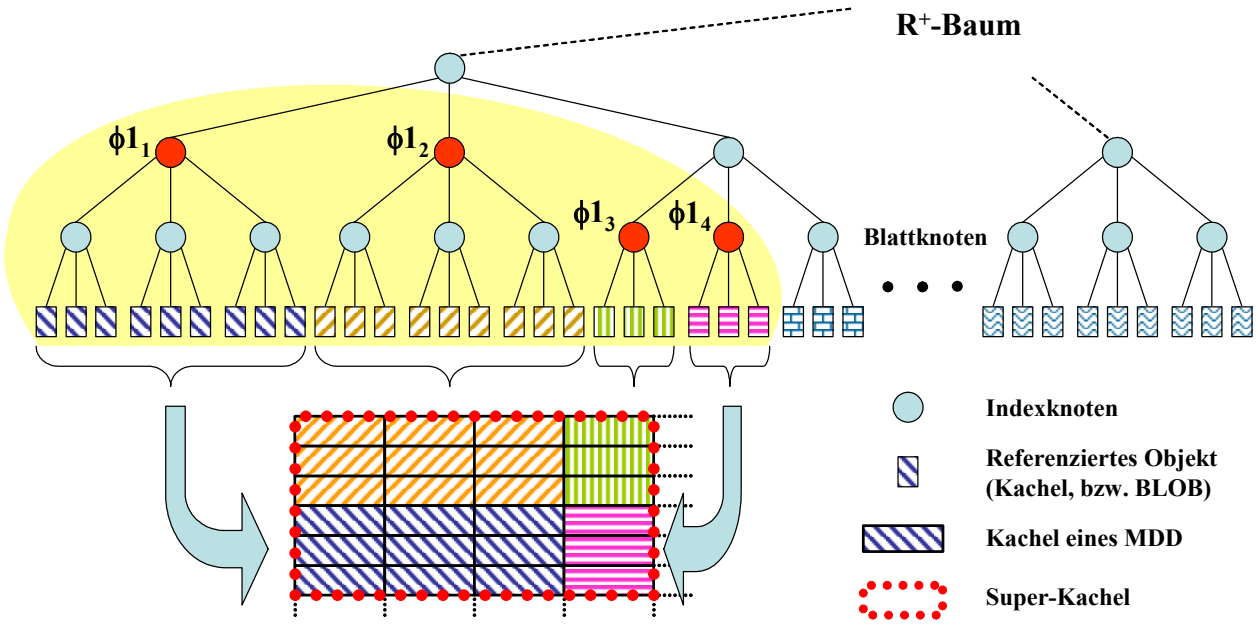


Abbildung 3.15: Prinzip des erweiterten Super-Tile-Algorithmus (eSTAR).

Um eine weitere Verbesserung zu erlangen, können in einem nächsten Schritt, Kindknoten von den Geschwistern dieser relevanten Super-Knoten im Baum herangezogen werden. Die-

ser Vorgang kann bis zu den Blattknoten wiederholt werden. Im vorliegenden Beispiel sind die beiden Blattknoten $\phi 1_3$ und $\phi 1_4$ auf diese Weise ermittelt worden. Somit wird in Abbildung 3.15 eine Super-Kachel aus den von vier Super-Knoten ($\phi 1_1$ bis $\phi 1_4$) referenzierten Kacheln gebildet. Je zwei dieser Super-Knoten liegen auf unterschiedlichen Baumebenen. Durch dieses Vorgehen, können weit bessere Annäherungen zur geforderten Super-Kachel-Größe erreicht werden, obwohl noch die gleichen Abhängigkeiten wie bei STAR bestehen. Die veränderte Situation ist in Abbildung 3.16 deutlich zu erkennen. Ebenso wie in Abbildung 3.14, wird auch hier der prozentuale Anteil der tatsächlichen Super-Kachel-Größe, gegenüber der angestrebten Super-Kachel-Größe, bei Kachelgrößen zwischen 16 KByte und 1024 KByte, gezeigt. Bei einem direkten Vergleich der beiden Abbildungen wird der große Unterschied deutlich.

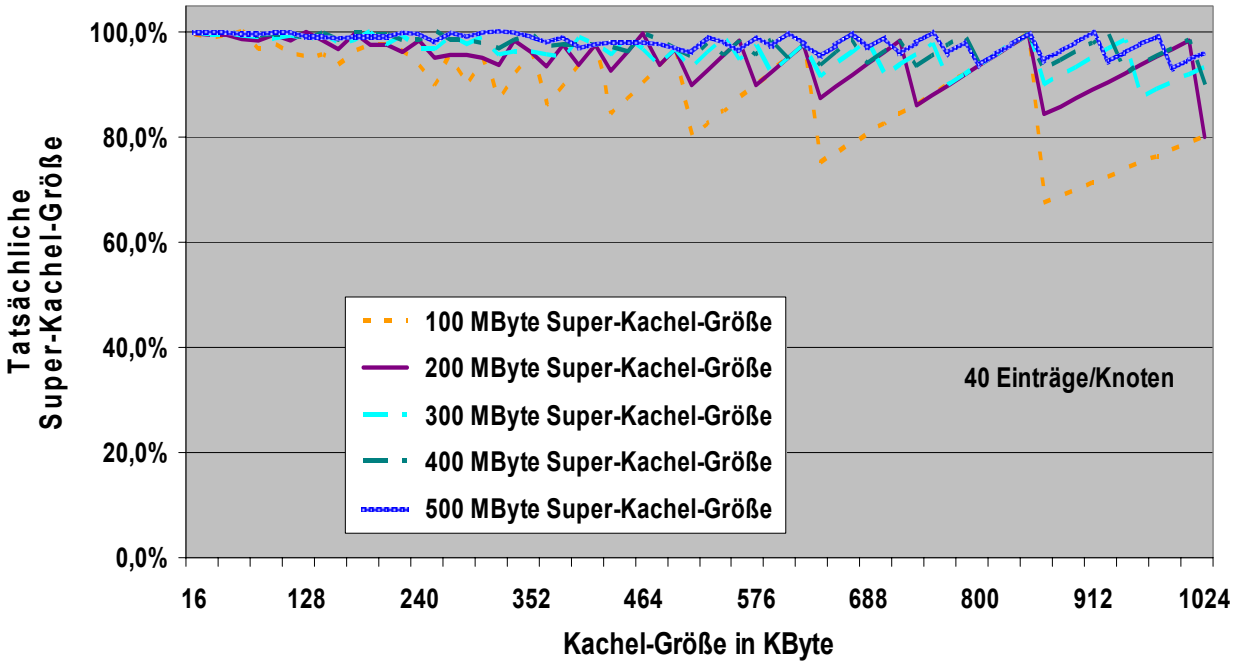


Abbildung 3.16: Tatsächlich erreichte Super-Kachel-Größe mit eSTAR, abhängig von der Kachelgröße, bei unterschiedliche, angestrebten Super-Kachel-Größen (40 Einträgen/Knoten).

Die mit steigender Kachelgröße ansteigende Zackenbewegung der tatsächlich erreichten Super-Kachel-Größe, lässt sich leicht erklären: Durch die Verwendung eines R^+-Baumes ist ein Blattknoten die geringste Granularität, die verwendet werden kann, um Super-Knoten zu kombinieren. Somit ist bei steigender Kachelgröße und gleich bleibender Anzahl der Einträge pro Indexknoten, das von einem Blattknoten indexierte Datenvolumen ansteigend. Das macht sich besonders bemerkbar, wenn kleine Super-Kachel-Größen (z.B. 100 MByte) gewünscht werden. Je größer die angestrebten Super-Kachel-Größen ausfallen, desto geringer ist der Einfluss, der von einem Blattknoten überdeckten Kacheln. Hier soll allerdings angemerkt werden, dass normalerweise bei steigender Kachelgröße, auch größere Super-Kacheln ge-

wählt werden und somit eine sehr gute Annäherung (≥ 80 %) der tatsächlichen Super-Kachel-Größe erreicht wird.

Nach der allgemeinen Beschreibung von eSTAR, mit den erwähnten Vorteilen gegenüber STAR, folgt eine formale Definition:

Definition 3.12: Erweiterter Super-Tile-Algorithmus (eSTAR)

Der erweiterte Super-Tile-Algorithmus weist folgende Eigenschaften auf:

1. Eine Menge von Super-Knoten ϕ eines R^+-Baumes, mit den Domänen $D_{\phi i}$, definieren das gesamte MDD α, wobei Super-Knoten auf unterschiedlichen Ebenen des R^+-Baumes liegen können:
   $sdom(\alpha) = \bigcup_{i=1}^{n} D_{\phi_i}$, wobei gilt: $D_{\phi_i} \cap D_{\phi_j} = \varnothing$ für $i, j \in \{1, ..., n\}; i \neq j$
2. Auf dem Pfad von den Blättern bis zur Wurzel darf nur ein Super-Knoten enthalten sein.
3. Die Domäne D_σ einer Super-Kachel, wird definiert durch die Domäne $D_{\phi i}$ eines oder mehrerer Super-Knoten ϕ des R^+-Baumes:
   $D_\sigma = \bigcup_{i=1}^{n} D_{\phi_i}$, wobei gilt: $D_{\phi_i} \cap D_{\phi_j} = \varnothing$ für $i, j \in \{1, ..., n\}; i \neq j$
   So ergibt die Menge aller Super-Kacheln σ das komplette MDD α (Definition 3.6).
4. Alle Datenobjekte (Kacheln), die von D_σ überdeckt werden, bilden eine Super-Kachel (Definition 3.4).

Wie bei STAR, hängt das tatsächliche Datenvolumen einer Super-Kachel bei eSTAR von den gleichen Faktoren ab. Diese sind die angestrebte Super-Kachel-Größe Δ_σ, die vorhandene Kachelgröße Δ_τ und die Anzahl m der Einträge pro Indexknoten. Allerdings ergibt sich aus der Eigenschaft Nr. 3 der Definition 3.12, eine weitaus bessere Annährung der tatsächlichen Super-Kachel-Größe Δ^*_σ an die angestrebte Super-Kachel-Größe Δ_σ. Das tatsächliche Datenvolumen lässt sich wie folgt berechnen:

$$\Delta^*_\sigma = \Delta_\tau \sum_{i=1}^{h_\phi} |\phi\text{sibling}_i| \, m^i, \text{ wobei gilt: } \lfloor h_\phi \rfloor = \log_m \frac{\Delta_\sigma}{\Delta_\tau}; \; |\phi\text{sibling}| \in \{1, ..., n\}$$

Zur Vereinfachung wurde angenommen, dass die Kachelgröße und die Anzahl m der Einträge pro Indexknoten im gesamten R^+-Baum gleich sind. Bei der Berechnung, entspricht der Faktor $|\phi\text{sibling}|$, der Anzahl der auf einer Baumebene kombinierten Super-Knoten (Geschwister, bzw. Siblings). Die Summenbildung berücksichtigt dabei die Tatsache, dass Super-Konten unterschiedlicher Baumebenen kombiniert werden. Die Anzahl der in diesen Super-Knoten enthaltenen Kacheln, wird mit der Kachelgröße multipliziert, um das tatsächliche Datenvolumen der Super-Kachel zu erhalten (es gilt: $\Delta^*_\sigma \leq \Delta_\sigma$).

Ermittlung der Super-Knoten

Nach dem Vergleich der beiden vorgestellten Verfahren, ist ersichtlich, dass eSTAR einen großen Vorteil gegenüber STAR aufweist. Allerdings ist die Bestimmung der Super-Kacheln weitaus komplexer. Bei STAR wird einfach ein Super-Knoten bestimmt und die überdeckten Kacheln zu einer Super-Kachel kombiniert. Bei eSTAR kommen in den meisten Fällen mehrere Super-Knoten in Frage, die zu einer Super-Kachel vereint werden können. Diese Super-Knoten können weiterhin noch auf unterschiedlichen Ebenen des R^+-Baumes liegen. Nun stellt sich die Frage, welche der möglichen Kandidaten zu Super-Knoten erhoben werden? Bei dieser Entscheidung hilft die Ausdehnung $extent(sdom(\tau))[i]$ der einzelnen Kacheln eines MDD α in jeder Dimension $i \in \{1, \ldots, d\}$. Wie in Abbildung 3.15 (unten) zu erkennen ist, wird versucht, das Seiten- oder Kantenverhältnis der ursprünglichen Ausdehnung der Kacheln bei den Super-Kacheln zu erhalten. Um dies zu erlangen, wurden die Kandidaten $\phi 1_1$ bis $\phi 1_4$ zu Super-Knoten. Durch das Aufrechterhalten der Kantenverhältnisse, wird die ursprüngliche Datenmodellierung beibehalten. Das bedeutet, dass zum Beispiel eine durch ausgerichtete Kachelung gewünschte Vorzugsdimension weiterhin gegeben ist (Abbildung 3.15, unten). Andererseits ist es möglich, bei regulärer Kachelung, eine Vorzugsdimension zu vermeiden, was auch erwünscht ist. Somit können Anfragen, entsprechend der erwarteten Anfragemuster, weiterhin gut beantwortet werden. Durch diese Möglichkeit der Erhaltung der Kantenverhältnisse der modellierten Kacheln, weist eSTAR einen weiteren Vorteil gegenüber dem STAR auf.

Zur Bestimmung der ursprünglichen Kantenverhältnisse der Ausdehnung der Kacheln, gibt es einige Besonderheiten zu beachten. Bei regulärerer und ausgerichteter Kachelung, ist nur die multidimensionale Ausdehnung einer Kachel zu bestimmen, da bis auf Randkacheln alle anderen Kacheln die gleiche Ausdehnung aufweisen (Definition 2.12, Abbildung 2.16). Für diese beiden Kachelungsarten ist eine optimale Ausdehnung einer Super-Kachel leicht zu ermitteln:

$$K_{[1:i]} = \frac{extent(sdom(\tau))[1]}{extent(sdom(\tau))[i]};\ \text{es gilt: } i \in \{2, ..., d\}$$

$$extent(sdom(\sigma))[1] = \sqrt[d]{\frac{\Delta_\sigma \prod_{i=2}^{d} K_{[1:i]}}{size(T)}}$$

$$extent(sdom(\sigma))[i] = \frac{extent(sdom(\sigma))[1]}{K_{[1:i]}};\ \text{es gilt: } i \in \{2, ..., d\}$$

Zunächst sind die Kantenverhältnisse $K_{[1:i]}$ einer Kachel τ in Bezug auf die Ausdehnung der ersten Dimension zu bestimmen ($i \in \{2, ..., d\}$). Anschließend kann unter Einbeziehung der gewünschten Super-Kachel-Größe Δ_σ, dem Basisdatentyp T und den Kantenverhältnissen $K_{[1:i]}$ die optimale Ausdehnung der Super-Kachel in der ersten Dimension ermittelt werden. Daraus lassen sich sehr einfach die optimalen Ausdehnungen der Super-Kachel σ der einzelnen Dimensionen berechnen. Die so berechnete optimale Domäne D einer Super-Kachel gilt

als Referenzdomäne. Bei der realen Bestimmung der Super-Kacheln sind die Domänen von Kacheln τ, bzw. von Super-Knoten ϕ zu beachten.

Einen Sonderfall nimmt die irreguläre Kachelung ein, da die Form der Kacheln unterschiedlich ist. Aufgrund der untergeordneten Praxisrelevanz der irregulären Kachelung wird hier eine einfache und funktionelle Lösung vorgestellt, die allerdings durchaus verbessert werden kann. Es wird eine mittlere Ausdehnung der Kacheln berechnet und als imaginäre Referenzkachel verwendet. Um die multidimensionale Ausdehnung der Referenzkachel zu bestimmen, kommt die Tatsache zum Tragen, dass Kachelgrenzen immer entlang einer kompletten Dimension verlaufen. Es reicht also aus, die jeweilige Ausdehnung $extent(sdom(\tau))[i]$ der Randkacheln jeder Dimension $i \in \{1, \ldots, d\}$ zu bestimmen. Die arbiträre Kachelung ist ein weiterer Sonderfall, bei dem dieser Weg nicht beschritten werden kann. Das liegt daran, dass die Form und somit die Ausdehnung der Kacheln innerhalb eines MDD starke Unterschiede aufweisen können. Weiterhin ist bei arbiträrer Kachelung nicht zwangsweise eine Vorzugsausrichtung der Kacheln vorhanden, bzw. falls vorhanden, sehr schwierig zu ermitteln. Auch hier besteht die Möglichkeit, eine mittlere Ausdehnung aller Kacheln eines MDD zu berechnen. Hierzu ist es im Gegensatz zur irregulären Kachelung allerdings erforderlich, die Domäne jeder einzelnen Kachel zu berücksichtigen, um eine imaginäre Referenzkachel zu bestimmen. Ein gangbarer Weg ist auch, möglichst Super-Knoten zu kombinieren, um das in Kapitel 3.5.2 beschriebene redundante Speichern von Kacheln innerhalb mehrerer Super-Kacheln zu reduzieren. Um das zu erreichen, muss überprüft werden, wie viele der involvierten Kacheln mehrfach von anderen Bereichen des R+-Baumes referenziert werden. Dies wird für jede Dimension getestet, die für eine Erweiterung relevant ist. Allerdings ist es meist nicht möglich, eine vollständige Redundanzfreiheit zu erreichen. Auch aus Gründen einer besseren Performance, wird keine vollständige Redundanzfreiheit gefordert, sondern nur eine Minimierung gewünscht. Aufgrund der im Projekt ESTEDI gezeigten nicht vorhandenen Praxisrelevanz der arbiträren Kachelung, wird in diesem Fall nicht auf das eSTAR-Konzept, sondern auf das weitaus einfachere STAR-Konzept (Kapitel 3.5.2) zurückgegriffen. Bei den weiteren Beschreibungen der Konzepte, liegt das Hauptaugenmerk auf regulärer und ausgerichteter Kachelung, da meist nur diese in der Praxis eingesetzt werden.

Die Ermittlung des ersten Super-Knotens ϕ_{Start} im R+-Baum eines MDD α erfolgt am Objektrand. Somit gilt $low(sdom(\phi))[i] = low(sdom(\alpha))[i]$ in jeder Dimension $i \in \{1, \ldots, d\}$. Wurde der erste Super-Knoten gefunden und aufgrund der noch nicht erreichten Super-Kachel-Größe erkannt, dass weitere Super-Knoten auf gleicher Ebene hinzugefügt werden können, so ist es abhängig von der multidimensionalen Ausdehnung dieses Super-Knotens, wie weiterhin verfahren wird. Entspricht das Verhältnis der Seitenkanten des Super-Knotens ϕ ungefähr dem einer Kachel τ, so wird versucht, diese Domäne in jeder Dimension entsprechend zu erweitern. Für die Erweiterung kommen nur Index-Knoten im R^+-Baum in Frage, dessen Domäne direkt an die zu erweiternde Domäne angrenzt. In Bezug auf den Startknoten ϕ_{Start} können nur Indexknoten zu Super-Knoten erhoben werden, für dessen untere Grenze seiner Domäne gilt: $low(sdom(\phi))[i] = high(sdom(\phi_{Start}))[i] + 1$. Das gleichförmige Erweitern eines Start-Super-Knoten ϕ_{Start} ist für zwei und drei Dimensionen in der Abbildung 3.17 dargestellt.

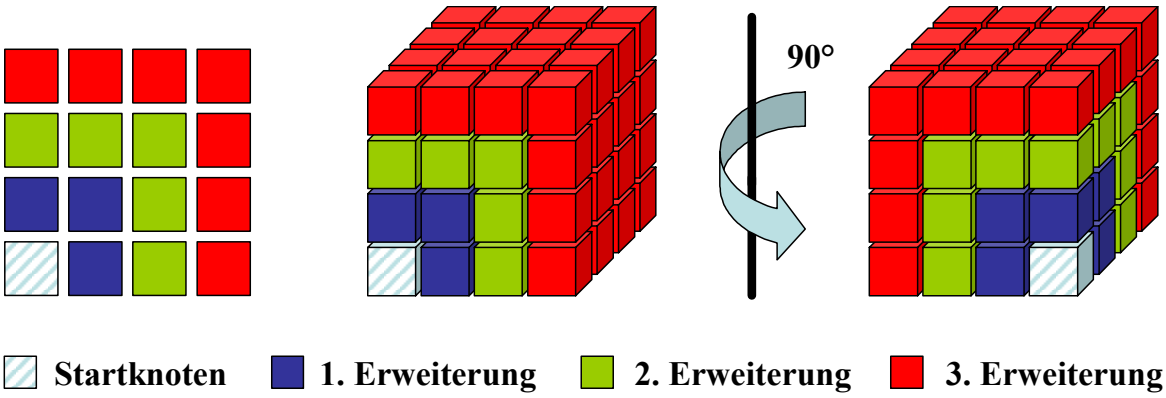


Abbildung 3.17: Mögliche gleichförmige Erweiterungen eines Super-Knotens (Startknoten).

Das ursprüngliche Seitenverhältnis des Super-Knotens (Startknoten), bleibt bei jeder Erweiterung erhalten. In diesem Beispiel wird der zwei- und dreidimensionale Startknoten in drei Stufen erweitert. Um dies im dreidimensionalen Fall besser zu erkennen, ist das Objekt zusätzlich um 90° gedreht dargestellt. Für jede erneute Erweiterung, steigt die Anzahl der benötigten Super-Knoten in Abhängigkeit von der Dimensionalität immer weiter an. Entsprechend der Anzahl n an Super-Knoten, die in einer Dimension dem ursprünglichen Super-Knoten (Startknoten) hinzugeführt werden, ergibt das in Abhängigkeit von der Dimensionalität d des Objektes, eine Anzahl N_ϕ an zusätzlich notwendigen Super-Knoten ϕ:

$$N_\phi = (n + 1)^d - 1$$

Diese Anzahl N_ϕ an Super-Knoten, müssen zusätzlich mit dem Startknoten kombiniert werden, um das Verhältnis der ursprünglichen Form beizubehalten. Um dies zu realisieren, werden Geschwister-Knoten des Startknoten im R^+-Baum gesucht, die den geforderten Bedingungen genügen und als Super-Knoten definiert.

Die Abhängigkeit der Anzahl N_ϕ an notwendigen Super-Knoten von der Dimensionalität d und der Anzahl n der Erweiterungen, ist in Tabelle 3.1 aufgeführt.

	N_ϕ bei 1-D	**N_ϕ bei 2-D**	**N_ϕ bei 3-D**	**N_ϕ bei 4-D**	**N_ϕ bei 5-D**
n = 1	1	3	7	15	31
n = 2	2	8	26	80	242
n = 3	3	15	63	255	1023
n = 4	4	24	124	624	3124

Tabelle 3.1: Abhängigkeit der zusätzlich benötigten Super-Knoten von der Dimensionalität und der Anzahl n an Erweiterungen.

Die Anzahl n der Erweiterung entspricht dabei der Anzahl an Super-Knoten, die in einer Dimension mit dem Startknoten kombiniert werden. Abhängig von der Kachelgröße und der Anzahl von Einträgen pro Indexknoten, wird es bei hohen Dimensionalitäten (> 6) immer unwahrscheinlicher, ein angestrebtes Datenvolumen für eine Super-Kachel durch gleichförmige Erweiterung hinsichtlich aller Dimensionen zu erreichen: Vor allem, weil die Domäne des Endergebnisses einer Erweiterung, einem multidimensionalen Hyperquader entsprechen muss.

Wie Abbildung 3.17 und Tabelle 3.1 zeigen, steigen bei höheren Dimensionen und bei zusätzlichen Erweiterungen die Anzahl der benötigten Super-Knoten. Da diese Super-Knoten eine Vielzahl von Kacheln überdecken, steigt das Datenvolumen der Super-Kachel in gleicher Weise. So ist es möglich, dass das gewünschte Datenvolumen einer Super-Kachel noch lange nicht erreicht wurde, aber bei einer zusätzlichen Erweiterung bereits überschritten wird.

Durch eine Aufweichung der harten Grenze der gewünschten Super-Kachel-Größe kann eine weitere Verbesserung erzielt werden. Das Aufweichen erlaubt in gewissen Bereichen ein Unter-, bzw. Überladen der Super-Kachel-Größe. Diese grundlegende Idee ist einleuchtend, da es nicht erforderlich ist, die gewünschte Super-Kachel-Größe immer exakt zu erreichen. Somit ist es sinnvoll, eine Überladung von ca. 10 % bis 30 % der Super-Kachel-Größe zuzulassen, bevor die tatsächlich erreichte Super-Kachelgröße weit unterschritten wird. Der Prozentsatz der möglichen Überladung ist als Tuningparameter in der Konfigurationsdatei *TertiaryStorage.conf* von HEAVEN anzugeben. Die grundlegende Idee, mit solchen „unscharfen" Grenzen umzugehen, entspringt ursprünglich der Fuzzy-Logik[29]. Ist trotz angemessener Überladung der angestrebten Super-Kachel-Größe, eine zusätzliche, in jeder Dimension gleichförmige Erweiterung nicht durchzuführen, so wird von dieser Vorgehensweise abgewichen. Die Erweiterung findet nicht mehr gleichförmig in jeder Dimension statt, sondern nur in ausgewählten Dimensionen. Es wird allerdings versucht, eine gewisse Annäherung zu erreichen. Die linke Seite der Abbildung 3.18 zeigt Möglichkeiten der ungleichförmigen Erweiterung für alle drei Dimensionen. Bereits an diesem einfachen Beispiel ist gut zu erkennen, dass sich die Datenvolumen der zusätzlichen Bereiche stark unterscheiden können. Für eine Erweiterung eines Startknoten ϕ_{Start} kommen nur direkt benachbarte Indexknoten ϕ in Frage, für die gilt: $low(sdom(\phi))[i] = high(sdom(\phi_{Start}))[i] + 1$.

Wird für den Startknoten ϕ_{Start} (schraffiert) eine Erweiterung in Dimension 2 angestrebt, so ist es erforderlich, zwei weitere Super-Knoten (rot) zu addieren. Bei Dimension 1 hingegen, müssen sechs Super-Knoten (grün) kombiniert werden. Dimension 3 begnügt sich mit drei Super-Knoten (nur Umrisse dargestellt). Zum Vergleich müssten bei einer gleichförmigen Erweiterung 18 Super-Knoten hinzugefügt werden. Natürlich ist es auch möglich, eine optimale Kombination aus unterschiedlichen Dimensionen zu bewerkstelligen. Ein Hauptkriterium hierbei ist die Bedingung, dass die Domäne der resultierenden Super-Kachel, einem mul-

[29] Die Fuzzy-Logik (ungenaue, verschwommene oder unscharfe Logik) ist eine Verallgemeinerung der zweiwertigen (boolschen) Logik. Die unscharfe Mengenlehre (Fuzzy-Set-Theorie) wurde bereits 1965 von L. A. Zadeh an der Universität von Berkley (USA) entwickelt.

tidimensionalen Hyperquader entspricht und nicht entartet ist. Als Entscheidungskriterien für eine Erweiterung, werden die genauere Annäherung an das angestrebte Datenvolumen der Super-Kachel und eine bessere Angleichung an die Seitenverhältnisse der Kacheln herangezogen. Die Seitenverhältnisse sind leicht über die Ausdehnung der Kachel $extent(sdom(\tau))$ zu ermitteln. Bei regulärer und ausgerichteter Kachelung reicht es aus, die Ausdehnung einer Kachel zu bestimmen. Die Ausdehnung aller Kacheln ist gleich, mit Ausnahme der oberen Randkacheln $high(sdom(\tau))[i] = high(sdom(\alpha)[i]$ in jeder Dimension $i \in \{1, \ldots, d\}$, die kleiner ausfallen können.

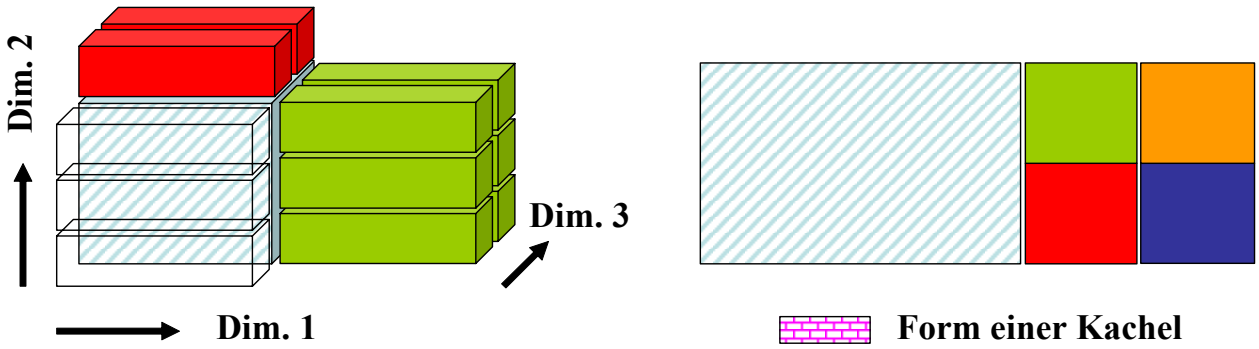


Abbildung 3.18: Links: Möglichkeiten der ungleichförmigen Erweiterung eines Startknotens; Rechts: Bilden einer wohlgeformten Super-Kachel.

Weiterhin können die bei der Datenmodellierung berücksichtigten Vorzugsrichtungen der Kacheln eine Entscheidungshilfe bringen. Betrachten wir in diesem Zusammenhang wieder unser Beispiel aus Abbildung 3.18 (links). Zum Erreichen der angestrebten Super-Kachel-Größe seien zusätzlich zum Startknoten ϕ_{Start} weitere sechs Super-Knoten ϕ erforderlich. So ergeben sich vier Möglichkeiten, um diese Forderung zu erfüllen. Betrachten wir zunächst Erweiterungen in den einzelnen Dimensionen. Durch eine einfache Erweiterung in Dimension 1 wird das Ziel erreicht. Auch durch eine zweifache, bzw. dreifache Erweiterung in Dimension 3, bzw. Dimension 2, wird das gleiche Ergebnis erzielt. Ebenso eine kombinierte Erweiterung in Dimension 2 und 3, liefert das gewünschte Resultat. Wir entscheiden uns für die Erweiterung in Richtung der ersten Dimension, da das Seitenverhältnis der ursprünglichen Kachel, zum Beispiel eine längere Ausdehnung in Dimension 1 aufweist. Somit können die für die Datenmodellierung herangezogenen, typischen Anfragen effizienter durch die Aufrechterhaltung von Vorzugsdimensionen, zum Beispiel bei Zeitreihen beantwortet werden.

Bei einem R^+-Baum kann grundsätzlich nicht garantiert werden, dass die einzelnen Hierarchieebenen des Baumes diese grundlegende Struktur der Datenelemente beibehalten. Vielmehr werden die Domänen der einzelnen Indexknoten, durch das Einfügen von Datenelementen in den Baum und somit durch den Split-Algorithmus der Knoten, bei einem Knotenüberlauf bestimmt. Mit eSTAR kann durch eine geeignete Kombination von Knoten, bzw. Super-Knoten, dieses ursprüngliche Seitenverhältnis der Kacheln, in einem gewissen Umfang nach-

gebildet werden. Entspricht das Verhältnis der Seitenkanten der Domäne des Startknotens ϕ_{Start} nicht dem einer Kachel, so werden Knoten im R^+-Baum gesucht, die zu einer wohlgeformten Super-Kachel führen. Ziel dabei ist es, einen Startknoten mit geeigneten Geschwisterknoten, bzw. dessen Kindknoten (Neffe), zu kombinieren. Diese geschickte Kombination mehrerer Super-Knoten zeigt die rechte Seite der Abbildung 3.18. Es ist zu erkennen, dass der Startknoten (schraffiert dargestellt) in Bezug auf die Seitenverhältnisse, nicht der ursprünglichen Form einer Kachel entspricht. Durch das Hinzufügen weiterer Super-Knoten, wird das gewünschte Seitenverhältnis wieder erreicht. Da eine Erweiterung durch einen Geschwisterknoten auf gleicher Indexebene des Startknoten nicht zu dem gewünschten Ergebnis führt, werden vier Neffen-Knoten der darunter liegenden Indexebene herangezogen. Auch in Abbildung 3.15 ist die Kombination von Super-Knoten unterschiedlicher Indexebenen dargestellt.

Wie bereits erwähnt, ist für einen effizienten Datenzugriff eine Indexstruktur in Kombination mit einem geeigneten Clustering die Grundvoraussetzung. Der in RasDaMan implementierte R^+-Baum wurde um das Konzept der Super-Knoten erweitert. Das bedeutet, dass beim Durchsuchen der Indexstruktur, der Speicherort (SR, SRC, SHC, T_{on}, T_{near} oder T_{off}) der gesuchten Kacheln, bzw. Super-Kacheln bereits bekannt ist. Eine Abbildung von Anfragen auf diese neue Struktur wurde vorgenommen. Dabei werden alle in RasDaMan vorhandenen Operationen ohne Einschränkung unterstützt (Kapitel 2.5.5). Durch die Aufrechterhaltung der räumlichen Nachbarschaft von Objekten können Bereichsanfragen effizient durchgeführt werden. In RasDaMan wird für jedes MDD einer Kollektion eine eigenständige Baumstruktur aufgebaut. Verwaltet wird die Indexstruktur in zwei Tabellen (RAS_HIERIX und RAS_HIERIXDYN), die im konventionellen DBMS von RasDaMan abgelegt sind. Durch den implementierten eSTAR wird eine horizontale Datenpartitionierung vorgenommen, da bestimmte Kacheln (BLOBs) aus der BLOB-Tabelle (RAS_TILES) zu Super-Kacheln kombiniert und auf TS-Medien ausgelagert werden.

3.5.4 Automatische Anpassung der Super-Kachel-Größe

Bisher wurde besprochen, wie sich einzelne Kacheln geschickt zu Super-Kacheln zusammenfassen lassen, um eine für die jeweilige Speicherhierarchie angepasste Zugriffsgranularität zu erreichen. Die Granularität für Festplattenzugriffe ist durch die gewählte Kachelung gegeben und liegt zwischen 16 KByte und 1 MByte. Die Wahl einer geeigneten Datengranularität für das Retrieval von Daten von einem TS-Medium, erfordert einige Überlegungen. Wird eine zu kleine Datengranularität gewählt, so sind zu viele E/A-Aufträge notwendig: Teure Positionierungsoperationen beim Zugriff auf eine Datei treten zu häufig auf. Wenn allerdings die Datengranularität zu hoch gewählt wird, ist der Transferüberschuss zu groß, da neben den Nutzdaten zu viele irrelevante Daten, die nicht Gegenstand der Anfrage sind, übertragen werden müssen. In diesem Kapitel wird ein analytisches Modell entwickelt, das einen Kompromiss darstellt und eine geeignete Datengranularität für TS-Zugriffe ermittelt. Wie groß sollte nun die Zugriffsgranularität für TS-Zugriffe ausfallen? Dies hängt von unterschiedlichen Einflussfaktoren ab:

1. Mittlere Positionierungszeit t_p des eingesetzten TS-Systems
2. Transferrate Γ des verwendeten Bandlaufsystems
3. Durchschnittliche Größe einer Kachel Δ_τ
4. Durchschnittliche Ausdehnung *extent*(*sdom*(Q)) der zu erwartenden Anfragen Q
5. Zu erwartender Verschnitt υ_σ bei Anfragen

Diese Einflussfaktoren erstrecken sich von medienspezifischen Gegebenheiten (1, 2), bis hin zu objektabhängigen, bzw. anwendungsabhängigen Eigenschaften (3, 4, 5). HEAVEN bietet grundsätzlich zwei Möglichkeiten, das Datenvolumen einer Super-Kachel festzulegen. Einerseits kann durch einen Administrator von HEAVEN, durch das explizite Eintragen einer Größe in die Konfigurationsdatei *TertiaryStorage.conf*, die Zugriffsgranularität vordefiniert werden. Dies erfordert eine gute Fachkenntnis des Administrators über oben genannte Einflussfaktoren, um eine optimale Konfiguration zu erreichen. Die Abhängigkeit der Super-Kachel-Größe von objektspezifischen Einflussfaktoren macht dieses Verfahren für die Praxis eher untauglich, da manuelle Anpassungen vorzunehmen sind. Die manuelle Eingabe kann allerdings bei vorhandenen Erfahrungswerten als Grundeinstellung dienen. Eine weitere Alternative bietet die automatische Berechnung einer optimalen Größe einer Super-Kachel. Um dies zu bewerkstelligen, ist es notwendig, die erwähnten Abhängigkeiten in Relation zueinander zu setzen. Zu erwähnen ist, dass alle Angaben für den Administrator als optional gelten und nur entsprechende Faktoren in die Berechnung mit einfließen. Folgende Formel wird zur Bestimmung der Super-Kachel-Größe Δ_σ herangezogen:

$$\Delta_\sigma = \left(c_1 \left(\frac{t_p}{\Psi} - t_p \right) \Gamma + c_2 \, \Delta_\sigma^{min} \log\left(\frac{\Delta_\tau}{\Delta_\delta} \right) \right) \left(1 - c_3 \, \upsilon_\sigma \right)$$

wobei gilt: $0 \le c_1 \le n$; $0 < \Psi \le 1$; $c_2 \ge 0$, $\upsilon_\sigma \le 1$; $0 \le c_3 < 1$

Das Datenvolumen Δ_σ der Super-Kachel, hängt überwiegend ab von den TS-Systemeigenschaften der mittleren Positionierzeit t_p und der Transferrate Γ. Diese Eigenschaften sind von einem Systemadministrator leicht durch die entsprechenden Datenblätter zu ermitteln und anzugeben. Um die bestehende Dominanz der Positionierungszeit gegenüber der Transferdauer zu minimieren wird die Konstante Ψ benötigt. Ψ ist der gewünschte Anteil der mittleren Positionierzeit t_p, gegenüber der Gesamtzeit t_g aus mittlerer Positionierzeit und Transferdauer (Abbildung 3.8, Seite 88). Mit Werten von 0,7 bis 0,9 wurden gute Ergebnisse für die Kachel-Größe erzielt. Der Wert 0,8 ist als Grundeinstellung für Ψ vorgesehen. Mit der Konstanten c_1 gehen Erfahrungswerte in die Berechnung der Super-Kachel-Größe mit ein. Dabei spielen Faktoren, wie die Güte des verwendeten Clustering und der erreichte Kompressionsfaktor der Daten auf Magnetband eine Rolle. Ein gutes multidimensionales Clustering der Daten auf dem TS-Medium, wird durch die Verkleinerung der Konstanten c_1 ausgedrückt. Clustering reduziert auch bei kleineren Zugriffsgranularitäten kostspielige Operationen zur Positionierung, da zum Beispiel die benötigten Daten direkt nacheinander auf dem Magnetband gespeichert vorliegen. Das bedeutet, dass mehrere wahlfreie Zugriffe mit teueren Positionierungsoperationen, zu einem sequentiellen Zugriff zusammengefasst werden. Somit domi-

niert die Positionierungszeit nicht mehr so stark. Im Gegensatz dazu, ist bei einer besseren Kompressionsrate der Daten auf dem Magnetband, eine Erhöhung der Konstante c_1 angebracht. Dies ist durch die Tatsache begründet, dass bei einer höheren Kompressionsrate, höhere Transferraten erreicht werden, da höhere native Datenvolumen in gleicher Zeit vom Magnetband gelesen werden können. Als Grundeinstellung für c_1 wird der Wert 0,7 verwendet.

Weiterhin berechnet sich die Super-Kachel-Größe Δ_σ aus dem minimalen Volumen einer Super-Kachel Δ_σ^{min} und den objektspezifischen Eigenschaften: Wie die mittlere Kachelgröße Δ_τ und die Größe der verwendeten Datenbankseite Δ_δ. Der logarithmische Verlauf verhindert eine zu schwache, bzw. zu starke Einflussnahme bei zu kleinem, bzw. zu großem $\Delta_\tau/\Delta_\delta$-Verhältnis. Als Voreinstellung für das minimale Volumen einer Super-Kachel Δ_σ^{min} wurde 20 MByte gewählt. Die für die Formel benötigten objektspezifischen Eigenschaften, wie die mittlere Kachelgröße Δ_τ und die Größe der verwendeten Datenbankseite Δ_δ werden automatisch ermittelt. In der verwendeten Installation, wird eine Datenbankseite auf 8 KByte festgesetzt, wie es die Grundeinstellung von RasDaMan und Oracle in der Version 8.1.7 vorschlägt. Die durchschnittliche Ausdehnung *extent*(*sdom*(Q)) der zu erwartenden Anfragen $Q = \{q_1, q_2, \ldots, q_n\}$, fließt indirekt in die Kachelgröße mit ein: Es wird die Kachelung entsprechend der typischen Anfragemuster gewählt. Durch die Konstante c_2 kann der Einfluss der objektspezifischen Eigenschaften, gegenüber den medienspezifischen Eigenschaften, verändert werden. Momentan ist die Konstante c_2 auf den Wert Eins gesetzt. Zukünftig soll es ermöglicht werden, anhand heuristischer Erhebungen eines Systemkataloges, die mittlere Ausdehnung von bereits gestellten Bereichsanfragen einfließen zu lassen. Ein solcher Systemkatalog für RasDaMan wurde in einer, im Rahmen dieser Dissertation betreuten Diplomarbeit erstellt [Albe04].

Durch die Nutzung eines Systemkataloges ist es ebenfalls möglich, aufgrund von Heuristiken, den eventuell aufgetretenen Verschnitt υ_σ (Definition 3.8) von Multianfragen ähnlicher Objekte, in die Berechnung der Super-Kachel-Größe mit aufzunehmen. Obwohl ein größeres Datenvolumen für TS-Zugriffe erwünscht ist, sollte allerdings der Verschnitt nicht zu hoch ausfallen, um unnötiges Laden von Daten zu vermeiden. Der durchschnittlich aufgetretene Verschnitt von Anfragen auf einzelne MDD einer Kollektion kann ermittelt und in einem Systemkatalog abgelegt werden. Wird diese Kollektion durch neue MDDs erweitert, die zum Beispiel wöchentlich hinzugefügt werden, so sind ähnliche Anfragemuster auf diese neuen MDD zu erwarten: Der in der Vergangenheit aufgetretene Verschnitt kann somit berücksichtigt werden. Bei steigendem Verschnitt reduziert sich die Super-Kachel-Größe. Die Konstante c_3 (Grundeinstellung 0,5) variiert dabei die Stärke der Einflussnahme des Verschnittes auf die Bestimmung des Datenvolumens einer Super-Kachel.

Abbildung 3.19 zeigt die bei der entwickelten Formel mögliche Einflussnahme unterschiedlicher Faktoren auf die Super-Kachel-Größe Δ_σ (in MByte). Bei beiden oberen Darstellungen wurde die Einflussnahme des zu erwartenden Verschnittes υ_σ aufgrund eines einfacheren Verständnisses nicht berücksichtigt. In der linken, oberen Darstellung wird die Transferrate Γ

(in MByte/s) und die mittlere Kachelgröße Δ_τ (in KByte) variiert. Diese beiden Faktoren stellen in der Praxis die am häufigsten auftretenden Veränderungen dar. Zum einen wird entsprechend der gewünschten Datenmodellierung in RasDaMan eine unterschiedliche Kachelgröße verwendet. Andererseits wirkt sich bei unterschiedlichen Bandtechnologien die Transferrate entscheidend gegenüber der mittleren Positionierungszeit aus. Es ist gut zu erkennen, dass sowohl mit steigender Kachelgröße, als auch mit steigender Transferrate, das Datenvolumen der Super-Kachel zunimmt.

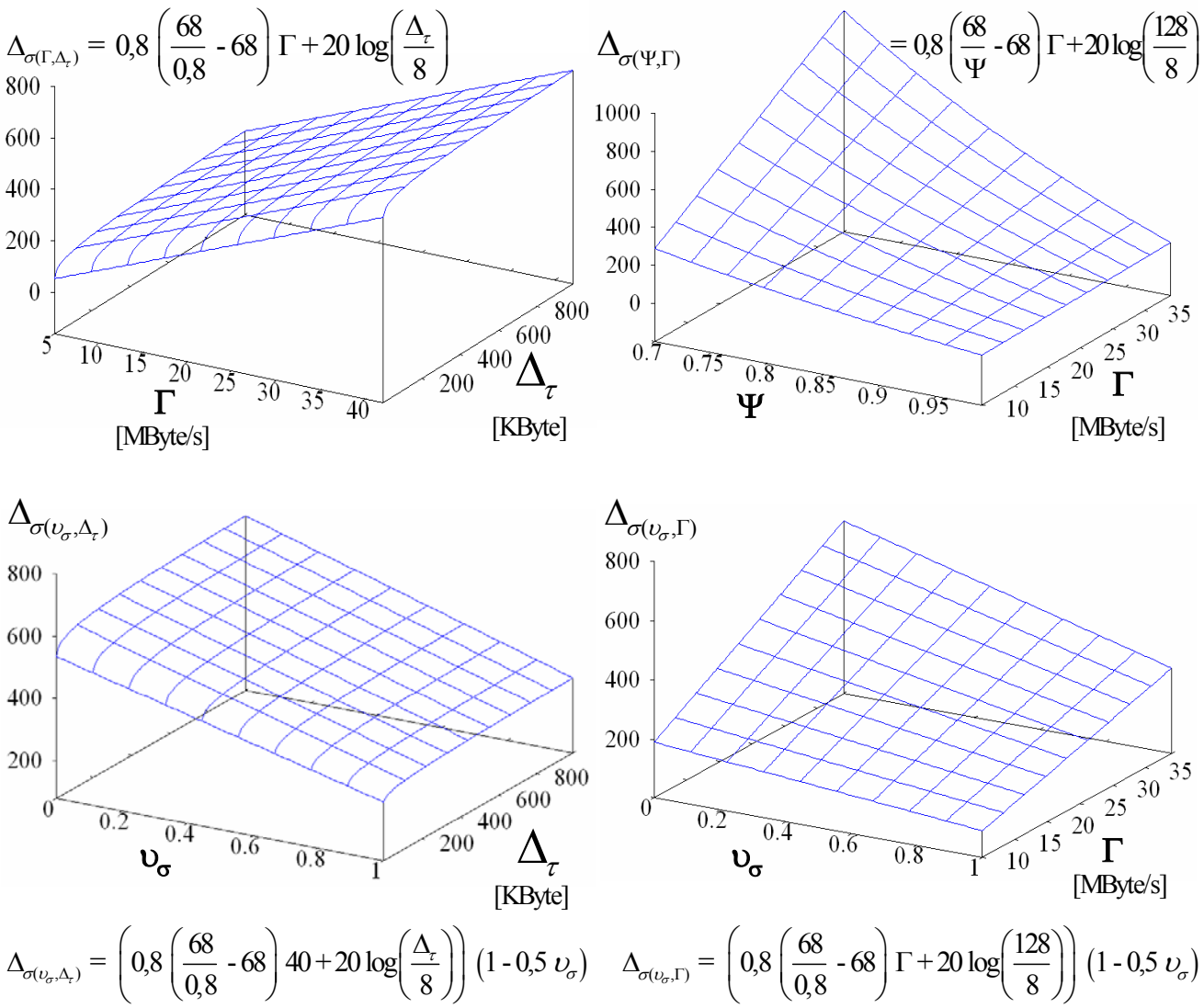


Abbildung 3.19: Einflussnahme unterschiedlicher Faktoren auf die Super-Kachel-Größe, unter Berücksichtigung von Standard- und Erfahrungswerten.

Die rechte, obere Seite der Abbildung 3.19, zeigt die Abhängigkeit der Super-Kachel-Größe Δ_σ (in MByte) von der Konstante Ψ und der Transferrate Γ (in MByte/s). Um den Einfluss der Positionierungszeit zu reduzieren, muss die Super-Kachel-Größe erhöht werden. Das bedeutet, die Konstante Ψ ist zu verkleinern. Eine steigende Transferrate, bewirkt ebenfalls eine steigende Super-Kachel-Größe, da sonst die Positionierungszeit eine noch stärkere Dominanz aufweisen würde.

Im linken, unteren Bereich der Abbildung 3.19, wird die Entwicklung der Super-Kachel-Größe Δ_σ (in MByte) in Abhängigkeit von der Kachelgröße Δ_τ (in KByte) und dem in der Vergangenheit bei ähnlichen Objekten aufgetretenen Verschnitt υ_σ gezeigt. Es ist zu erkennen, dass bei steigendem Verschnitt die Super-Kachel-Größe reduziert wird. Je nach Kachelgröße wirkt sich die Reduzierung mehr oder weniger stark aus.

Ein ähnliches Verhalten zeigt sich bei der Abhängigkeit des Datenvolumens einer Super-Kachel Δ_σ (in MByte) vom Verschnitt υ_σ und von der Transferrate Γ (in MByte/s) eines TS-Systems (Abbildung 3.19, rechts unten). Die Super-Kachel-Größe wird ebenfalls bei steigendem Verschnitt reduziert, was auch bei steigender Transferrate bemerkbar ist.

Positionierungsverhalten von TS-Systemen

Durch die Wahl der mittleren Positionierungszeit der TS-Systeme für die Ermittlung der Super-Kachel-Größe, entsteht eine Unabhängigkeit von unterschiedlichen TS-Technologien. So sind zum Beispiel die tatsächlichen Positionierungszeiten von Magnetbändern mit Serpentinen-, bzw. Schrägspuraufzeichnung sehr unterschiedlich (Abbildung 3.20).

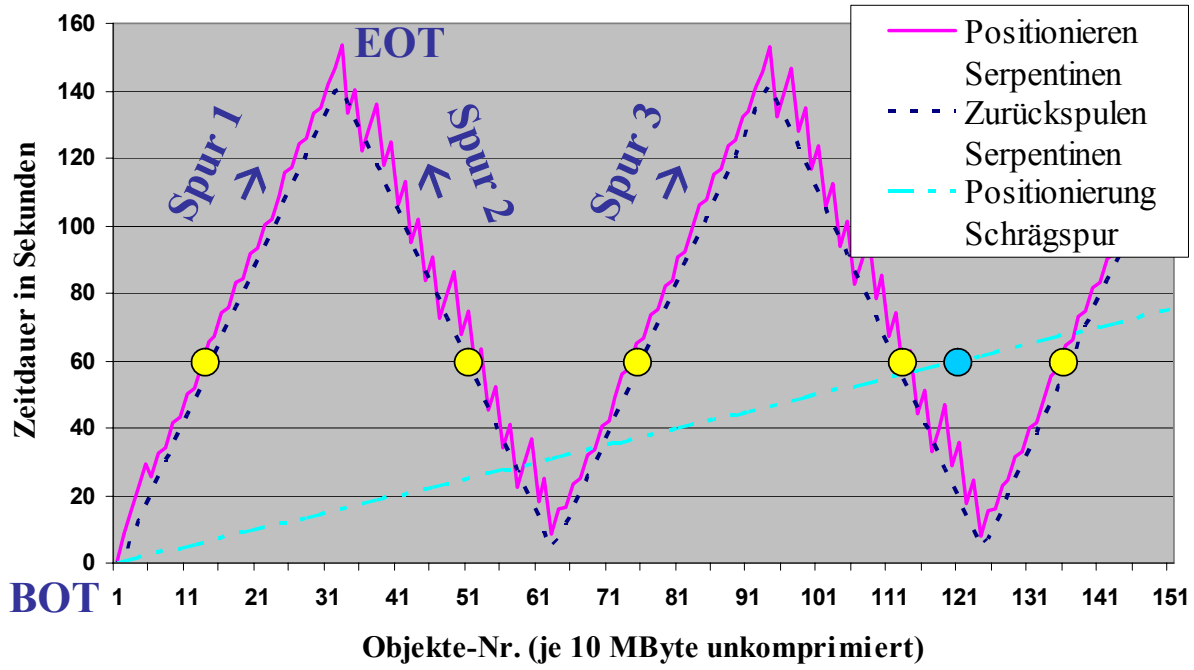


Abbildung 3.20: Positionierungsverhalten bei Serpentinen-, bzw. Schrägspuraufzeichnung.

Die Positionierungszeit bei Magnetbändern mit Schrägspuraufzeichnung weist einen linearen Verlauf über das gesamte, zur Verfügung stehende Datenvolumen auf. Im Gegensatz dazu erreichen Magnetbänder mit Serpentinenaufzeichnung nur für jeweils kurze Bereiche einen linearen Verlauf. Die dreieckige Kurvenform wird durch die Serpentine und die jeweilige Laufrichtung des Magnetbandes bei der Positionierung erreicht. Die Spur 1 wird in Vorwärtsrichtung positioniert, von Bandanfang (BOT, *Begin of Tape*) bis zum Bandende (EOT, *End of*

Tape). Danach wird die Spur gewechselt und das Magnetband rückwärts durchsucht (Spur 2). In diesem Beispiel sind nur zwei komplette Serpentinen dargestellt. Ermittelt wurde dieser Verlauf mit einem DLT 4000 Magnetband. Der gezackte Verlauf der Positionierungskurve gegenüber dem Rückspulverhalten, ist durch unterschiedliche Positionierungsgeschwindigkeiten zu erklären. Die stärkere Zackung bei „Rückwärts"-Spuren (z.B. Spur 2) wird durch das Überspulen des zu lesenden Datenobjektes und der erforderlichen Richtungsumkehr beim Lesen verursacht. Der gravierende Unterschied zwischen beiden Medien besteht darin, dass bei der Serpentinentechnik mehrere Datenobjekte (je Datenspur ein Objekt) den gleichen zeitlichen Aufwand für die Positionierung benötigen (kleine Kreise in Abbildung 3.20). Bei der Schrägspuraufzeichnung haben durch den linearen Verlauf alle Datenobjekte unterschiedliche Positionierungszeiten.

3.5.5 Erreichtes Optimierungspotential

Das entwickelte Super-Kachel-Konzept, ermöglicht durch eine dem jeweiligen Speicherort angepasste Datengranularität und dem realisierten Intra-Super-Tile-Clustering, eine Reduzierung der TS-Zugriffe und eine Minimierung von kostspieligen Operationen zur Positionierung. Die speziell für das Datenretrieval von TS-Medien angepasste Datengranularität spart kostenintensive Zugriffe ein. Es wird dabei die große Anzahl von Zugriffen auf kleine Objekte (Kacheln) ersetzt durch eine geringe Anzahl von Zugriffen auf größenoptimierte Objekte (Super-Kachel). Dadurch wird die Dominanz der Positionierungszeit gegenüber der Übertragungszeit abgeschwächt (Abbildung 3.8). Entfallen Zugriffe auf kleine Datenobjekte, reduzieren sich selbstverständlich auch die kostenintensiven Operationen zur Positionierung dieser Objekte. Die Größe eines angepassten Datenvolumens einer Super-Kachel wird, abhängig von medien- und objektspezifischen Eigenschaften, automatisch ohne Nutzerinteraktion ermittelt.

Weiterhin wird durch multidimensionales Clustering versucht, die räumliche Nähe der Daten auch auf dem Magnetband beizubehalten. So kann vermieden werden, dass einzelne Kacheln auf Magnetband „zufällig" verteilt vorliegen. Gerade bei Bereichsanfragen werden räumlich benachbarte Daten angefragt, die durch das Intra-Super-Tile-Clustering zu einer Super-Kachel kombiniert wurden. Das führt zu einer Minimierung von aufwändigen Operationen zur Positionierung. Bei Bereichsanfragen, ist nicht nur die Kombination von räumlich benachbarten Kacheln zu Super-Kacheln wichtig, sondern auch das Erhalten der räumlichen Nachbarschaft zwischen den Super-Kacheln auf TS-Medien. Dieses Inter-Super-Tile-Clustering wird im folgenden Kapitel behandelt.

Eine wichtige Rolle spielt auch eine, für zukünftige Anfragemuster optimierte Datenmodellierung. Dies kann durch die Wahl einer geeigneten Kachelungsstrategie erreicht werden. Durch den entwickelten eSTAR, wird die bei der Datenmodellierung gewählte Kachelung bei Super-Kacheln aufrechterhalten, um auch beim Retrieval von TS-Medien eine optimale Performance zu gewährleisten. So werden Vorzugsdimensionen, die bei der Datenmodellierung entstehen, beibehalten und entsprechend auf TS-Medien propagiert. Vergleichen wir *HEAVEN* mit den in Kapitel 2.4 untersuchten Systemen, so bietet das entwickelte Super-Kachel-Konzept einen

Mehrwert hinsichtlich einer dem jeweiligen Speicherort angepassten Datengranularität und dem realisierten Intra-Super-Tile-Clustering (Kapitel 5).

3.6 Export von multidimensionalen Daten

Nach der Ermittlung einer geeigneten Datengranularität (Super-Kachel) für das Speichern von Objekten auf tertiären Speichermedien und der Realisierung von Intra-Super-Tile-Clustering mittels eSTAR, folgen in diesem Kapitel Möglichkeiten, Datenobjekte auf TS-Medien zu exportieren, um diese effizient anfragen zu können. Dabei wird beim Datenexport das Inter-Super-Tile-Clustering realisiert, das in Kombination mit einer geeigneten *Scheduling*-Strategie (Kapitel 3.7.3) eine gute Performance beim Datenretrieval verspricht. Bevor näher auf den Datenexport eingegangen wird, werden Bezeichner eingeführt, die zur Beschreibung tertiärer Speichermedien und Speichersysteme dienen:

Definition 3.13: Tertiäres Speichermedium (Volume) v, Menge aller Speichermedien V

Ein Volume v ist ein einzelnes tertiäres Speichermedium, wie zum Beispiel ein Magnetband aus der Menge V aller von HEAVEN verwalteten Medien. V_{on} ist die Menge aller Online-Medien ($V_{on} \subset V$), d.h. die Menge der Medien, die zum aktuellen Zeitpunkt in einem Laufwerk eingelegt und zugreifbar sind (Speicherebene T_{on}). Die Menge V_{near} beinhaltet alle Nearline-Medien ($V_{near} \subset V$), also die Menge aller Medien, die sich in einer automatisierten Medien-Bibliothek, allerdings nicht in einem Laufwerk befinden (Speicherebene T_{near}). Auf die Menge V_{off} der Offline-Medien ($V_{off} \subset V$) kann nicht ohne manuelle Nutzerinteraktion zugegriffen werden (Speicherebene T_{off}). Zu jedem Zeitpunkt gilt $v \in \{V_{on} \cup V_{near} \cup V_{off}\}$ und $V_{on} \cap V_{near} \cap V_{off} = \varnothing$.

Definition 3.14: TS-Medium v(τ) eines Objektes, Objektmenge $\Theta_\tau(v)$ eines TS-Mediums

Wir bezeichnen mit v(τ) das Medium $v \in V$, auf dem das Speicherobjekt (Kachel) $\tau \in \Theta_\tau$ (innerhalb einer Super-Kachel σ) gespeichert ist. $\Theta_\tau(v) \subset \Theta_\tau$ ist die Menge aller Speicherobjekte τ, die auf dem Medium v gespeichert sind. Analog dazu ist $\Theta_\sigma(v) \subset \Theta_\sigma$ die Menge aller Super-Kacheln σ, die auf dem Medium v gespeichert sind. Werden alle Super-Kacheln σ auf TS-Medien gespeichert und existieren keine Replikate auf unterschiedlichen Medien, so gilt zu jedem Zeitpunkt:

$$\bigcup_{v \in V} \Theta_\tau(v) \subset \Theta_\tau$$

$$\bigcup_{v \in V} \Theta_\sigma(v) = \Theta_\sigma \text{ und } \Theta_\sigma(v_i) \cap \Theta_\sigma(v_j) = \varnothing \text{ für alle } v_i, v_j \in V \text{ mit } v_i \neq v_j$$

Nicht, bzw. noch nicht exportierte Kacheln $\tau \in \Theta_\tau$ befinden sich im Speicherbereich SR. Bei regulärer, ausgerichteter oder irregulärer Kachelung gilt zusätzlich $\Theta_\tau(v_i) \cap \Theta_\tau(v_j) = \varnothing$ für alle $v_i, v_j \in V$ mit $v_i \neq v_j$. Bei arbiträrer Kachelung entfällt diese Einschränkung, aufgrund der

teilweisen redundanten Speicherung von Kacheln in mehreren Super-Kacheln. Anzumerken ist, dass aus Gründen der Datensicherung meist beim Beschreiben eines TS-Mediums gleichzeit ein gespiegeltes Backupmedium erstellt wird. Diese Backupmedien werden nach entsprechender Konfiguration des verwendeten HSM-System automatisch beschrieben. In dieser Arbeit werden Backupmedien aufgrund der automatischen Verwaltung durch das eingesetzte HSM-System nicht weiter betrachtet.

Definition 3.15: Menge der Schreib-/Lesestationen S und Menge der Roboterarme R eines HSM-Systems

Die Menge S der Schreib-/Lesestationen eines HSM-Systems, bzw. eines tertiären Speichersystems ist definiert durch $S = \{s_1, \ldots, s_{Ns}\}$. Dabei ist mit $N_S = |S|$ die Anzahl der in HEAVEN vorhandenen Schreib-/Lesestation $s \in S$ gegeben.

Die Menge R der in einem HSM-System vorhandenen unabhängigen Roboterarme ist spezifiziert durch $R = \{r_1, \ldots, r_{NR}\}$: Dabei ist mit $N_R = |R|$ die Anzahl der in HEAVEN vorhandenen Roboterarme $r \in R$ gegeben.

Spezifische Operationen für Zugriffe auf TS-Medien sind wie folgt definiert:

Definition 3.16: TS-Operationen Get-Volume o_{gv}, Load-Volume o_{lv}, Position-Volume o_{pv}, Rewind-Volume o_{rv}, Eject-Volume o_{ev} und Deposit-Volume o_{dv}

Die Operation *Get-Volume* (hole Medium) o_{gv} beinhaltet alle Vorgänge, um ein TS-Medium v mit dem Roboterarm aus der Medienbibliothek (Volume Shelter) zur Schreib-/Lesestation s zu befördern.

Die Operation *Load-Volume* o_{lv} entspricht dem Ladevorgang des TS-Mediums v in eine Schreib-/Lesestation s.

Eine Positionierungsoperation (*Position-Volume*) o_{pv} auf einem TS-Medium v ist erforderlich, um den Schreib-/Lesekopf an den Anfang einer Datei zu setzten.

Um ein TS-Medium v aus der Schreib-Lesestation s zu entfernen, ist es bei den meisten Medientypen zunächst erforderlich, das Medium zurückzuspulen o_{rv} (*Rewind-Volume*).

Die Operation *Eject-Volume* o_{ev} beinhaltet das Auswerfen eines TS-Mediums v aus der Schreib-/Lesestation s.

Die Operation *Deposit-Volume* o_{dv} beinhaltet alle Vorgänge, um ein TS-Medium v von einer Schreib-/Lesestation s mit dem Roboterarm r zurück zur Medienbibliothek zu bringen und dort abzulegen.

Die Mengen O_{gv}, O_{lv}, O_{pv}, O_{rv}, O_{ev}, bzw. O_{dv} kombinieren die jeweils für eine Anfrage q notwendigen Einzeloperationen o_{gv}, o_{lv}, o_{pv}, o_{rv}, o_{ev}, bzw. o_{dv}. Die Menge aller für eine Anfrage q erforderlichen Operationen ist definiert als $O_q = \{O_{gv}, O_{lv}, O_{pv}, O_{rv}, O_{ev}, O_{dv}\}$.

Definition 3.17: Operation zum Medienwechsel (Swap-Volume) o_{sv} bei TS-Systemen

Eine Medienwechseloperation (*Swap-Volume*) o_{sv} bei TS-Systemen setzt sich meist zusammen aus den Operationen *Rewind-Volume*, *Eject-Volume*, *Deposit-Volume*, *Get-Volume* und *Load-Volume*. Abhängig von der Anzahl an Schreib-/Lesestationen N_S und der Anzahl an unabhängigen Roboterarmen können einzelne Operationen vermieden, bzw. parallel ausgeführt werden. Die Menge O_{sv} vereint die für eine Anfrage q anfallenden Medienwechseloparationen. Die Menge aller für eine Anfrage q anfallenden Operationen $O_q = \{O_{sv}, O_{pv}\}$ mit $O_{sv} = \{O_{gv}, O_{lv}, O_{rv}, O_{ev}, O_{dv}\}$. Das Warten auf freie Ressourcen und die anfallenden Kosten für die Operationen zum Medienwechsel werden als Anlaufkosten bezeichnet.

Bei einigen HSM-Systemen, können über diese Definitionen hinaus noch weitere Begriffsbildungen erwähnt werden. Ein Volume-Set V_s besteht aus einem oder mehreren Medien, die zusammen eine logische Einheit bilden und wie ein großes Medium erscheinen. Volume-Sets können dazu benutzt werden, Gruppen von Dateien auf physikalischer Ebene zu kontrollieren. Ein File-Set ist ein Speicherbereich innerhalb eines Volume-Sets und kann sich somit über mehrere Medien erstrecken. Zusätzlich sind Übergangsfunktionen zwischen den einzelnen Hierarchieebenen zu definieren:

Definition 3.18: Funktionen *locate*(object), *move*(object, source, target), *replicate*(object, source, target), *access*(object), *buildST*(object) und *export*(object)

Die Funktion *locate*(τ) liefert die in der Speicherhierarchie am höchsten gelegene Speicherebene, welche das Objekt τ enthält:

$$
locate(\tau) = \begin{cases}
P & \text{, falls } \tau \in \Theta_P \\
SR & \text{, falls } \tau \notin \Theta_P \ \wedge \ \tau \in \Theta_{SR} \\
SRC & \text{, falls } \tau \notin \Theta_P \ \wedge \ \tau \notin \Theta_{SR} \ \wedge \ \tau \in \Theta_{SRC} \\
SHC & \text{, falls } \tau \notin \Theta_P \ \wedge \ \tau \notin \Theta_{SR} \ \wedge \ \tau \notin \Theta_{SRC} \ \wedge \ \tau \in \Theta_{SHC} \\
T_{on} & \text{, falls } \tau \notin \Theta_P \ \wedge \ \tau \notin \Theta_{SR} \ \wedge \ \tau \notin \Theta_{SRC} \ \wedge \ \tau \notin \Theta_{SHC} \ \wedge \ \tau \in \Theta_{T_{on}} \\
T_{near} & \text{, falls } \tau \notin \Theta_P \ \wedge \ \tau \notin \Theta_{SR} \ \wedge \ \tau \notin \Theta_{SRC} \ \wedge \ \tau \notin \Theta_{SHC} \ \wedge \ \tau \notin \Theta_{T_{on}} \ \wedge \ \tau \in \Theta_{T_{near}} \\
T_{off} & \text{, falls } \tau \notin \Theta_P \ \wedge \ \tau \notin \Theta_{SR} \ \wedge \ \tau \notin \Theta_{SRC} \ \wedge \ \tau \notin \Theta_{SHC} \ \wedge \ \tau \notin \Theta_{T_{on}} \ \wedge \ \tau \notin \Theta_{T_{near}} \\
 & \quad \wedge \ \tau \in \Theta_{T_{off}} \\
error & \text{, sonst}
\end{cases}
$$

Die Funktion *move*(object, source, target) verschiebt ein Objekt (object) $\tau \in \Theta_\tau$ von einer Speicherebene (source) zu einer anderen Speicherebene (target):
move : object, source, target → $\mathbb{B}$,
wobei gilt source, target ∈ {P, SR, SRC, SHC, T_{on}, T_{near}, T_{off}} für source ≠ target.

Die Funktion *replicate*(object, source, target) repliziert ein Objekt (object) $\tau \in \Theta_\tau$ von einer Speicherebene (source) zu einer anderen Speicherebene (target):
replicate : object, source, target → $\mathbb{B}$,
wobei gilt source, target ∈ {P, SR, SRC, SHC, T_{on}, T_{near}, T_{off}} für source ≠ target;

Die Funktion *access*(object) realisiert den Zugriff auf ein Objekt (object) $\tau \in \Theta_\tau$. Das beinhaltet die Lokalisierung des Objektes in der Speicherhierarchie und gegebenenfalls das Laden des Objektes in den Primärspeicher:

$$access(\tau) = \begin{cases} locate(\tau) & , \text{falls } \tau \in \Theta_P \\ locate(\tau);\ replicate(\tau,\ SR,\ P) & , \text{falls } \tau \in \Theta_{SR} \\ locate(\tau);\ replicate(\tau,\ SRC,\ P) & , \text{falls } \tau \in \Theta_{SRC} \\ locate(\tau);\ replicate(\tau,\ SHC,\ P) & , \text{falls } \tau \in \Theta_{SHC} \\ locate(\tau);\ replicate(\tau,\ T_{on},\ P) & , \text{falls } \tau \in \Theta_{T_{on}} \\ locate(\tau);\ move(\tau,\ T_{near},\ T_{on});\ replicate(\tau,\ T_{on},\ P) & , \text{falls } \tau \in \Theta_{T_{near}} \\ locate(\tau);\ move(\tau,\ T_{off},\ T_{on});\ replicate(\tau,\ T_{on},\ P) & , \text{falls } \tau \in \Theta_{T_{off}} \end{cases}$$

Die Funktion *buildST*(object) kombiniert räumlich benachbarte Kacheln $\tau \in \Theta_\tau$ eines Objektes (object) $\alpha \in \Theta_\alpha$ zu Super-Kacheln $\sigma \in \Theta_\sigma$. Dabei verwendet *buildST*(object) für reguläre, ausgerichtete und irreguläre Kachelung das eSTAR-Konzept[30].

Die Funktion *export*(object) übernimmt das Exportieren eines Objektes (object) $\alpha \in \Theta_\alpha$ auf ein tertiäres Speichermedium. Je nach Konfiguration von HEAVEN und dem verwendeten HSM-System, sind vier Möglichkeiten zu unterscheiden:

$$export(\alpha) = \begin{cases} buildST(\alpha);\ move(\alpha,\ P,\ SRC);\ replicate(\alpha,\ SRC,\ T_{on}) & \vee \\ buildST(\alpha);\ move(\alpha,\ P,\ SRC);\ replicate(\alpha,\ SRC,\ SHC);\ move(\alpha,\ SHC,\ T_{on}) & \vee \\ buildST(\alpha);\ move(\alpha,\ P,\ SHC);\ replicate(\alpha,\ SHC,\ T_{on}) & \vee \\ buildST(\alpha);\ move(\alpha,\ P,\ T_{on}) & \end{cases}$$

Nach der Beschreibung allgemeiner Definitionen folgen nähere Betrachtungen zum Exportieren von MDD auf tertiäre Speichermedien.

3.6.1 Export von multidimensionalen Daten bei HEAVEN

Um das Exportieren von multidimensionalen Daten von Sekundärspeicher auf Tertiärspeicher zu vereinfachen, wurde die multidimensionale Anfragesprache RasQL um eine Export-Anweisung erweitert. Das neue Konstrukt ist wie folgt aufgebaut:

```
EXPORT  FROM        <Kollektion>
        [ WHERE     <Selektionsbedingung> ]
        [ LOCATION  <Speicherort> ]
```

Wird eine Exportanweisung ohne die optionalen Klauseln ausgeführt, so werden alle in einer Kollektion enthaltenen MDD auf TS-Medien exportiert. Durch die WHERE-Klausel ist es

[30] Bei arbiträrer Kachelung wird STAR verwendet.

möglich, beliebige MDD, die einer Selektionsbedingung genügen, als Exportkandidaten aus einer Kollektion auszuwählen. Um eine Auswahl zu treffen, können alle in RasDaMan zur Verfügung stehenden Operationen (Kapitel 2.5.5) angewendet werden. Da HEAVEN zulässt, gleichzeitig mehrere HSM-Systeme an RasDaMan anzukoppeln, kann mit der LOCATION-Klausel der entsprechende Speicherort angegeben werden. Eine Auflistung der jeweiligen Speicherorte, ist in der von HEAVEN verwendeten Konfigurationsdatei *TertiaryStorage.conf* anzugeben. Ist kein Speicherort mittels LOCATION gewählt, wird der in der Konfigurationsdatei an erster Stelle stehende Speicherort als Voreinstellung verwendet. Durch die gleichzeitige Nutzung mehrerer HSM-Systeme, stellt HEAVEN zum Beispiel eine komfortable Möglichkeit zur Verfügung, bei Bedarf eine komplette Systemumstellung durchzuführen. So können von HEAVEN verwaltete Daten eines veralteten HSM-Systems, durch eine REIMPORT-Anweisung (Kapitel 3.8), zurück in den Sekundärspeicherbereich (SR) von RasDaMan geladen und durch einen erneuten Export auf ein neues HSM-System verlagert werden. Entsprechend des vorhandenen Datenvolumens, kann dieser Vorgang allerdings sehr lange dauern.

Um Daten auf TS-Medien zu exportieren, ist für jede Kollektion explizit eine entsprechende RasQL-Anweisung anzustoßen. Durch dieses Vorgehen ist es möglich, häufig frequentierte Datensätze im Sekundärspeicherbereich (SR) des von RasDaMan verwendeten relationalen DBMS dauerhaft vorzuhalten. Das garantiert für diese Datensätze ein schnelles Datenretrieval. Sollen Datensätze direkt nach dem Einfügen, automatisch auf Tertiärspeicher geschrieben werden, so kann sehr einfach die Exportanweisung in das Einfügeprogramm integriert werden. Der Exportvorgang erfolgt daraufhin ohne weitere Nutzerinteraktion. Denkbar ist auch ein regelbasiertes, automatisiertes Exportieren. Dabei können heuristische Erhebungen eines Systemkataloges verwendetet werden, um den Export bestimmter Datensätze bei niedriger Systemlast (z.B. in der Nacht) automatisch durchzuführen. Als Auslagerungskriterien können das Einfügedatum, die Zugriffshäufigkeit und der Zeitpunkt des letzten Zugriffes herangezogen werden. Ein solcher Systemkatalog für RasDaMan wurde in einer Diplomarbeit, die im Rahmen dieser Dissertation entstanden ist, für diesen Zweck vorbereitet [Albe04].

Entkoppelter Exportvorgang

Wird der Exportvorgang von HEAVEN analysiert, so ist schnell ein positiver Effekt zu bemerken. Durch die Verwendung eines HSM-Systems kann das kostenintensive Migrieren von Daten auf tertiäre Speichermedien vollständig von RasDaMan entkoppelt werden. Da die Eigenentwicklung des Tape-Controller-Toolkits (TCT, Kapitel 3.3), die Funktionsweise eines HSM-Systems nachbildet, wird die Entkopplung des Exportvorganges auf gleiche Weise realisiert. Erreicht wird das durch den HSM-Cache (SHC), da die zu exportierenden MDD aus der Sicht von RasDaMan, nur in diesen Sekundärspeicher geschrieben werden müssen. Das anschließende, zeitaufwändige Migrieren der MDD (in Granularität von Super-Kacheln) auf TS-Medien, erfolgt automatisch durch das HSM-System, ohne Interaktion von RasDaMan. So kann RasDaMan in dieser Zeit neue Aufträge bearbeiten, was zu einer sehr guten Performance führt. Abbildung 3.21 stellt die Zustandsübergänge für dieses entkoppelte Exportieren von MDD dar.

Das obere Zustandsübergangsdiagramm zeigt den Exportvorgang von RasDaMan und das untere Diagramm das Schreiben der MDD vom HSM-Cache auf TS-Medien. Wie bereits aus der Definition 3.18 zu entnehmen, umfasst die RasDaMan Exportfunktion *export*(α) das Bestimmen und Generieren der Super-Kacheln. Die Funktion *buildST*(α) verwendet dazu den eSTAR, um Intra-Super-Tile-Clustering zu erreichen. Nach diesem Vorgang wird das MDD α, in Super-Kachel-Granularität, vom Primärspeicher (P) in den HSM-Cache (SHC) verschoben. Der Einfachheit halber, wurde in Abbildung 3.21 das Verschieben der multidimensionalen Daten angenommen. Alternativ kann, wie in der Definition 3.18 beschrieben, auch mittels *replicate*(α, P, SHC) eine Replikation der Daten erfolgen, um durch Ausnutzung von Caching ein effizienteres Datenretrieval zu erreichen. Die Daten verweilen zusätzlich im TS-Cache-Bereich des konventionellen DBMS von RasDaMan (SRC). HEAVEN unterstützt entweder eine Datenübertragung über NFS oder mittels FTP. Nach dem Übertragen der Daten, ist der Exportauftrag für RasDaMan beendet und neue Aufträge können bearbeitet werden.

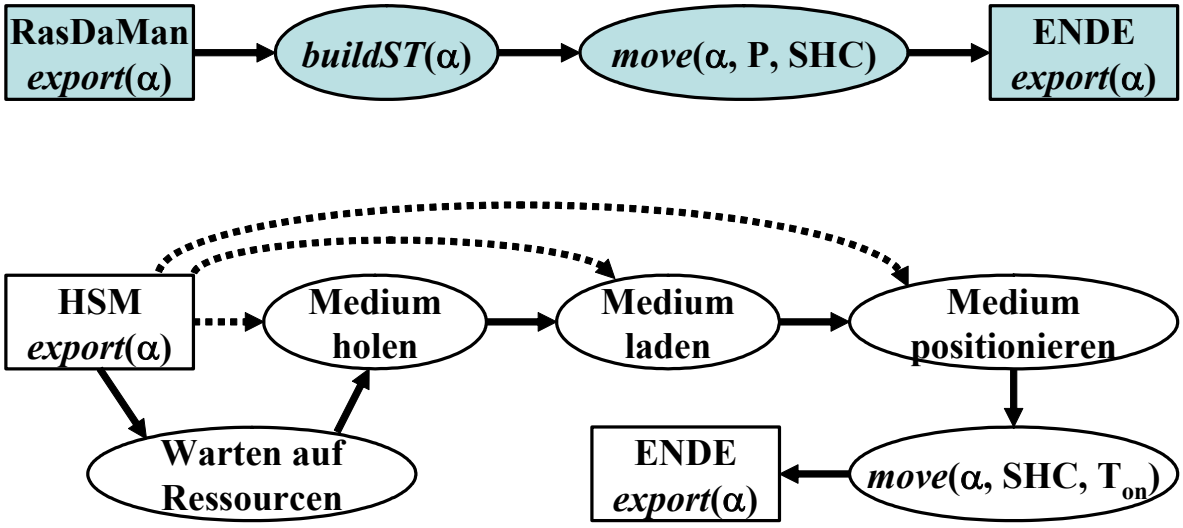


Abbildung 3.21: Zustandsübergänge für das Exportieren von MDD auf TS-Medien.

Der entkoppelte Exportvorgang des HSM-Systems (bzw. TCT), wird in der unteren Darstellung der Abbildung 3.21 gezeigt. Der Weg der wahrscheinlichsten Zustandsübergänge bei einem Erstzugriff ist an den durchgehenden Pfeilen zu erkennen. Die gestrichelten Pfeile weisen je nach Zustand des HSM-System alternative Pfade auf. So besteht zum Beispiel die Möglichkeit, dass das gewünschte Medium bereits in einer Schreibstation eingelegt ist und somit nur noch positioniert (Operation *Position-Volume*: o_{pv}) werden muss, bevor der Schreibvorgang beginnen kann. Je nach gewählter Auslagerungsstrategie des HSM-Systems, werden die im HSM-Cache neu eingetroffenen Daten nach bestimmten Kriterien verdrängt. Das kann von einer sofortigen Migration, bis hin zu einer intelligenten, lastabhängigen Migration gehen. Nähere Details sind den jeweiligen Administrationshandbüchern zu entnehmen [IBM03, Sun02]. Beim Beginn der Migration von Daten aus dem Cache-Bereich des HSM-Systems, ist es in den meisten Fällen zunächst notwendig, auf die benötigten Ressourcen zu warten (Abbildung 3.21, unten). Dazu zählen zum Beispiel ein freier Roboterarm $r \in R$ und

eine freie Schreib-/Lesestation s ∈ S. Stehen diese Ressourcen zur Verfügung, so wird das TS-Medium mit dem Roboterarm aus der Medienbibliothek geholt (Operation *Get-Volume*: o_{gv}) und in einem nächsten Schritt in die Schreib-/Lesestation eingelegt (Operation *Load-Volume*: o_{lv}). Danach wird das Speichermedium an die richtige Stelle positioniert (Operation *Position-Volume*: o_{pv}) und das Migrieren der MDD vom HSM-Cache auf das TS-Medium wird durchgeführt. Die Speichergranularität besteht dabei aus Super-Kacheln. Da die einzelnen Super-Kacheln sequentiell auf das TS-Medium geschrieben werden, ist kein aufwändiges Positionierung zwischen diesen Super-Kacheln notwendig. Nach dem Schreiben der letzten Super-Kachel, ist der Exportvorgang für das HSM-System beendet. Das Medium verbleibt allerdings zunächst in der Schreib-/Lesestation. Dieses Vorgehen wird als „Lazy Eject" (träges Auswerfen) bezeichnet. Es wird von einer erhöhten Wahrscheinlichkeit ausgegangen, dass in naher Zukunft weitere Aufträge das gleiche Medium betreffen. Ist dies der Fall, entfallen teure Medienwechseloperationen. Die Verweildauer eines Mediums in einer Schreib-/Lesestation, hängt stark von der Anzahl der vorhandenen Schreib-/Lesestationen und deren Frequentierung ab. Das Zurückspulen (Operation *Rewind-Volume*: o_{rv}), Auswerfen (Operation *Eject-Volume*: o_{ev}) und das Zurückbringen eines Mediums in die Medienbibliothek (Operation *Deposit-Volume*: o_{dv}), geht indirekt über die Wartezeit auf freie Ressourcen in die Exportzeit mit ein.

Abbildung 3.22 zeigt den entkoppelten Exportvorgang eines MDD α mit einem Datenvolumen von 1,32 GByte (28 Super-Kacheln mit je 48,7 MByte). Das linke Balkendiagramm veranschaulicht den Exportvorgang von RasDaMan. Es werden der Reihe nach die einzelnen Super-Kacheln mit *buildST*(α) generiert und anschließend mit *move*(α, P, SHC) in den Cache-Bereich des HSM-Systems SHC transferiert.

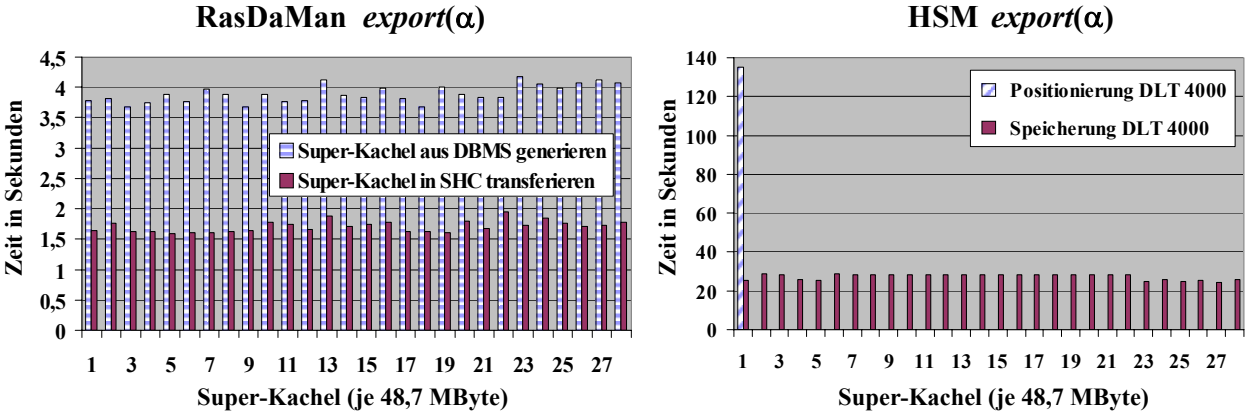


Abbildung 3.22: Vergleich des entkoppelten Exportvorganges von HEAVEN.

Wurde die letzte Super-Kachel in den Speicherbereich SHC übertragen, ist der Exportvorgang für RasDaMan beendet. Das rechte Balkendiagramm der Abbildung 3.22 zeigt den zeitaufwändigen entkoppelten Exportvorgang des HSM-Systems. Sobald die erste Super-Kachel von RasDaMan in den Speicherbereich SHC geladen wurde, kann das Speichern auf dem Magnetband beginnen. Dabei ist zunächst eine initiale Positionierung zum Beginn des freien

Speicherbereiches auf dem Magnetband notwendig. Danach erfolgt das kontinuierliche, schrittweise Schreiben der 28 Super-Kacheln. Die hier gezeigte Messung wurde mit dem Fünffach-Wechsler DLT-LXLS der Firma Overland Data mit einem integrierten DLT 4000 Laufwerk durchgeführt.

3.6.2 Exportieren mit Intra- und Inter-Super-Tile-Clustering

Um ein performantes Datenretrieval zu garantieren, sind bereits beim Exportieren von Datenobjekten einige Besonderheiten zu beachten. Die Tertiärspeicherschicht in einem hierarchischen Speichersystem, ist charakterisiert durch sehr große Datenvolumina und sehr hohe Anfragezeiten für Zugriffe. Das liegt an der Verwendung zahlreicher, günstiger Wechselmedien (z.B. Magnetbänder), die sich eine geringe Anzahl an teuren Schreib-/Lesestationen S und Robotermechanismen R teilen. Die hohe Zugriffszeit wird typischerweise dominiert durch die Medienwechselzeit (12 s – 40 s) und durch die hohe Positionierungszeit (27 s – 95 s). Die Transferrate eines TS-Systems, ist dagegen sehr gut (36 MByte/s bei Super-DLT 600) und nur um etwa den Faktor Zwei langsamer als die Transferraten von Festplatten. Zugriffe auf Daten, die auf Tertiärspeicher abgespeichert wurden, können unakzeptable Verzögerungen mit sich bringen. Sie werden durch notwendige Positionierungsoperationen oder sogar Medienwechsel verursacht. Die Notwendigkeit von Positionierungsoperationen und Medienwechsel, wird durch die Datenverteilung auf den TS-Medien (speziell Magnetbänder) diktiert. Vernünftiges Platzieren der Datenobjekte auf TS-Medien, ist aus diesen Gründen kritisch und kann sehr stark die Performance des Gesamtsystems beeinflussen. Die Datenplatzierung auf TS-Medien wird durch den Exportvorgang festgelegt. Das Problem der optimalen Platzierung von Daten auf Magnetbänder ist bedingt durch zwei Teilprobleme: 1) Eine geschickte Verteilung der Daten auf unterschiedliche Medien kann häufige Medienwechseloperationen[31] o_{sv} vermeiden. 2) Eine optimale Anordnung der Daten innerhalb eines Mediums reduziert Positionierungsoperationen o_{pv}.

1) Datenverteilung auf unterschiedliche Medien

Die einzige Möglichkeit, Medienwechselkosten zu minimieren ist, die Anzahl der verschränkten Datenzugriffe auf unterschiedliche Medien zu reduzieren. Durch das Speichern gleichartiger Objekte auf dem gleichen Medium, können häufige Medienwechsel vermieden werden. Das liegt daran, dass bei Anfragen meist gleichartige Objekte gemeinsam angefasst werden, bzw. miteinander verglichen werden. Die Wahrscheinlichkeit, dass innerhalb einer Anfrage Operationen auf Objekten, mit zum Beispiel unterschiedlicher Domäne D oder Basisdatentyp T, ausgeführt werden, ist vielfach geringer. Diese Beobachtung wurde auch durch die Erfahrungen aus dem Projekt ESTEDI bestätigt. Für das Exportieren von Datenobjekten auf TS-Medien, können daraus für *HEAVEN* zwei Forderungen abgeleitet werden:

- Alle Super-Kacheln σ eines MDD α sollen auf einem TS-Medium gespeichert werden.
- Alle MDD α einer Kollektion C sollen auf einem TS-Medium, bzw. wenigen TS-Medien gespeichert werden.

[31] Eine Medienwechseloperation (Swap-Volume) o_{sv} setzt sich meist zusammen aus den Operationen Rewind-Volume o_{rv}, Eject-Volume o_{ev}, Deposit-Volume o_{dv}, Get-Volume o_{gv} und Load-Volume o_{lv}.

Wichtig für ein performantes Datenretrieval ist vor allem, dass die Anzahl der involvierten TS-Medien möglichst minimal ist. Dadurch können Medienwechsel eingespart werden. Eine gewollte Verteilung von Datenobjekten auf mehreren TS-Medien, um parallele Hardwarestrukturen zu nutzen, wird hier nicht betrachtet. Diese Maßnahme wird auch als Striping bezeichnet und entspricht dem RAIT Level 0.

Selbstverständlich hängt die Durchführbarkeit der beiden Forderungen, von dem freien Speicherplatz auf dem TS-Medium und dem Speichervolumen der Super-Kacheln, bzw. MDD einer Kollektion ab. Ein Unsicherheitsfaktor ist die aktivierte Hardwarekomprimierung der TS-Systeme. Der zu erreichende Komprimierungsgrad ist stark abhängig von den zu speichernden Datenobjekten. Zumindest die Bestrebung, alle Super-Kacheln eines MDD auf einem TS-Medium zu speichern, lässt sich meist realisieren. Voraussetzung hierfür ist das kontinuierliche Exportieren aller Super-Kacheln eines MDD auf das TS-Medium. Bei der Erweiterung der Anfragesprache RasQL um eine Export-Anweisung, wurde darauf geachtet, dass die Super-Kacheln eines MDD nur komplett auf ein TS-Medium geschrieben werden können.
Für die zweite Forderung gelten ähnliche Überlegungen. Allerdings ist es hier oft nicht möglich, alle MDD einer Kollektion aufgrund des Speichervolumens auf ein TS-Medium zu speichern. Als Hauptbestrebung wird versucht, die Anzahl der betroffenen TS-Medien so gering wie möglich zu halten. Bei der Ausführung der Export-Anweisung ohne Angabe einer Selektionsbedingung, werden alle Super-Kacheln aller in einer Kollektion enthaltenen MDD kontinuierliche auf ein TS-Medium, bzw. bei Bedarf auf mehrere TS-Medien, geschrieben. Das gewährleistet, dass keine kollektionsfremden Datenobjekte in der Zwischenzeit auf das gleiche Medium geschrieben werden.

2) Datenverteilung innerhalb eines Mediums
Neben der Minimierung der Anzahl an TS-Medien, die für die Speicherung von MDD, bzw. Kollektionen verwendet werden, um Medienwechseloperationen zu reduzieren, ist die Datenverteilung auf einem TS-Medium für ein performantes Datenretrieval von großer Bedeutung. Durch eine geschickt gewählte Datenverteilung, lässt sich bei Anfragen die Anzahl der Positionierungsoperationen reduzieren. Datenobjekte (Super-Kacheln) sollen im Idealfall direkt aufeinander folgend und in gleicher Reihenfolge auf dem TS-Medium vorliegen, wie sie später bei der Anfrageausführung angefordert werden (Kapitel 3.7). Grundsätzlich werden keine Speicherobjekte (Super-Kacheln) unterschiedlicher MDD oder Kollektionen verschränkt abgespeichert. Das bedeutet, es werden alle Super-Kacheln σ eines MDDs α zusammenhängend auf einem TS-Medium gespeichert: Gefolgt von den restlichen MDD der Kollektion C und weiteren Kollektionen, bis die Kapazitätsgrenze des TS-Mediums erreicht ist.

Wird ein komplettes MDD α angefragt, ist nach einer initialen Positionierungsoperation zur ersten Super-Kachel σ durch einen kontinuierlichen Lesestrom, keine weitere Positionierung mehr erforderlich. Ist bei einer Bereichsanfrage nicht der gesamte Datenraum eines MDD α betroffen, so sind zusätzliche Positionierungsoperationen nicht zu vermeiden. Diese werden

durch das Überspringen, bzw. Überlesen von nicht für die Anfrage relevanten Datenelementen σ hervorgerufen. Das wiederum liegt an der linearisierten, eindimensionalen Repräsentation der multidimensionalen Datenobjekte (MDD) auf dem tertiären Speichermedium. Die Anzahl und Ausprägung der zusätzlichen Positionierungsoperationen, ist abhängig von der Reihenfolge der Datenobjekte auf dem TS-Medium. Ziel ist es, die Anzahl und Zeitdauer der Positionierungsoperationen zu minimieren. Um das zu erreichen, ist es notwendig, die räumliche Nachbarschaft von Zellen, bzw. Kacheln τ des multidimensionalen Raumes so gut wie möglich auf dem TS-Medium beizubehalten. Beim Speichern von d-dimensionalen Daten auf einem Magnetband, wird der multidimensionale Datenraum auf eine Dimension linearisiert. Die Anzahl der direkten, räumlichen Nachbarn sinkt dabei von 3^d-1 auf 2 direkte, räumliche Nachbarn (Kapitel 3.5.1). Eine Linearisierung multidimensionaler Datenräume bedeutet immer einen gewissen Verlust in der räumlichen Nachbarschaft. Mittels einer geeigneten Linearisierungsordnung, kann allerdings die Nähe von benachbarten Zellen mehr oder weniger gut aufrechterhalten werden.

Bei HEAVEN wird die Aufrechterhaltung der räumlichen Nachbarschaft von Zellen, bzw. Kacheln τ, in zwei Schritten durchgeführt: Um eine höhere Datengranularität für TS-Zugriffe zu erhalten, werden zunächst durch das entwickelte eSTAR-Verfahren, räumlich benachbarte Kacheln τ im Datenraum aus den in Kapitel 3.5 besprochenen Gründen zu Super-Kacheln σ kombiniert. Das Beibehalten der räumlichen Nachbarschaft von Kacheln innerhalb einer Super-Kachel, wird als Intra-Super-Tile-Clustering bezeichnet. Die Bildung der Super-Kacheln erhält vollständig die räumliche Nachbarschaft zwischen Kacheln τ und Super-Kacheln σ im Datenraum eines MDD α. Neben den Vorteilen einer größeren Zugriffsgranularität, der Aufrechterhaltung von Vorzugsdimensionen und der Beibehaltung der räumlichen Nachbarschaft, wird die Anzahl der zu speichernden Datenobjekte (Super-Kacheln) stark reduziert.
In einem weiteren Schritt, werden die Super-Kacheln σ eines MDD α auf ein TS-Medium geschrieben. Durch die verwendete Linearisierungsordnung, wird das Inter-Super-Tile-Clustering realisiert. Die notwendige linearisierte Abspeicherung, der im multidimensionalen Datenraum des MDD α enthaltenen Super-Kacheln σ, begrenzt, wie beschrieben, die Anzahl der direkten räumlichen Nachbarn auf Zwei. Auch die beste Linearisierungsordnung, kann durch die verminderte räumliche Repräsentationen der Datenobjekte auf dem TS-Medium bei Bereichsanfragen, nicht alle Positionierungsoperationen einsparen. In diesem Zusammenhang bietet die eingeführte Granularität der Super-Kachel den positiven Nebeneffekt, dass durch das Intra-Super-Tile-Clustering, bereits im Vorfeld, die Menge der möglichen Positionierungsoperationen sehr stark reduziert wird. Zum Beispiel bei einem MDD mit einem Datenvolumen von 25 GByte, einer Kachelgröße von 128 KByte und einer Super-Kachel-Größe von 150 MByte, reduziert sich die Anzahl der auf einem TS-Medium zu speichernden Objekten von 204.800 Kacheln auf 171 Super-Kacheln (Faktor 1.197).

Abbildung 3.23 zeigt den Exportvorgang eines MDD α auf ein Magnetband, unter Berücksichtigung des Intra- und Inter-Super-Tile-Clustering. In der linken oberen Hälfte ist das gekachelte, zweidimensionale MDD α zu erkennen. Bei einem anstehenden Exportiervorgang

werden zunächst durch das eSTAR-Verfahren jeweils vier Kacheln τ zu Super-Kacheln σ kombiniert (Intra-Super-Tile-Clustering; Abbildung 3.23 rechts oben). Anschließend erfolgt das Schreiben der Super-Kacheln auf ein Magnetband, entsprechend einer Linearisierungsordnung[32] (Inter-Super-Tile-Clustering). Im unteren Bereich der Abbildung 3.23 ist das Magnetband mit den gespeicherten Super-Kacheln σ und den darin enthaltenen Kacheln τ zu erkennen. Gerade die Linearisierung eines multidimensionalen Raumes auf eine Dimension, ist immer mit einem Verlust der räumlichen Nachbarschaft verbunden.

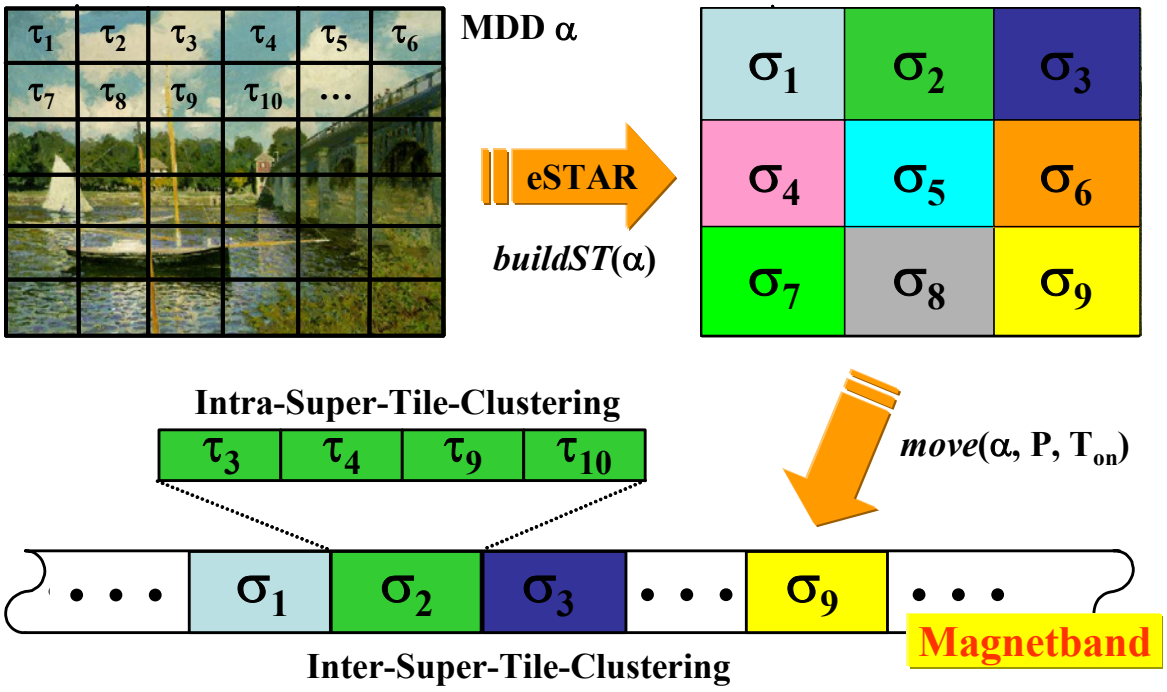


Abbildung 3.23: Exportvorgang eines MDD α auf ein Magnetband.

Bei *HEAVEN* wird die durch den R^+-Baum vorgegebene Zugriffsreihenfolge auf die Teilobjekte des multidimensionalen Objektes α, als Linearisierungsordnung bei der Speicherung der Super-Kacheln auf TS-Medien verwendet. Durch die Verwendung des in RasDaMan eingesetzten R^+-Baumes zur Generierung der Exportordnung, ist keine weitere Berechnung, bzw. Bestimmung einer Ordnung notwendig. Auch bei einem späteren Datenretrieval ist kein zusätzlicher Aufwand erforderlich, um die Ladereihenfolge der für eine Anfrage benötigten Super-Kacheln zu bestimmen. Abbildung 3.24 zeigt exemplarisch, die durch den R^+-Baum generierte Linearisierungsordnung der beiden MDD α und β.

Bei dem zweidimensionalen MDD α, handelt es sich um ein Satellitenbild mit einem Datenvolumen von 1 GByte. Für dieses Beispiel wurde eine Super-Kachel-Größe von 50 MByte veranschlagt. Die entstandenen 21 Super-Kacheln sind entsprechend der durch den R^+-Baum vorgegebnen Linearisierungsordnung aufsteigend nummeriert (σ_1 bis σ_{21}). Zusätzlich ist die

[32] Aus Gründen der Übersichtlichkeit wurde in der Abbildung 3.23 eine reihenweise Abtastung (compound-order; row-order) als Linearisierungsordnung gewählt. Diese entspricht nicht der realisierten Linearisierung.

erste Super-Kachel σ_1 durch die Farbe Grün und die letzte Super-Kachel σ_{21} durch die Farbe Gelb hervorgehoben. Die roten Pfeile zeigen den Verlauf der Linearisierung. Sprünge innerhalb des multidimensionalen Datenraumes sind als rote, gepunktete Pfeile dargestellt. Es ist zu erkennen, dass die durch die Datenmodellierung (Kachelung) entstandene Vorzugsdimension, bei der Exportreihenfolge beibehalten wird.

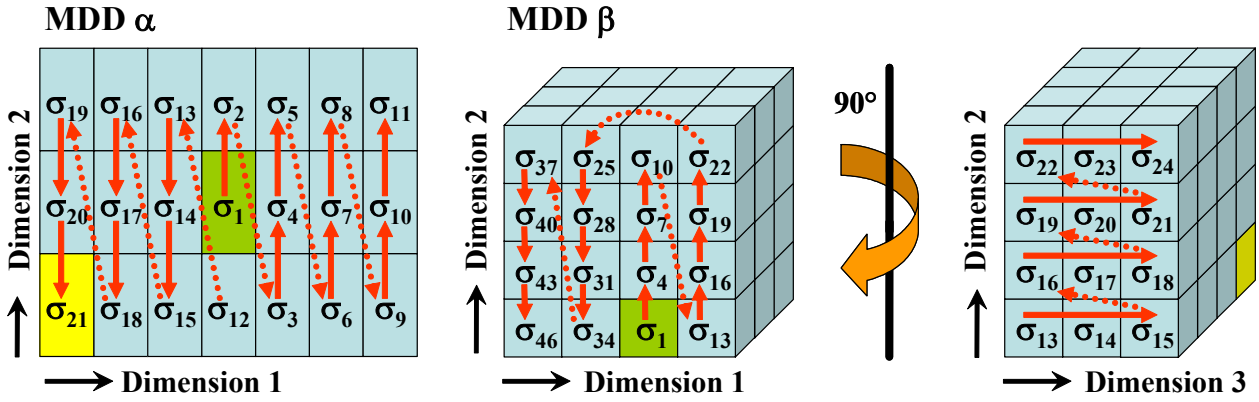


Abbildung 3.24: Durch den R^+-Baum vorgegebene Linearisierungsordnung zweier MDD.

Das mittlere und rechte Objekt der Abbildung 3.24 zeigt das MDD β. Dabei handelt es sich um eine dreidimensionale Klimasimulation mit einem Datenvolumen von 2,64 GByte, bestehend aus 48 Super-Kacheln. Auch hier wurden die Super-Kacheln, entsprechend der Anfrageordnung des R^+-Baumes, nummeriert und die erste und letzte Super-Kachel farblich markiert. Um den Linearisierungsverlauf im dreidimensionalen Fall besser erkennen zu können, ist das MDD β zusätzlich um 90° gedreht dargestellt.

Zu erwähnen ist, dass die durch den R^+-Baum generierte Linearisierungsordnung, die räumliche Nachbarschaft der Super-Kacheln nicht optimal nachbildet. Eine bessere Linearisierung ist vor allem bei höherdimensionalen Datenräumen durch raumfüllende Kurven (*Space Filling Curves*, SFC) zu erreichen. Allerdings steigt der Aufwand zur Bestimmung der Linearisierungsordnung. Die bekanntesten Vertreter hierbei sind die Z-Kurve [Most66] und die Hilbert-Kurve [Hilb1891]. Abbildung 3.25 zeigt die Linearisierung eines zweidimensionalen 4x4 Datenraumes mittels der Z-Kurve (links) und der Hilbert-Kurve (rechts).

Betrachtet man die Z-Kurve, so ist es möglich, durch Verschränkung der Bit-Repräsentation (Bit-Interleaving) die Z-Werte zu berechnen. Diese Berechnungsmethode geht zurück auf [OM84]. Das Prinzip der Bestimmung der Z-Werte, ist in Abbildung 3.25 zu erkennen. In dem dargestellten 4x4 Datenraum ist die Bit-Repräsentation in jeder Dimension durch 2 Bits möglich. Um den Z-Wert 9 zu erhalten, werden die entsprechenden Bits der Dimension 1 (Bits 01) und der Dimension 2 (Bits 10) verschränkt. Beginnend beim ersten Bit der Dimension 2 (Bit 1), folgt das erste Bit der Dimension 1 (Bit 0). Danach folgt das zweite Bit der Dimension 2 (Bit 0) und das zweite Bit der Dimension 1 (Bit 1). Insgesamt ergibt das die Bitfolge 1001, was dem Z-Wert 9 entspricht.

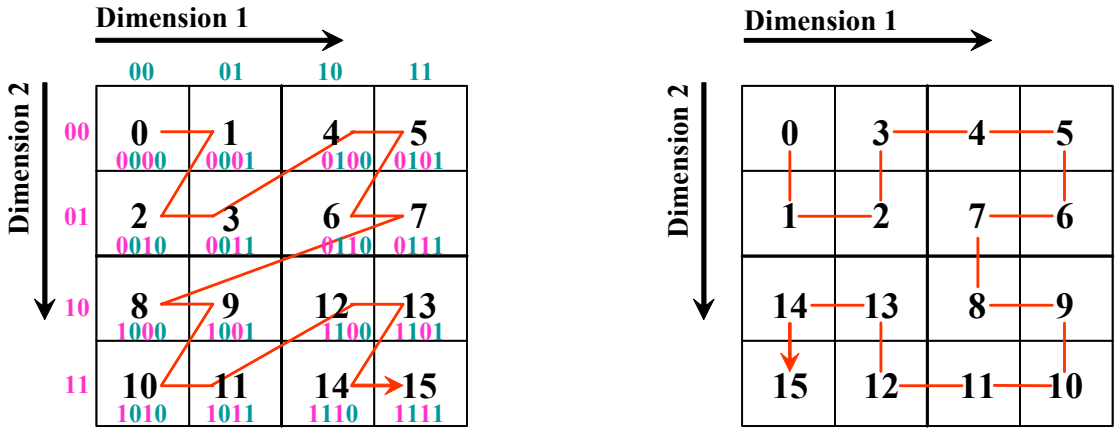


Abbildung 3.25: Z-Kurve (links) und Hilbert-Kurve (rechts) eines 4x4 Datenraumes.

Ein direkter Vergleich der beiden raumfüllenden Kurven aus Abbildung 3.25 lässt erkennen, dass bei der Z-Kurve (links) Sprünge auftreten. Obwohl die Objekte 7 und 8 in der Z-Ordnung aufeinander folgen, sind sie räumlich gesehen keine direkten Nachbarn. Die Hilbert-Kurve repräsentiert die räumliche Nähe besser, da keine Sprünge vorkommen. Eine Sprungfreiheit ist bei der Hilbert-Kurve allerdings nur gewährleistet, wenn Datenräume mit gleichen Ausdehnungen in allen Dimensionen vorliegen ($extent(\alpha)[1] = extent(\alpha)[2] = \ldots = extent(\alpha)[d]$). In vielen Bereichen, wie auch bei HEAVEN, kann davon grundsätzlich nicht ausgegangen werden. Eine gute Übersicht und einen Vergleich unterschiedlicher raumfüllender Kurven bietet [Widh04]. Als klassischen Vertreter für die Verwendung der Z-Ordnung zur Indexierung multidimensionaler Daten, ist der UB-Baum (Universal B-Tree) zu erwähnen [Baye97]. Bei der Indexierung mittels eines UB-Baumes werden dünn besetzte multidimensionale Datenräume, zum Beispiel eines Data Warehouses, durch eine Z-Kurve linearisiert und auf einen B^+-Baum abgebildet. Bei HEAVEN wird im Gegensatz dazu, mit dicht besetzten multidimensionalen Datenräumen (Array-Daten) gearbeitet. Die prinzipiellen Unterschiede von dünn, bzw. dicht besetzten Datenräumen, wurden in Kapitel 2.1 behandelt.

Nach der allgemeinen Beschreibung von raumfüllenden Kurven, ist zu klären, wie diese bei HEAVEN verwendet werden können, um eine bessere Exportordnung zu erreichen. Um eine raumfüllende Kurve für die durch das eSTAR-Verfahren ermittelten Super-Kacheln zu generieren, ist es hilfreich, die räumlich ausgedehnten Super-Kacheln auf den Mittelpunkt zu reduzieren. Der Mittelpunkt (Center) einer Super-Kachel σ berechnet sich wie folgt:

$$center(sdom(\sigma))[i] = low(sdom(\sigma))[i] + \left\lceil \frac{high(sdom(\sigma))[i] - low(sdom(\sigma))[i]}{2} \right\rceil$$

es gilt $i \in \{1, ..., d\}$

Durch die Reduzierung aller in einem MDD α enthaltenen, räumlich ausgedehnten Super-Kacheln σ auf die jeweiligen Mittelpunkte, entsteht ein dünn besetzter Datenraum mit der Domäne des MDD α. Nach erfolgter Bestimmung der Mittelpunkte, wird entsprechend der

verwendeten raumfüllenden Kurve (SFC) eine Ordnung bestimmt. Wird die Z-Kurve herangezogen, kann, wie bereits beschrieben, durch Verschränkung der Bit-Repräsentation (Bit-Interleaving) der Super-Kachel-Mittelpunkte, die Z-Werte berechnet werden. Entsprechend der entstandenen Linearisierungsordnung, werden die Super-Kacheln auf das TS-Medium exportiert. Diese Z-Ordnung ist ebenfalls beim Datenretrieval anzuwenden. Somit werden die Datenobjekte in gleicher Reihenfolge angefragt, wie sie auf dem TS-Medium vorliegen. Durch die Verwendung von SFCs ist es möglich, alle Objekte mit einem Durchlauf des Mediums zu erreichen, ohne zusätzliche Start-, Stopp- und Rückspuloperationen. Durch die verwendete raumfüllende Kurve, ist besonders für höherdimensionale Daten, die räumliche Nachbarschaft der Super-Kacheln auch auf den TS-Medien gewährleistet. Zeitintensive Positionierungsoperationen werden minimiert. Um beim Datenretrieval nicht jedes Mal die Berechnung der Z-Ordnung durchführen zu müssen, kann die Ordnung der Super-Kacheln in den entsprechenden Super-Knoten des R^+-Baum integriert werden.

3.6.3 Erreichtes Optimierungspotential

Die Art und Weise, wie das Exportieren multidimensionaler Daten auf TS-Medien vollzogen wird, hat starke Auswirkungen auf ein künftiges performantes Datenretrieval. Durch eine geschickt gewählte Exportreihenfolge der Datenobjekte und eine durchdachte Speicherung der Daten auf den vorhandenen TS-Medien, werden beim Datenretrieval Medienwechsel und Positionierungsoperationen minimiert.

Als entscheidender Punkt steht die Aufrechterhaltung der räumlichen Nachbarschaft von Zellen innerhalb des multidimensionalen Datenraumes. Um die Verluste bei der Linearisierung einer mehrdimensionalen Repräsentation der MDD auf die eindimensionale Abbildung auf einem TS-Medium möglichst gering zu halten, erfolgt bei HEAVEN der Exportvorgang in zwei Schritten: Zunächst werden durch das eSTAR-Verfahren einzelne Kacheln zu Super-Kacheln kombiniert, um eine größere Zugriffsgranularität zu erhalten. Bei diesem Übergang bleibt die räumliche Nachbarschaft vollständig erhalten (Intra-Super-Tile-Clustering). Durch das implementierte eSTAR-Verfahren, werden auch die in der Datenmodellierung entstandenen Vorzugsdimensionen beibehalten. In einem weiteren Schritt erfolgt die Bestimmung der Linearisierungsordnung, die beim Schreiben der Super-Kacheln auf TS-Medium verwendet wird. Bei HEAVEN wird zur Ermittlung dieser Ordnung der R^+-Baum herangezogen (Inter-Super-Tile-Clustering). Das hat den Vorteil, dass weder beim Exportieren, noch beim späteren Retrieval, zusätzliche Berechnungen notwendig sind, um eine spezielle Linearisierungsordnung zu erhalten. Alternativ wurde die Möglichkeit beschrieben, eine Linearisierungsordnung durch die Verwendung von raumfüllenden Kurven zu generieren. Gerade bei höherdimensionalen Datenräumen, erhalten raumfüllende Kurven, wie zum Beispiel die Z-Kurve oder die Hilbert-Kurve, die räumliche Nachbarschaft besser.

Im Gegensatz zu den im Kapitel 2.3 beschriebenen Systemen, wird bei HEAVEN durch die Kombination von Intra- und Inter-Super-Tile-Clustering, ein weitaus besseres Clustering der Datenobjekte auf TS-Medien erreicht. Bei herkömmlichen Systemen wird auf Clustering

entweder ganz verzichtet oder nur entlang einer fest eingestellten Dimension geclustert. Auch bei der Erhöhung der Zugriffsgranularität, wird bei HEAVEN durch eSTAR die räumliche Nachbarschaft genutzt. Im Gegensatz dazu, bilden Konkurrenzsysteme oft Dateifamilien aus beliebigen Speicherobjekten, ohne zusätzliche Kriterien zu beachten.

3.7 Retrieval von multidimensionalen Daten

Bevor das Retrieval von multidimensionalen Daten in Zusammenhang mit HEAVEN und tertiären Speichermedien erläutert wird, erfolgt zunächst eine grundlegende Beschreibung der Anfragebearbeitung bei der originalen Version von RasDaMan. Anschließend werden Möglichkeiten und Besonderheiten des Query-Scheduling in Zusammenhang mit tertiären Speichermedien betrachtet. Dabei wird nicht auf die Ermittlung der für eine Anfrage relevanten Kacheln eingegangen, sondern vielmehr auf die Beschleunigung der Ladeaufträge selbst. Die eigentliche Bestimmung der relevanten Kacheln in RasDaMan erfolgt über einen R^+-Baum. Eine detaillierte Beschreibung ist bei [Furt99] zu finden.

3.7.1 Anfragebearbeitung bei RasDaMan

Zur Verdeutlichung der Anfragebearbeitung bei der originalen Version von RasDaMan werden zunächst zwei alternative Vorgehensweisen betrachtet: Abbildung 3.26 zeigt hierzu zwei unterschiedliche Möglichkeiten, wie das Laden der Objekte aus dem DBMS und die Bearbeitung durch die CPU erfolgen kann. Der Einfachheit halber sind hier die Lade- und Berechnungsoperationen von gleicher Dauer. In der Realität überwiegt meist die Dauer der Berechnungen um Größenordnungen gegenüber den Ladeoperationen.

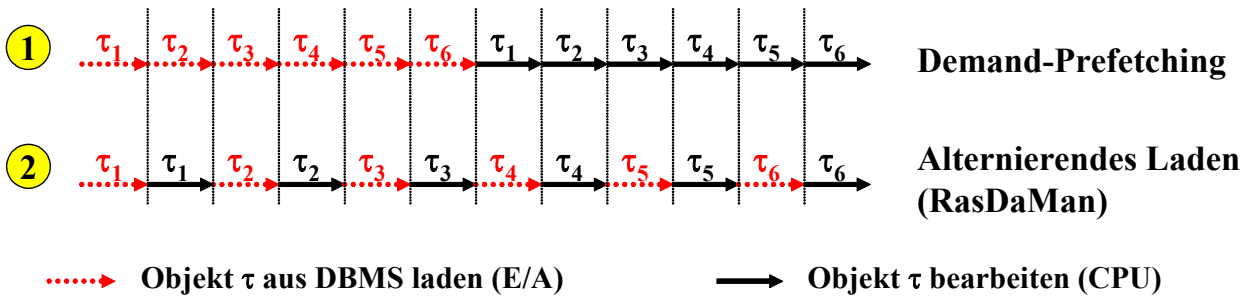


Abbildung 3.26: Lade- und Bearbeitungsalternativen von multidimensionalen Objekten τ.

Variante 1 zeigt das bedarfsgetriebene, vollständige Vorladen (*Demand-Prefetching*[33]) der Kacheln τ_1 bis τ_6 aus dem DBMS. Erst nach der Initialisierung der kompletten Datenbasis beginnt die Evaluierung der Objekte durch die CPU. Dem gegenüber steht die Variante 2 mit

[33] Der hier geprägte Begriff des Demand-Prefetching bezeichnet das bedarfsgetriebene Vorladen von Datenobjekten, die aufgrund der Anfrage im Vorfeld bekannt sind. Im Gegensatz dazu, wird beim reinen Prefetching das Vorladen von Datenobjekten, die in naher Zukunft mit hoher Wahrscheinlichkeit referenziert werden, auf „gut Glück“ versucht.

dem alternierenden Laden der Objekte aus der Datenbank. Sobald die Kachel τ_1 geladen wurde, erfolgen auf diesem Objekt die notwendigen Berechnungen durch die CPU. Nach Beendigung der Evaluierung wird das nächste Objekt geladen und alternierend fortgefahren. Dieses verschränkte Laden und Evaluieren der Objekte wurde in RasDaMan realisiert. Obwohl die beiden Varianten in Bezug auf die Ausführungsdauer auf den ersten Blick (Abbildung 3.26) gleichwertig erscheinen, bietet die Variante 2 bei genauerer Betrachtung gravierende Vorteile.

Das alternierende Laden bringt Vorteile, falls in der Anfrage Quantoren (Allquantoren oder Existenzquantoren) (2.5.5.1) enthalten sind. Ein Quantor erlaubt einen frühzeitigen Abbruch (Early Termination), sobald das Resultat der Operation feststeht. Beispielhaft können durch einen Existenzquantor alle Zellen eines MDD α untersucht werden, ob ein Zellwert einen höheren Temperaturwert als 23° Celsius aufweist. Wird die erste Zelle im MDD α mit einem höheren Temperaturwert gefunden, so evaluiert der Existenzquantor zu „wahr“ und das Resultat dieser Operation steht fest. Die restlichen Zellen müssen nicht mehr weiter betrachtet werden. Vergleichen wir nun mit diesen Voraussetzungen die beiden Varianten aus Abbildung 3.26. Wir nehmen an, dass eine Zelle der Kachel τ_2 einen höheren Temperaturwert aufweist. Somit kann die Evaluierung des MDD α frühzeitig abgebrochen werden. Hier zeigt sich klar der Vorteil der alternierenden Vorgehensweise, da bei Variante 2 neben der Einsparung von Berechnungsoperationen durch die CPU auch Ladeoperationen eingespart werden können. Die Kachel τ_3 muss nicht mehr geladen werden. Bei Variante 1 müssen nach wie vor alle Kacheln geladen werden, bevor der Existenzquantor ein Resultat liefern kann. Hier können lediglich Berechnungsoperationen der CPU eingespart werden.

Obwohl RasDaMan ein alternierendes Laden von Kacheln unterstützt, können bei einem vorzeitigen Abbruch durch einen All-, bzw. Existenzquantor, aufgrund des auf MDD basierenden Iteratorkonzeptes, keine Ladeoperationen eingespart werden. Dieser Zusammenhang wird in Abbildung 3.27 am Beispiel eines Teiloperatorbaumes von RasDaMan, anhand des Existenzquantors *some* erläutert. Dabei soll wieder das MDD α auf das Vorhandensein von Zellwerte überprüft werden, die einen Temperaturwert von 23° Celsius übersteigen.

Bei der Bearbeitung der Teilanfrage aus Abbildung 3.27 geht RasDaMan wie folgt vor: Entsprechend der Philosophie von RasDaMan, gehen die einzelnen Datenobjekte (Kacheln) so spät als möglich vom persistenten Zustand in den transienten Zustand über. Das bedeutet, Kacheln werden erst für die Bearbeitung aus der Datenbank in den Hauptspeicher geladen, wenn es unbedingt notwendig ist. Somit erfolgt der Datenzugriff erst im Vergleichsoperator Größer (>).Wie in Variante 2 aus Abbildung 3.26, wird zunächst die erste Kachel τ_1 aus der Datenbank geladen. Danach wird auf jeder Zelle dieser Kachel der Vergleich > 23 angestellt. Das Ergebnis des Vergleichsoperators, ist ein neues Objekt τ’ der gleichen Domäne aus entsprechenden Boolean-Werten. Da allerdings zwischen den Operatoren ein auf MDD basiertes Iteratorkonzept besteht, kann die evaluierte Kachel τ’ nicht sofort an den Existenzquantor *some* weitergeleitet werden. Es muss vielmehr vom Vergleichsoperator das gesamte MDD α’ evaluiert werden, bevor dieses MDD an den Existenzquantor übergeben werden kann. Das

wiederum bedeutet, dass trotz alternierendem Laden, alle Kacheln des MDD α geladen und durch den Vergleichsoperator in ein MDD α' überführt werden müssen. Erst jetzt folgt die Überprüfung, ob eine Zelle des MDD α' einen Wahr-Wert enthält. Auch wenn die erste untersuchte Zelle des MDD α einen Temperaturwert von über 23° Celsius enthält, kann noch kein vorzeitiger Abbruch erfolgen. Ein kachelbasiertes Iteratorkonzept würde diesen Missstand beseitigen. In diesem Fall wird die erste Kachel τ_1 aus der Datenbank geladen, mit dem Vergleichsoperator überprüft und die Kachel τ_1' sofort an den Existenzquantor weitergeleitet. Stellt sich dort heraus, dass ein Zellwert der Kachel τ_1' einen Wahr-Wert enthält, kann die weitere Bearbeitung vorzeitig abgebrochen werden. Durch diesen frühzeitigen Abbruch greift das alternierende Verfahren zum Laden von Kacheln. So können Ladeoperationen eingespart werden. Außerdem ist die Vergleichsoperation (> 23) nicht mehr auf dem kompletten MDD α durchzuführen. Allerdings stellt das kachelbasierte Iteratorkonzept nicht die Lösung aller Probleme dar. Neben einer sehr zeitaufwändigen Umstellung in RasDaMan, wirken sich die um ein Vielfaches gestiegenen Methodenaufrufe negativ auf die Bearbeitungszeit aus.

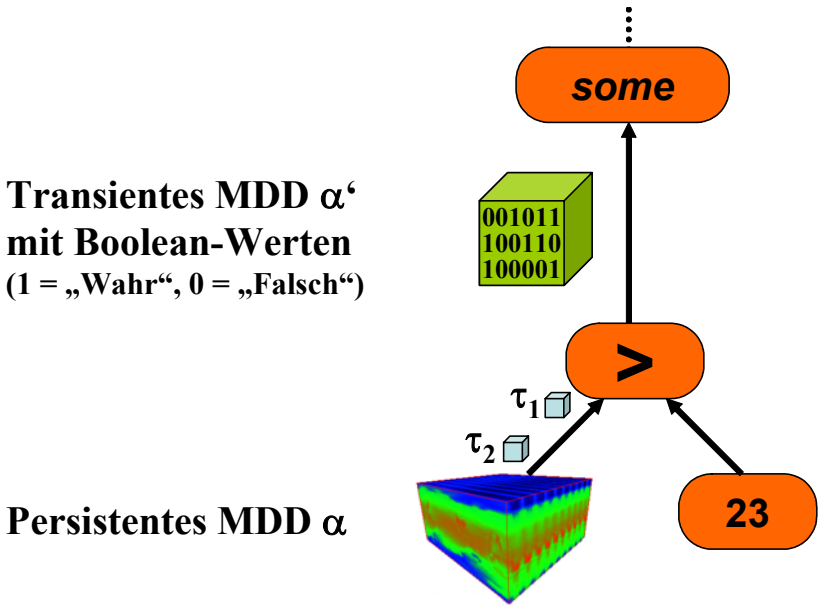


Abbildung 3.27: Ausschnitt eines RasDaMan-Operatorbaumes mit Existenzquantor *some*.

Neben dem frühzeitigen Abbruch bei Quantoren, ist es durch eine alternierende Vorgehensweise beim Laden von Datenobjekten auch möglich, ressourcenschonend mit dem vorhandenen Hauptspeicher (inklusive Swap-Speicher[34]) umzugehen. So ist es teilweise möglich, bereits bearbeitete Objekte, früher aus dem Hauptspeicher zu entfernen, um Platz für noch zu ladende Datenobjekte zu schaffen. Das vermeidet ein frühzeitiges Auslagern von Teilbereichen des Hauptspeichers in den langsameren Swap-Speicher. Außerdem wird die bestehende Maximalgrenze des Hauptspeichers später erreicht. Ein solches ressourcenschonendes Vorgehen, ist besonders im HPC-Bereich mit den dort vorhandenen großen Speichervolumen der multidimensionalen Daten wichtig. Als Beispiel hierfür können Aggregatoperationen angege-

[34] Hintergrundspeicher auf der Festplatte, um Teile des Hauptspeichers bei Bedarf auslagern zu können.

ben werden: So ist es bei der Berechnung der Durchschnittstemperatur eines MDD α möglich, bereits berechnete Kacheln aus dem Hauptspeicher zu verdrängen und nur die aggregierten Skalarwerte vorzuhalten. Solche Verdrängungstechniken können ebenso beim kompletten Vorladen von Datenobjekten angestrebt werden. Bei der alternierenden Vorgehensweise ist allerdings eine frühere Verdrängung möglich.

Das beschriebene ressourcenschonende Vorgehen ist allerdings in der momentanen RasDaMan-Version nicht realisiert. Hier werden die MDDs erst nach Beendigung der Anfragebearbeitung aus dem Hauptspeicher entfernt. Weiterhin ist klar zu sagen, dass ein Hauptproblem von RasDaMan das Speichermanagement in diesem Zusammenhang ist. Reicht für eine Bearbeitung der Hauptspeicher (inkl. Swap-Speicher) nicht aus, verfällt der RasDaMan-Server in einen undefinierten Zustand und wird sofort beendet. So verbleiben zum Beispiel die beiden MDD α und α' aus Abbildung 3.27 während der gesamten Anfragebearbeitung im Hauptspeicher: Auch wenn diese für eine weitere Berechnung nicht mehr benötigt werden. Eine große Verbesserung, kann durch eine vorzeitige Verdrängung von nicht mehr benötigten Objekten aus dem Hauptspeicher erfolgen. Dadurch wird bei einer Anfrage die Grenze des Hauptspeichers später erreicht. Zur Beseitigung des Problems des Speicherüberlaufes müsste weiterhin das Zwischenspeichern, bzw. Rausschreiben von transienten Objekten auf die Festplatte ermöglicht werden.

Zusammenfassend lässt sich sagen, dass obwohl bei RasDaMan ein alternierendes Vorgehen beim Laden von Daten realisiert wurde, die dadurch möglichen Vorteile nicht ausgeschöpft werden. So können durch das auf MDD basierende Iteratorkonzept keine Ladeoperationen eingespart werden. Bezogen auf die Dauer der Ladeoperationen gleichen sich beide Varianten. Nun folgen Besonderheiten und Randbedingungen, die bei einem Datenretrieval von tertiären Speichermedien beachtet werden müssen.

3.7.2 Anfragebearbeitung bei HEAVEN

Aus der Sicht des Anwenders, erfolgt das Retrieval von multidimensionalen Daten innerhalb HEAVEN ebenso, wie in der ursprünglichen Version von RasDaMan, ohne Anbindung von tertiären Speichersystemen. Es lassen sich alle in RasDaMan zur Verfügung stehenden Operationen ohne Einschränkung ausführen. Für den Anwender ist der Retrievalvorgang transparent. Als einziger Unterschied gilt die teilweise höhere Antwortzeit bei Anfragen, wenn sich die gewünschten Daten auf TS-Medien befinden. Das liegt daran, dass im Unterschied zum Exportieren von Daten auf TS-Medien, die Aktionen von RasDaMan und dem HSM-System zumindest teilweise gekoppelt sind. Abbildung 3.28 zeigt die möglichen Zustandsübergänge für das Retrieval von MDD. Aus Gründen einer besseren Übersichtlichkeit, wurde der HSM-Zugriff (HSM *access*(σ), schraffiert dargestellt) im unteren Zustandsdiagramm weiter aufgesplittet.

Da alle von RasDaMan unterstützten Operationen auch mit HEAVEN ausgeführt werden können, beschränken wir uns bei der Beschreibung des Datenretrievals auf das Laden der not-

wendigen Kacheln, die dem angefragten Datenbereich entsprechen. Eine Kachel entspricht der kleinsten Zugriffsgranularität. Dadurch wird es gegebenenfalls bei der Bearbeitung notwendig, im Primärspeicher nachzufiltern, falls eine Kachel nicht vollständig Gegenstand des Anfragebereiches ist. Nähere Information über die möglichen Operationen sind in Kapitel 2.5.5 oder bei [Rasd02, Rits99, Baum99, Baum04] zu finden.

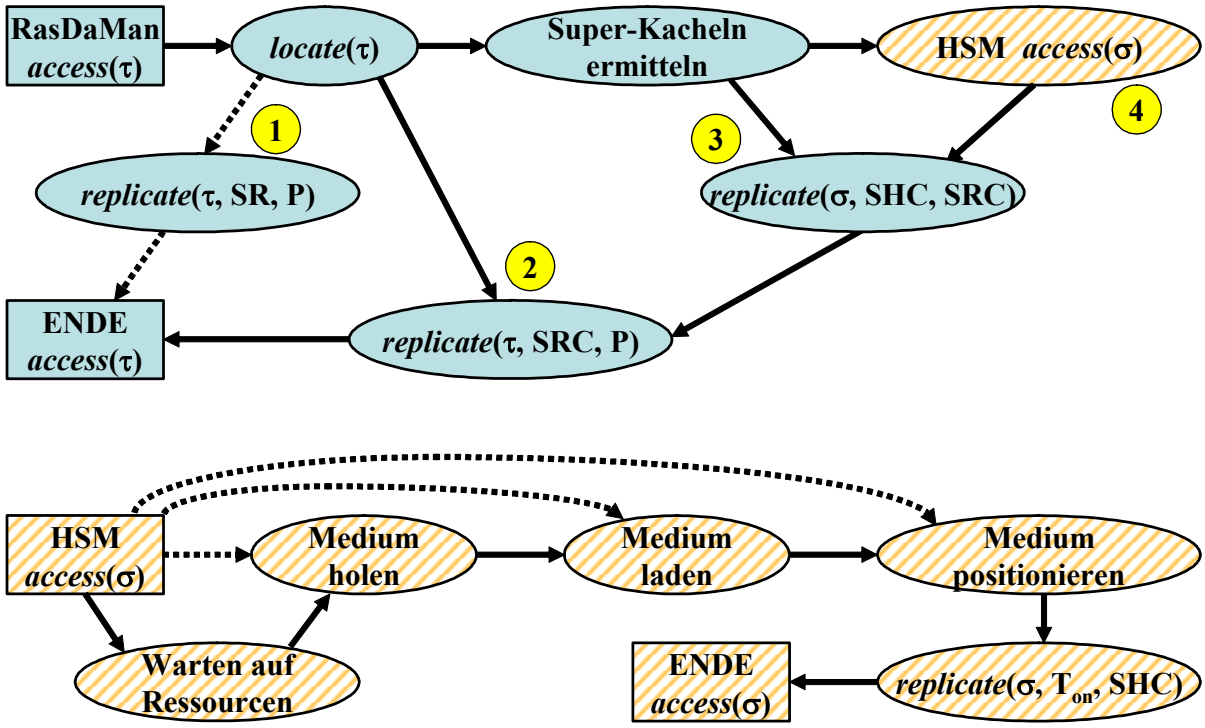


Abbildung 3.28: Zustandsübergangsdiagramme für das Retrieval von Daten.

Bei einem Zugriff auf eine Kachel *access*(τ), wird zunächst mit *locate*(τ), die in der Speicherhierarchie am höchsten gelegene Speicherebene[35] ermittelt, welche das Objekt τ enthält. Wie in Abbildung 3.28 (oben) ersichtlich, gibt es unterschiedliche Möglichkeiten. Wurden die Daten nicht auf TS-Medien exportiert, so wird mit *replicate*(τ, SR, P) die Kachel $\tau \in \Theta_{SR}$ aus dem Sekundärspeicher SR des von RasDaMan genutzten konventionellen DBMS in den Primärspeicher P geladen (Variante 1). Danach ist der Zugriff beendet und die gewünschten Oparationen können ausgeführt werden.
Eine weitere Möglichkeit (Variante 2) mit etwa gleich schneller Zugriffszeit, wird erreicht, wenn das Objekt auf TS-Medium exportiert wurde und im TS-Cache-Bereich SRC des konventionellen DBMS von RasDaMan vorgehalten wird. Mit *replicate*(τ, SRC, P) wird die Kachel $\tau \in \Theta_{SRC}$ in den Primärspeicher P geladen und der Zugriff ist beendet.
Ist die Kachel τ im Cache-Bereich des HSM-Systems SHC gespeichert (Variante 3), so ist zunächst die entsprechende Super-Kachel $\sigma \in \Theta_{SHC}$ zu ermitteln. Das bedeutet allerdings

[35] Je weiter oben sich eine Speicherebene in der Speicherhierarchie (Abbildung 2.4) befindet, desto schneller ist der Zugriff auf diesen Speicherbereich.

keinen weiteren Indexzugriff, da die Information bereits bei der Lokalisierung der Kachel mit ausgelesen wird. Somit hat die Lokalisierung einer Kachel und einer Super-Kachel eine Zeit- und E/A-Komplexität von $O(\log_m N)$, was der Höhe des R^+-Baumes entspricht. N ist die Anzahl der Knoten und m die Anzahl der Einträge je Indexknoten. Mit *replicate*(σ, SHC, SRC) wird die Super-Kachel vom HSM-Cache geladen, die enthaltenen Kacheln extrahiert und in den Speicherbereich SRC repliziert. Nachträglich ist das gewünschte Objekt τ mit der Funktion *replicate*(τ, SRC, P) in den Primärspeicher zu kopieren.
Bei der letzten Möglichkeit (Variante 4) befindet sich das angeforderte Objekt τ auf einem tertiären Speichermedium $v(\tau)$ im Speicherbereich T_{on}, T_{near} oder T_{off}, was den zeitaufwändigsten Fall beschreibt. Im Gegensatz zum Exportieren von Daten ist beim Datenretrieval keine Entkopplung zwischen RasDaMan und dem HSM-System vorhanden. Das bedeutet, RasDaMan ist angewiesen zu warten, bis das gewünschte Objekt $\tau \in \Theta_\tau(v)$ durch die Funktion *access*(σ) von einem TS-Medium geladen wurde. Die Zustandsübergänge für den Datenzugriff bei einem HSM-System (Abbildung 3.28, unten, schraffiert) gleichen in weiten Teilen den Zustandsübergängen beim Datenexport (Abbildung 3.21, unten). Es wurde nur eine *move*-Funktion durch die Funktion *replicate*(σ, T_{on}, SHC) ersetzt. Die vorhandenen Zeitverzögerungen durch das Warten auf Ressourcen und den Operationen zum Holen (o_{gv}), Laden (o_{lv}) und Positionieren (o_{pv}) eines TS-Mediums, sind in gleicher Weise vorhanden.

Besonderheiten beim Laden von TS-Daten
Zur Verdeutlichung der Besonderheiten bei der Anfragebearbeitung in Bezug auf das Laden von Daten, die sich auf tertiären Speichermedien befinden, werden drei Möglichkeiten der Eignung untersucht. Neben den beiden bereits im vorherigen Kapitel 3.7.1 beschriebenen Alternativen, dem bedarfsgetriebenen, vollständigen Vorladen von Daten, bzw. dem alternierenden Laden von Daten, wird auch eine parallele Ladestrategie untersucht.

Zunächst wird die aus der Sicht der Anfragebearbeitung bessere Variante, das alternierende Laden von Daten, betrachtet (Variante 2, Abbildung 3.26). Sind bei diesen Ladeoperationen TS-Zugriffe notwendig, so kann es durch das alternierende Vorgehen zu großen Performanceeinbußen kommen. Das ist begründet durch die nicht vorhandenen, exklusiven Rechte für eine Schreib-/Lesestation. Das hat zur Folge, dass bei starker Frequentierung des HSM-Systems, ein Medienwechsel stattfindet. Solche Medienwechsel werden besonders durch die Wartezeiten im Ladeverhalten, die während der Berechnungsphase anfallen, wahrscheinlicher (Variante 2, Abbildung 3.26). Diese Berechnungsphasen können bei komplexen Operationen entsprechend lange dauern. Aufgrund dieser Verzögerungen, kann auch durch verwendete Techniken, wie dem trägen Auswerfen (Lazy Eject) von gerade verwendeten TS-Medien, nicht sichergestellt werden, dass kein Medienwechsel stattfindet. Unter Umständen, ist die Verzögerung bis zum Laden des nächsten Datenelements zu lange und ein konkurrierender Auftrag erhält das Laufwerk. Das verursacht einen sehr zeitaufwändigen Medienwechsel (bis ca. 410 Sekunden). Deshalb ist aus der Sicht der Ladevorgänge von TS-Medien, eine alternierende Bearbeitung nicht zu empfehlen. Hier ist ein bedarfsgetriebenes, vollständiges Vorladen (Demand-Prefetching) der Daten von TS-Medien besser geeignet. Da in diesem Fall kein anderer Ladeauftrag einen Medienwechsel verursacht. Das entspricht der Variante 1 aus der

Abbildung 3.26. Auch der kontinuierliche Lesestrom auf dem TS-Medium wirkt sich positiv auf die Performance aus. Die vollständige Initialisierung der kompletten Datenbasis vor der Evaluierung der Objekte durch die CPU hat große Vorteile bei der Bearbeitung von Objektmengen. Optimierungen bei Anfragen auf Objektmengen folgen in Kapitel 3.7.3.1.

Wie im vorherigen Kapitel beschrieben, ist das vollständige Vorladen (Demand-Prefetching) von Daten, gegenüber dem alternierenden Laden, nur bei Anfragen die Quantoren beinhalten benachteiligt. Da bei RasDaMan ein vorzeitiger Abbruch (Early Termination) bei Quantoren keine Ladeoperationen einspart, hat das vollständige Vorladen von Daten keine negativen Auswirkungen auf die Bearbeitungszeit. Könnten in Zukunft, wie theoretisch bei Quantoren möglich, Ladeoperationen bei RasDaMan eingespart werden, so können durch eine einfache Analyse des Operatorbaumes, Quantoren erkannt und das Ladeverhalten von Demand-Prefetching auf alternierendes Laden automatisch umgestellt werden. Das ist ohne viel Aufwand zu implementieren.
Bei beiden Varianten kann ein ressourcenschonender Umgang mit dem Hauptspeicher realisiert werden. Der Vorsprung des alternierenden Ladens, durch die Möglichkeit einer früheren Verdrängung, ist als nicht so gravierend zu betrachten. Aus diesen Gründen wurde entschieden, bei Ladeoperationen auf TS-Medien, das vollständige Vorladen zu realisieren. Dabei ist anzumerken, dass hiervon nur das Laden von dem Speicherbereich T_{on} in den TS-Cache-Bereich des von RasDaMan verwendeten konventionellen DBMS (SRC[36]) betroffen ist. Der Ladevorgang vom SRC in den Primärspeicher P für die Bearbeitung der Datenobjekte durch die CPU, erfolgt nach wie vor bei RasDaMan durch das alternierende Laden.

Neben den beiden untersuchten Verfahren 1 und 2 ist ebenfalls ein paralleles Laden der Datenobjekte von TS-Medien möglich. Die Abbildung 3.29 zeigt zwei mögliche Varianten (3a und 3b) des parallelen Ladens. Der komplette Ladevorgang von TS-Daten ist in drei Teilbereiche untergliedert: Wir unterscheiden das Laden der Datenobjekte von einer tertiären Speicherebene T_{on} zum Cache-Bereich des HSM-Systems SHC (rot gestrichelte Pfeile). Sowie das Laden von Datenobjekten aus dem HSM-Cache SHC in den TS-Cache-Bereich von RasDaMan SRC (blaue eng gestrichelte Pfeile) und dem Laden der Datenobjekte aus dem SRC in den Primärspeicher (grün gepunktete Pfeile). Das anschließende Bearbeiten der Datenobjekte durch die CPU ist als schwarzer Pfeil dargestellt. Die Länge der einzelnen Pfeile zeigt nicht die in der Realität auftretende Laufzeit an.

Die Variante 3a zeigt das parallele Laden von Datenobjekten vom Speicherbereich T_{on} in den Cache-Bereich SHC des HSM-Systems, während Datenobjekte vom SHC über den SRC in den Primärspeicher geladen und durch die CPU bearbeitet werden. Nachdem die erste Super-Kachel σ_1 in den Speicherbereich SHC geladen wurde, beginnt parallel zum Laden der zweiten Super-Kachel σ_2 in den SHC, das Laden der Super-Kachel σ_1 vom SHC in den SRC und weiter in den Primärspeicher (P). Danach können Berechnungen mit der Super-Kachel σ_1 angestellt werden. Betrachtet man die Variante 3a der Abbildung 3.29, fällt auf, dass das

[36] Vergleiche hierzu die in Abbildung 3.2 auf Seite 71 dargestellte Systemarchitektur von HEAVEN.

Laden der Super-Kacheln von einem TS-Medium in den SHC (Variante 3a, oberer Strang) vollständig, ohne Unterbrechung geschieht. Die weiteren Ladevorgänge und die Berechnung (Variante 3a, unterer Strang) werden alternierend durchgeführt. Das bereitet auf dieser Ebene keine Probleme, da in den Zeiträumen der Berechnung, keine tertiären Ressourcen (Schreib-/Lesestation) von andern Ladeaufträgen enteignet werden können. Dieses Vorgehen wurde mit der Eigenentwicklung des Tape-Controller-Toolkit[37] (TCT) realisiert. Bei der Anbindung eines konventionellen HSM-Systems kann diese Technik nur eingesetzt werden, wenn es ein nicht blockierendes Vorladen (Demand-Prefetching) der Daten von TS-Medien in den Cache-Bereich des HSM-Systems erlaubt. Allerdings ist die Möglichkeit des Vorladens nicht bei allen HSM-Systemen gegeben, bzw. die Schnittstellen hierzu variieren. Das widerspricht allerdings der grundlegenden Forderung aus dem ESTEDI-Projekt, eine einfache und unkomplizierte Anbindung für eine Vielzahl von konventionellen HSM-Systemen zu realisieren. Deshalb wurde im Rahmen von *HEAVEN* auf diese Möglichkeit bei konventionellen HSM-Systemen verzichtet und der Zugriff über einen blockierenden Ladeauftrag durch das Dateisystem realisiert. Das bedeutet, eine Parallelisierung nach der beschriebenen Art ist so nicht möglich. Allerdings konnte die Funktionsweise und das Potential für ein optimiertes Laden von TS-Daten durch TCT evaluiert und bewiesen werden.

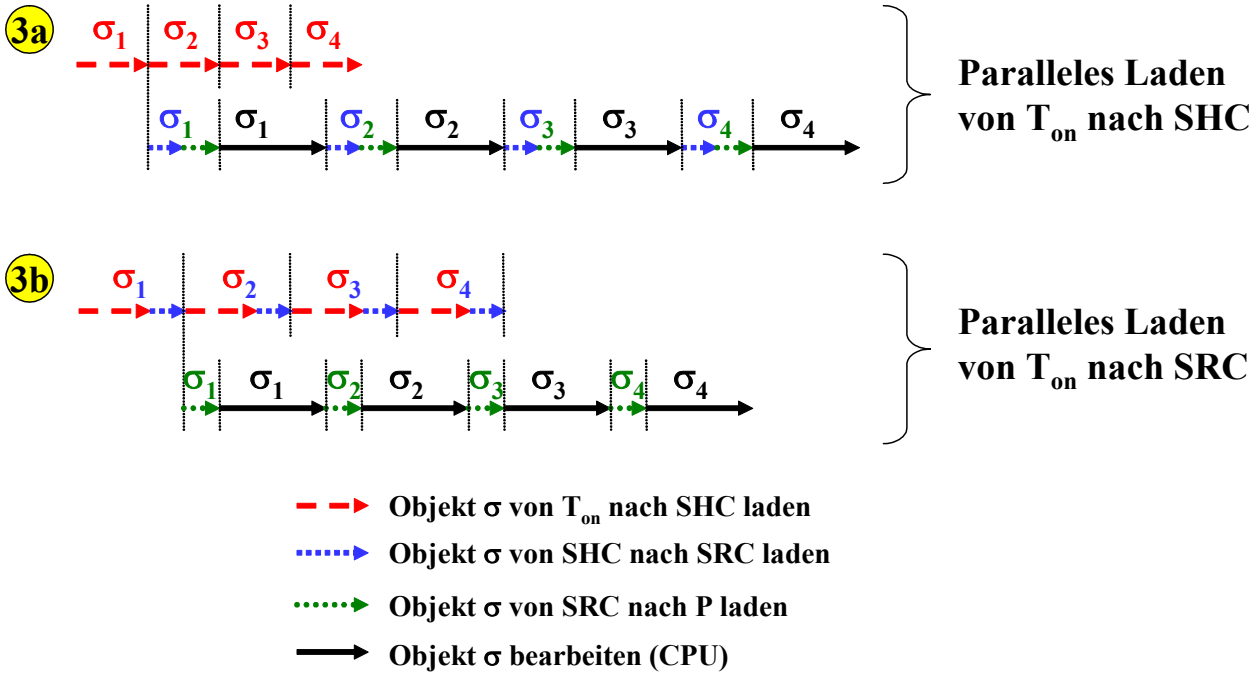


Abbildung 3.29: Möglichkeiten des parallelen Ladens der Daten von TS-Medien.

Eine weitere Beschleunigung der Anfragebearbeitung könnte durch die Variante 3b erreicht werden (Abbildung 3.29). Hier erfolgt das parallele Laden von Datenobjekten vom Speicher-

[37] Nachbildung eines HSM-Systems, mit dem beliebige Bandlaufwerke an *HEAVEN* angebunden werden können (Kapitel 3.3).

bereich T_{on} direkt in den TS-Cache-Bereich SRC von RasDaMan, während Datenobjekte vom SRC in den Primärspeicher geladen und durch die CPU bearbeitet werden. Der untere Strang der Variante 3b entspricht dem Verhalten von RasDaMan in der originalen Version ohne TS-Anbindung. Hier erfolgt das alternierende Laden der für eine Anfrage benötigten Datenelemente von dem verwendeten konventionellen DBMS in den Primärspeicher zur anschließenden Bearbeitung. Diese „originale“ Bearbeitung kann erfolgen, sobald die erste Super-Kachel in den Speicherbereich SRC geladen wurde. Im direkten Vergleich zur Variante 3a ist diese Möglichkeit sehr viel komplexer zu realisieren. Das Problem bereitet der im oberen Strang enthaltene Ladevorgang vom SHC-Bereich in den SRC-Bereich von RasDaMan (blaue eng gestrichelte Pfeile). Da dieser Ladevorgang, ebenso wie das Laden der Daten von SRC in den Primärspeicher, bzw. das Bearbeiten durch die CPU von einem RasDaMan-Server ausgeführt wird, ist eine Parallelisierung des Servers notwendig. Das steht allerdings im Konflikt mit der bei [Hahn05] realisierten Inter-, bzw. Intra-Objekt-Parallelisierung. Es wird das Laden und Bearbeiten mehrerer Objekte oder Teilobjekte nebenläufig ausgeführt. Dadurch findet eine weitere Beschleunigung der Anfragebearbeitung statt.

Zum Abschluss der theoretischen Betrachtungen zeigt Abbildung 3.30 einen Vergleich der beschriebenen Möglichkeiten bei einer Bereichsanfrage ohne, bzw. mit ausgeführten Operationen. Das zu ladende Datenvolumen der Anfrage beträgt 2,64 GByte. Die Messungen wurden mit dem Fünffach-Wechsler DLT-LXLS der Firma Overland Data mit einem integrierten Quantum DLT4000 Laufwerk[38] ausgeführt.

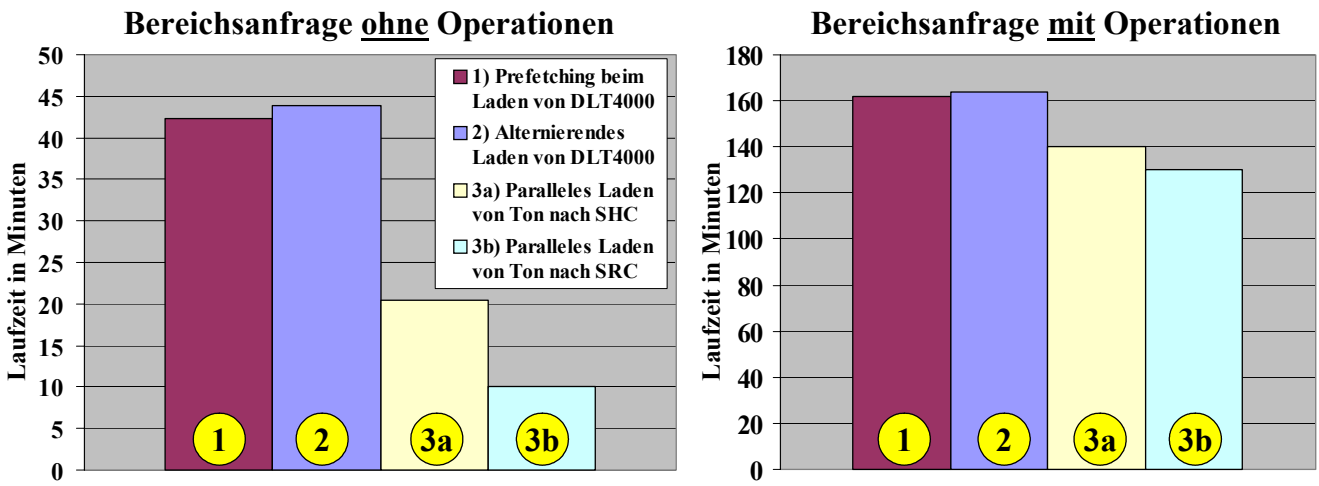


Abbildung 3.30: Vergleich der beschriebenen Möglichkeiten der Anfragebearbeitung.

Zunächst werden die vier Varianten bei einer Bereichsanfrage ohne Berechnungsoperationen verglichen (Abbildung 3.30, links). Es gehen nur die Ladeoperationen in die Messung der Laufzeit mit ein. Wie zu erwarten, zeigt das komplette Demand-Prefetching der Daten (Vari-

[38] Transferrate 1,5 MByte/s. Moderne Bandtechnologien erreichen Übertragungsraten von über 30 MByte/s (z.B. Super-DLT 600).

ante 1) gegenüber der alternierenden Vorgehensweise (Variante 2) aufgrund eines kontinuierlicheren Lesestroms eine etwas bessere Performance. In dieser Messung wurde der Variante 2 allerdings nicht durch einen konkurrierenden Auftrag die Schreib-/Lesestation $s \in S$ entzogen. In diesem Fall wäre die Variante 2 der eindeutige Verlierer. Die beiden parallelen Varianten 3a und 3b weisen eine deutlich bessere Performance auf. Nach einer initialen Ladeoperation der ersten Super-Kachel von einem tertiären Speichermedium, reicht es bei den weiteren Ladeoperationen aus, vom Sekundärspeicher zu lesen. Vorausgesetzt das parallele Vorladen der angeforderten Super-Kachel von T_{on} in den Speicherbereich SHC wurde bereits abgeschlossen. Bei Variante 3a sind die Datenobjekte vom Cache-Bereich des HSM-Systems zu laden. Dieser Ladevorgang entfällt bei Variante 3b, was eine weitere Beschleunigung mit sich bringt. Bis auf das initiale Lesen der ersten Super-Kachel entspricht die Variante 3b der Zugriffsperformance der originalen RasDaMan-Version.

Bei einer Anfrage mit Berechnungsoperationen, zeigen die einzelnen Varianten geringere Laufzeitunterschiede (Abbildung 3.30, rechts). Dies ist durch einen reduzierten Anteil der reinen Ladeoperationen gegenüber der Gesamtlaufzeit, inklusive der Berechnungszeit, zu erklären. Die Verwendung moderner Bandlaufsysteme zeigt insbesondere durch den beschleunigten Datentransfer (z.B. Super-DLT mit 30 MByte/s) eine bessere Performance. Zeiten für die Positionierung bleiben in etwa gleich. Das hat zur Folge, dass die teuren Ladeoperationen von TS-Medien nicht mehr so stark ins Gewicht fallen und sich die Balken der unterschiedlichen Varianten weiter angleichen. Nach der Beschreibung unterschiedlicher Strategien des Ladeverhaltens von HEAVEN, bzw. RasDaMan, werden nun mögliche Optimierungen für direkte Zugriffe auf TS-Medien besprochen.

Optimierung von TS-Zugriffen

Sind bei einer Anfrage, Zugriffe auf TS-Medien beteiligt, kann nicht auf die traditionelle Art und Weise verfahren werden, da das zu einer katastrophalen Performance der Anfragebearbeitung führt. Es müssen vielmehr neue Konzepte eingesetzt werden, um das zu verhindern. Nun stellt sich die Frage, wie sich die zeitaufwändigen Zugriffe auf TS-Medien reduzieren lassen. Im Idealfall werden die angeforderten Daten bereits in einer hohen Speicherebene vorgehalten: Wie zum Beispiel dem TS-Cache-Bereich SRC des von RasDaMan genutzten konventionellen DBMS (Variante 2) oder im Cache-Bereich SHC des HSM-Systems (Variante 3). Diese beiden optimalen Varianten werden im Kapitel 3.9 näher betrachtet. Hier sollen vielmehr performancesteigernde Möglichkeiten behandelt werden, die das direkte Datenretrieval von TS-Medien betreffen. Dabei werden die in den vorherigen Kapiteln aufgeführten Techniken, wie das Super-Kachel-Konzept und das Inter- und Intra-Super-Tile-Clustering, genutzt und weiter ergänzt, um die Zugriffszeiten zu optimieren. Wie die Zustandsübergänge in Abbildung 3.28 (unten) zeigen, setzen sich die Retrievalkosten bei einem Zugriff auf TS-Medien zusammen aus Anlaufkosten (Warten auf freie Ressourcen, Medienwechseloperationen $O_{sv} = \{O_{rv}, O_{ev}, O_{dv}, O_{lv}, O_{gv}\}$), variablen Positionierungskosten (O_{pv}) und den Übertragungskosten der angefragten Datenobjekte. Aufgrund der guten Übertragungsraten von TS-Systemen und den damit verbundenen geringen Übertragungskosten, fallen besonders die Anlaufkosten und die variablen Positionierungskosten ins Gewicht.

Die einzige Möglichkeit, Anlaufkosten zu minimieren, ist die Bestrebung, die Anzahl der zufälligen Datenzugriffe auf unterschiedliche Medien zu reduzieren. Durch das Speichern logisch zusammengehöriger oder gleichartiger Objekte auf dem gleichen Medium, können häufige Medienwechsel vermieden werden. Das liegt daran, dass bei Anfragen überwiegend gleichartige Objekte gemeinsam angefasst, bzw. miteinander verglichen werden. Die Wahrscheinlichkeit, dass innerhalb einer Anfrage, Operationen auf unterschiedlichen Objekttypen ausgeführt werden, ist vielfach geringer. Das bedeutet, dass wie im vorherigen Kapitel beschrieben, MDDs einer Kollektion, wenn möglich, auf einem Medium untergebracht werden. Dadurch sind die MDDs einzelner Kollektionen, entsprechend ihrer Zugriffsgewohnheit geclustert. Medienwechseloperationen werden minimiert. Eine weitere Strategie, die bei TS-Systemen mit mehreren Schreib-/Lesestationen ($s \in S$) Anwendung findet, ist die Bestrebung, ein Magnetband nach einem Zugriff möglichst lange in der Station zu belassen. Es wird von einer erhöhten Wahrscheinlichkeit ausgegangen, dass in naher Zukunft weitere Aufträge das gleiche Medium betreffen. Ist dies der Fall, entfallen Kosten für den Medienwechsel. Dieses Verfahren wird auch als „Lazy Eject“ (träges Auswerfen) bezeichnet. Die Verweildauer eines Mediums in einer Schreib-/Lesestation, hängt stark von der Anzahl N_S der vorhandenen Schreib-/Lesestationen ($s \in S$) und deren Frequentierung ab. Einstellungen und Änderungen bezüglich des trägen Auswerfens von TS-Medien sind in den Konfigurationsdateien der TS-Systeme vorzunehmen.

Einen weiteren positiven Effekt bringt bei TS-Zugriffen die zusätzlich eingeführte größere Zugriffsgranularität der Super-Kacheln. Dadurch wird die Anzahl der Zugriffe auf TS-Medien reduziert. Anstelle jede Kachel einzeln zu Laden, wird durch die zu Super-Kacheln zusammengefassten, räumlich benachbarten Kacheln, nur noch eine geringere Anzahl an Zugriffen benötigt. Nebenbei wird durch die eingeführten Kachel-Container sichergestellt, dass die Kacheln einer Super-Kachel nicht auf mehrere TS-Medien verteilt wurden und auch nicht auf einem Magnetband weit verstreut vorliegen. Das wird durch das entwickelte Intra-Super-Tile-Clustering erreicht. Somit werden durch die eingeführte, höhere Datengranularität Operationen zum Medienwechsel o_{sv} und zur Positionierung o_{pv} eingespart.

Durch die Realisierung des Inter-Super-Tile-Clustering in Kombination mit dem Intra-Super-Tile-Clustering, wurden die Voraussetzungen geschaffen, teuere Operationen zur Positionierung o_{pv} erheblich zu reduzieren (Abbildung 3.11). Das wird erreicht durch das Ausnutzen der Eigenschaften multidimensionaler Daten in Bezug auf die räumliche Nähe von Datenzellen. Diese räumlichen Nachbarschaften d-dimensionaler Daten, werden durch eine Linearisierungsordnung auf ein eindimensionales Speichermedium, das Magnetband, reduziert und so weit wie möglich beibehalten. Das ist eine ideale Voraussetzung für Bereichsanfragen, welche typischen Anfragemuster entsprechen. Selbstverständlich ist es bei einer Anfrage notwendig, die auf TS-Medien gespeicherten Daten in der gleichen Reihenfolge anzufordern, wie sie exportiert wurden. Das erfordert eine Planung der eigentlichen Anfrageausführung, um zufällige Zugriffe auf TS-Medien zu reduzieren. Allgemeine Betrachtungen der Ablaufplanung von Anfragen (*Query-Scheduling*) hinsichtlich HEAVEN werden im Folgenden beschrieben.

3.7.3 Query-Scheduling bei HEAVEN

Allgemein gesehen steht Query-Scheduling für die zeitliche Ablaufplanung von Anfragen. Je nach Anforderung werden verschiedene, teilweise konkurrierende, Ziele verfolgt. Das sind zum Beispiel die Antwortzeit, die Fairness, die Effizienz, der Durchsatz und die Auslastung der Betriebsmittel. Um die Akzeptanz des Nutzers gegenüber dem System zu verbessern, ist die Reduzierung der Antwortzeit ein Hauptziel. Bei HEAVEN liegt daher das Hauptaugenmerk auf der Optimierung der Ladevorgänge (E/A-Scheduling) von Objekten, die sich auf TS-Medien befinden. Die realisierten Techniken optimieren dabei das Ladeverhalten der Objekte beim Übergang aus den Speicherbereichen T_{on}, T_{near}, bzw. T_{off} in den Sekundärspeicher des TS-Cache-Bereiches des konventionellen DBMS von RasDaMan (Speicherbereich SRC). Die Materialisierung der Datenobjekte vom Sekundärspeicher (SRC) in den Hauptspeicher (P) erfolgt in gleicher Weise, wie das Laden der Datenobjekte aus dem konventionellen DBMS (Speicherbereich SR) der herkömmlichen RasDaMan-Version. Bevor die genannten Arten des Query-Scheduling genauer untersucht werden, folgt eine Beschreibung des allgemeinen Scheduling-Problems in HEAVEN, hinsichtlich des Datenretrievals von tertiären Speichermedien:

> Gegeben ist eine Menge an TS-Medien V, eine Menge an Schreib-/Lesestationen S, eine Menge an unabhängigen Roboterarmen R, eine Menge an parallel arbeitenden RasDaMan-Servern und eine Menge an Anfragen Q, die wiederum eine Menge an Datenfragmenten I_τ benötigen. Dabei können sich die einzelnen Datenfragmente auf beliebigen Speicherebenen (SR, SRC, SHC, T_{on}, T_{near} oder T_{off}) befinden. Aufgabe: Finde eine optimale Ausführungsreihenfolge für die Menge der Anfragen Q mit maximalem Gesamtdurchsatz und minimaler Antwortzeit für die Einzelanfragen q.

Wie bei vielen Aufgaben der kombinatorischen Optimierung, liegt die Findung einer optimalen Lösung dieses Problems nach [PAAS96] in der Komplexitätsklasse NP-vollständig[39]. Probleme dieser Klasse haben die unangenehme Eigenschaft, dass die Rechenzeit bei zunehmender Datenmenge rasch ins astronomische steigt. Eine vollständige und optimale Lösung ist daher bestenfalls für kleine Datenmengen zu erreichen. Um zumindest eine nahezu optimierte Lösung zu erhalten, kann auf Heuristiken basierte Algorithmen oder approximative Algorithmen zurückgegriffen werden. In den folgenden Ausführungen werden Algorithmen, basierend auf Heuristiken und Erfahrungen aus dem Projekt ESTEDI vorgestellt, die eine gute und effiziente Lösung des Problems darstellen. Zunächst kann das Problem des Query-Scheduling in drei Teilbereiche gegliedert werden. Dabei werden das *Intra-Query-Scheduling*, das *Inter-Query-Scheduling* und die Anfrageverschränkung (*Query Interleaving*) unterschieden.

---

[39] Der Begriff NP stammt aus der Komplexitätstheorie. Er bezeichnet eine Klasse von Problemen, die von einer nichtdeterministischen Turingmaschine in polynomialer Zeit entschieden werden können. Viele Probleme, die in der Komplexitätsklasse NP liegen, insbesondere die NP-vollständigen, lassen sich vermutlich nicht effizient lösen, da sie exponentiellen Rechenaufwand erfordern.

3.7.3.1 Intra-Query-Scheduling

Um eine Optimierung des Datenretrievals von tertiären Speichermedien zu erreichen, kann zunächst eine Anfrage auf die notwendigen Ladevorgänge der relevanten Daten reduziert werden. Eine zeitliche Ablaufplanung innerhalb einer Anfrage (Intra-Query-Scheduling), gewährleistet somit eine optimierte Ausführungsreihenfolge der Ladeoperationen, durch Sortierung der Anfragewarteschlange. So ist eine geschickte Vorsortierung der Warteschlange der zu ladenden Kacheln (bzw. Super-Kacheln), innerhalb einer Anfrage, ein wesentlicher Bestandteil des Intra-Query-Scheduling. Da bei einer gegeben Anfrage eines Nutzers eine genaue Analyse der notwendigen Ladeaufträge möglich ist, kann die Abarbeitung der Warteschlange als Stapelverarbeitung (Batch Processing) angesehen werden. Das liegt an der Tatsache, dass während der Abarbeitung der Warteschlange keine weiteren Ladeaufträge hinzukommen. Eine Anfrage ist in sich abgeschlossen. Allerdings ist es möglich, dass sich die Anzahl der zu ladenden Datenobjekte einer Warteschlange während der Abarbeitung der Anfrage reduzieren. Diese Einschränkung gilt, falls eine Selektionsbedingung einen All-, bzw. Existenzquantor enthält. Evaluiert dieser Quantor frühzeitig zu „falsch“, bzw. „wahr“, ist eine weitere Auswertung nicht mehr notwendig, da die Selektionsbedingung bereits erfüllt ist. Es erfolgt ein vorzeitiger Abbruch (Early Termination), wodurch theoretisch teure Ladeoperationen von TS-Medien eingespart werden können. Als positiven Nebeneffekt ergibt sich möglicherweise eine erheblich beschleunigte Anfragebearbeitung. Wie bereits in Kapitel 3.7.1 und 3.7.2 beschrieben, ist dies in der aktuellen Implementierung von RasDaMan nicht möglich. Eine notwendige Reorganisation der Anfragewarteschlange wäre allerdings leicht zu bewerkstelligen. Es sind lediglich Ladeaufträge zu entfernen. Die zu Beginn ermittelte Reihenfolge der Ladeaufträge innerhalb der Warteschlange bleibt erhalten.

Gerade die Eigenschaft, dass bei Intra-Query-Scheduling die Abarbeitung der Warteschlange entsprechend einer Stapelverarbeitung durchgeführt werden kann, vereinfacht die Optimierung erheblich. Durch die Abgeschlossenheit der Ladeaufträge einer Anfrage q kann vor dem Laden des ersten Objektes $\sigma \in I_q$ die Warteschlange in geeigneter Weise sortiert werden. Die einmal bestimmte Reihenfolge ist während der gesamten Anfragebearbeitung statisch[40] und wird nicht durch äußere Einwirkungen verändert. Auch eine Unterbrechung der Abarbeitung durch bevorrechtigte Anfragen findet bei RasDaMan nicht statt. Es handelt sich somit um ein non-preemptives Intra-Query-Scheduling. Die Warteschlange wird während der Anfragebearbeitung sequentiell abgearbeitet. Abbildung 3.31 zeigt die Auswirkungen beim Retrieval eines kompletten MDD (linke Seite) und bei einer Bereichsanfrage (rechte Seite) ohne, bzw. mit Intra-Query-Scheduling.

Zunächst wird das Datenretrieval eines kompletten MDD betrachtet. Die Super-Kacheln σ_1 bis σ_{12} wurden in diesem Beispiel in aufsteigender lexikalischer Ordnung auf ein TS-Medium exportiert. Wird beim Laden der Super-Kacheln nicht auf diese Ordnung geachtet und eine beliebig andere Reihenfolge gewählt (σ_9, σ_5, σ_1, σ_{10}, σ_6, σ_2, σ_{11}, σ_7, σ_3, σ_{12}, σ_8 und σ_4), steigen die notwendigen Operationen zum Positionieren stark an. Variante 1 der Abbildung 3.31

[40] Der vorzeitige Abbruch einer Anfrageausführung aufgrund von Existenz-Quantoren wird nicht berücksichtigt.

zeigt, dass vor jeder der zwölf Leseoperationen eine Positionierung auf dem Magnetband notwendig ist. Das ergibt insgesamt 24 Operationen und beschreibt den schlechtesten Fall (Worst Case). Die benötigte Zeit $t_{(N\sigma)}$, um eine Anzahl N_σ an Super-Kacheln σ eines MDD α von einem TS-Medium zu lesen, kann durch folgende Formel ausgedrückt werden:

$$t_{(N_\sigma)} = t_a + \sum_{i=1}^{N_\sigma} \left(t_{p_i} + \frac{\Delta_{\sigma_i}}{\Gamma} \right)$$

Als kleinste Zugriffsgranularität auf TS-Medien gilt eine Super-Kachel σ. Für jede zu lesende Super-Kachel addiert sich eine Positionierungszeit t_p und eine Transferzeit. Die Transferzeit berechnet sich aus dem Datenvolumen einer Super-Kachel Δ_σ und der Transferrate Γ des TS-Systems. Wird das Volumen Δ_σ aller Super-Kacheln eines MDD als konstant angenommen und das gesamte MDD gelesen, so gilt: $N_\sigma * \Delta_\sigma = \Delta_\alpha$. Befindet sich das entsprechende Medium noch nicht in einer Lesestation des TS-Systems, so fällt zusätzlich eine Anlaufzeit t_a an. Die Anlaufzeit beinhaltet, wie aus Definition 3.17 hervorgeht, die Operationen *Get-Volume* O_{gv}, *Load-Volume* O_{lv}, *Rewind-Volume* O_{rv}, *Eject-Volume* O_{ev}, und *Deposit-Volume* O_{dv}.

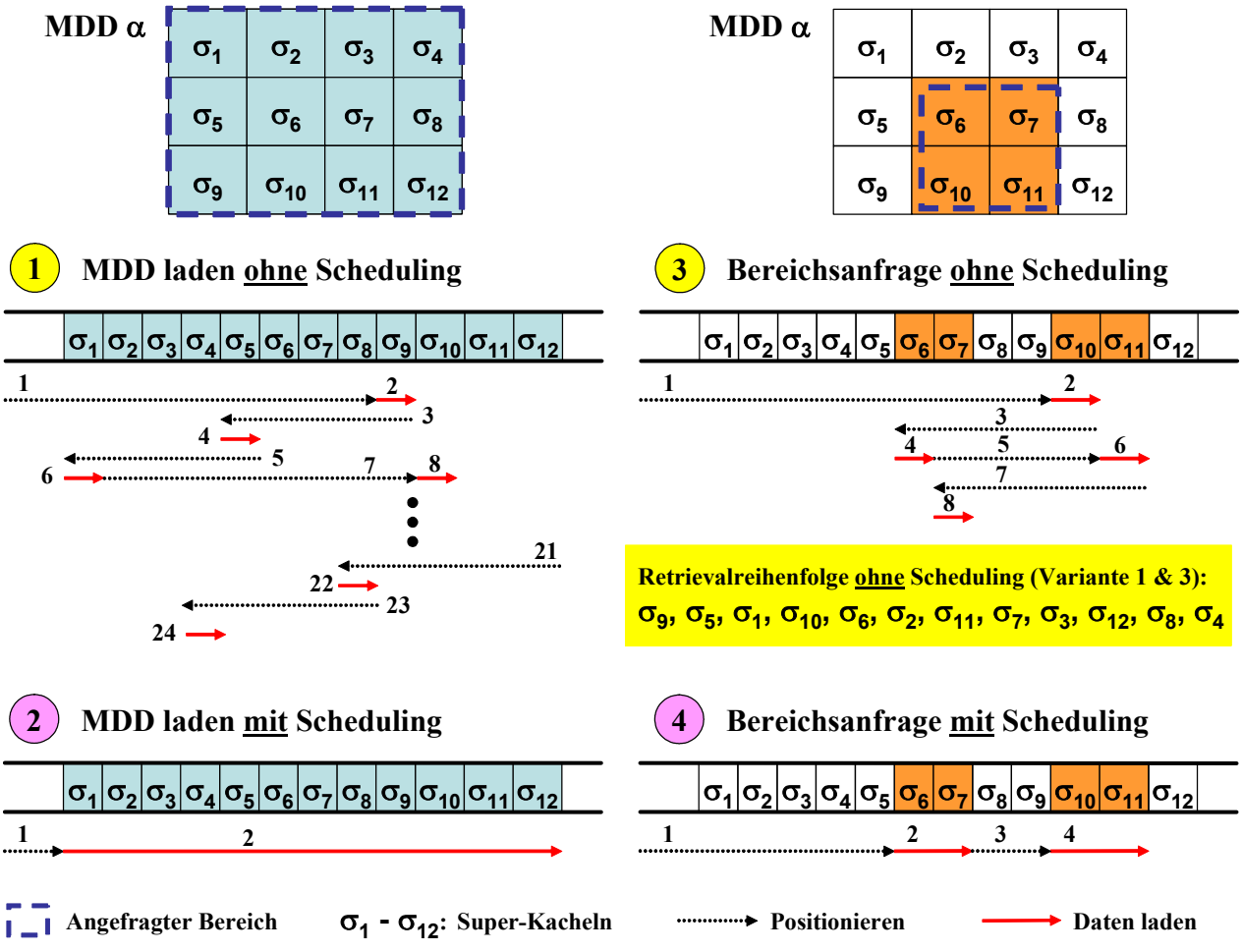


Abbildung 3.31: Komplett- und Bereichsanfrage an MDD α, ohne (Variante 1 & 3) und mit (Variante 2 & 4) Intra-Query-Scheduling.

Die Höhe der bei jedem Lesen einer Super-Kachel anfallenden Positionierungskosten, kann erheblich reduziert werden, indem die Objekte in der Ordnung angefragt werden, wie sie sich auf dem Magnetband befinden. Zugriffe werden somit entsprechend einer beim Exportieren bestimmten Ordnung serialisiert. Die Variante 2 der Abbildung 3.31 zeigt dieses Vorgehen beim Laden eines kompletten MDD α. In unserem Beispiel minimiert sich die Anzahl der Positionierungen von zwölf auf eine Operation. Es ist ebenfalls zu erkennen, dass nicht wie bei Variante 1, mehrmals über Objekte, die für die Anfrage relevant sind, hinweg gespult werden muss. Dadurch wird die auf dem Magnetband zurückgelegte Strecke reduziert, was sich neben dem Performancegewinn auch positiv auf die Lebensdauer von Magnetbändern auswirkt. TS-Medien sind nur für eine bestimmte Anzahl von Positionierungs- und Schreib-/Lese-Zyklen ausgelegt. Die Anzahl der Leseoperationen bleibt gleich, da sich die Menge der zu ladenden Super-Kacheln nicht ändert. Allerdings können die Objekte in einem gleich bleibenden linearen Lesestrom geladen werden. Das bedeutet weiteren Performancegewinn. Das TS-System muss nicht zwischen den unterschiedlichen Geschwindigkeiten für das Lesen und das Positionieren umgestellt und justiert werden. So kann das Lesen der zwölf Super-Kacheln theoretisch auch als ein Lesevorgang mit dem zwölffachen Datenvolumen angesehen werden. Für diesen Idealfall ändert sich aus theoretischer Sicht die Formel für die Berechnung der Zeitdauer des Ladevorganges einer Anzahl N_σ an Super-Kacheln σ eines MDD α von TS-Medium wie folgt:

$$t_{ideal(N_\sigma)} = t_a + t_p + \sum_{i=1}^{N_\sigma} \frac{\Delta_{\sigma_i}}{\Gamma}$$

In der Realität wird dieser Idealfall mit nur einer Positionierungsoperation nicht ganz erreicht, da kleine Bereiche zwischen den Datenobjekten „überlesen“ werden müssen. Dieses „Überlesen“ entspricht streng genommen einer Positionierungsoperation mit sehr kurzer Dauer (ms). Die wirkliche Zeitdauer berechnet sich aus diesem Grund durch die oben beschriebene Formel mit einer initialen, höheren Positionierungszeit (s) und mehreren sehr kurzen Positionierungsoparationen (ms). Die kurzen Positionierungsoperationen entstehen durch „Überlesen“ von Bandmarken: Wie zum Beispiel der Marken HDR (File Header), TM (Tape Mark) oder EOF (End of File). Da die zu speichernden Objekte auf Magnetband, ähnlich wie bei einer Festplatte, in Blöcke bestimmter Größe geschrieben werden, entstehen am Dateiende Datenblöcke, die meist nicht vollständig gefüllt sind. Diese Bereiche müssen ebenso „überlesen“ werden. Messungen, mit dem in dieser Arbeit verwendeten Fünffach-Wechsler DLT-LXLS der Firma Overland Data, haben ergeben, dass die Zeitdauer um weniger als 1 % gegenüber dem Ladevorgang eines vergleichbaren Komplettobjektes (1,4 GByte) steigt. Das gilt allerdings nur bei einer Datengranularität von Super-Kacheln mit einem Datenvolumen von mindestens 40 MByte. Wurde eine Kachelgröße von 124 KByte als Datengranularität gewählt, so steigt die Zeitdauer des Ladevorganges um mehr als 50 % gegenüber dem Laden des Komplettobjektes. Als Grund für den prozentualen Anstieg bei kleiner werdender Speicher-, bzw. Retrievalgranularität, ist die steigende Anzahl der notwendigen Operationen zum „Überlesen“ der Bereiche zwischen aufeinander folgenden Objekten zu nennen. Dabei spielen weniger die Bandmarken eine Rolle, als vielmehr die steigende Anzahl der Leseoperationen von Blöcken mit ungenutzten Speicherbereichen.

Betrachten wir Bereichsanfragen, die nicht das komplette MDD betreffen, so ist aufgrund der beim Exportieren gewählten Linearisierungsordnung nicht mehr zu gewährleisten, dass alle notwendigen Objekte durch einen einzigen linearen Lesestrom vom TS-Medium geladen werden können. Die rechte Seite der Abbildung 3.31 zeigt diesen Sachverhalt. Hier betrifft die Bereichsanfrage die Super-Kacheln σ_6, σ_7, σ_{10} und σ_{11}. Da die Super-Kacheln in lexikalischer Ordnung auf TS-Medium vorliegen, ist aufgrund des notwendigen Sprungs von Super-Kachel σ_7 auf σ_{10}, kein durchgehender linearer Lesestrom möglich. Allerdings können, wie beim Lesen eines kompletten MDD, durch eine geschickte Vorsortierung der Anfragewarteschlange, Operationen zum Positionierung eingespart und Leseoparationen kombiniert werden. Die Variante 3 der Abbildung 3.31 zeigt eine Bereichsanfrage ohne Anfrageprozessverwaltung (Query-Scheduling) im Vergleich zur Variante 4 mit Query-Scheduling. Zur Berechnung der Zeit zum Lesen des angefragten Bereiches, kann wieder die oben aufgeführte Formel herangezogen werden. Die einzelnen Positionierungszeiten können dabei sehr unterschiedlich ausfallen. Das liegt an den geringen Positionierungskosten bei den linearisierten Leseoperationen und den hohen Kosten einer initialen Positionierung, bzw. beim Überspringen von Datenobjekten (z.B. von Super-Kachel σ_7 auf σ_{10}). Die Anzahl der Positionierungsoperationen mit kurzer Dauer (ms), gegenüber den höheren Positionierungszeiten (s), hängt stark von der Form und Lage der Anfragebox (Bereichsanfrage) innerhalb des Objektes und der beim Exportieren gewählten Linearisierungsordnung ab. Abbildung 3.32 zeigt anhand einer Messung[41] den Unterschied eines zufälligen Ladens von Super-Kacheln gegenüber einer Bereichsanfrage mit Intra-Query-Scheduling. Das zu ladende Datenvolumen der Bereichsanfrage umfasst 1,32 GByte (28 Super-Kacheln mit je 48,7 MByte).

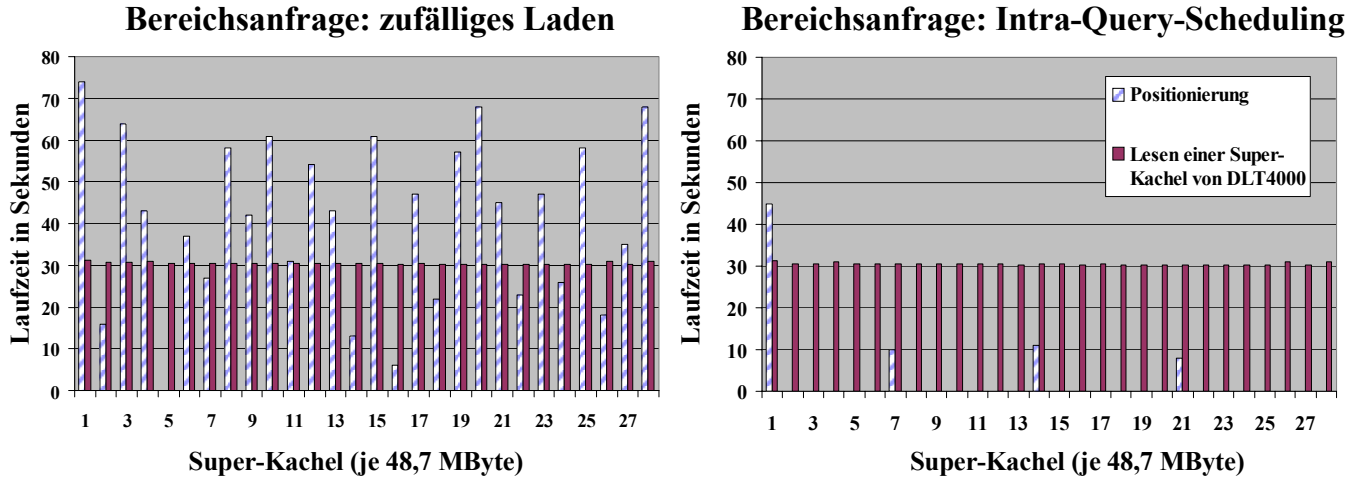


Abbildung 3.32: Bereichsanfrage mit zufälligem Laden und mit Intra-Query-Scheduling.

Die linke Seite der Abbildung 3.32 zeigt deutlich die erforderlichen Positionierungsoperationen vor dem Laden der jeweiligen Super-Kacheln. Das entspricht der Variante 3 aus Abbildung 3.31. Zum Vergleich sind in der rechten Messung nur vereinzelte Positionierungs-

[41] Durchgeführt mit einem Fünffach-Wechsler DLT-LXLS der Firma Overland Data mit einem integrierten Quantum DLT4000 Laufwerk. Die mittlere Zugriffszeit beträgt 68 Sekunden und die Transferrate 1,5 MByte/s.

operationen notwendig. Dazu gehört die nicht zu vermeidende initiale Positionierung des Lesekopfes an den Anfang des Datenbereiches (vor Super-Kachel Nr. 1). Weiterhin sind Positionierungsoperationen nur vor den Super-Kacheln σ_7, σ_{14} und σ_{21} zu erkennen. Das bedeutet, dass sich zum Beispiel zwischen den geladenen Super-Kacheln σ_6 und σ_7 weitere, für die Anfrage nicht relevante Super-Kacheln befinden, die übersprungen werden müssen. Durch die Einsparung von Positionierungsoperationen o_{pv} lassen sich bemerkenswerte Performancegewinne erzielen. In diesem Beispiel sinkt die Dauer der Anfragebearbeitung von 33,4 Minuten auf 15,5 Minuten. Das entspricht einer Performancesteigerung von über 53,3 %, bzw. einem Speed-Up von 2,14.

Optimierungen bei Anfragen auf Objektmengen

Nach der Betrachtung der Möglichkeiten zur Optimierung von Anfragen, die ein einzelnes Objekt betreffen, versprechen Anfragen auf Objektmengen großes Potential um Anfragen auf TS-Medien zu beschleunigen. Als Beispiel kann die Addition zweier multidimensionaler Objekte als eine binär induzierte Operation genannt werden (Kapitel 2.5.5). Der Basistyp und die Domänen der MDD α und β müssen bei dieser Anfrage übereinstimmen. Pro Koordinate der Domäne wird der entsprechende Zellwert aus dem ersten und zweiten Objekt geladen, die Operation angewendet und das Ergebnis in dem abgebildeten MDD gleicher Domäne abgelegt ($\alpha \times \beta \rightarrow \text{ind}\oplus(\alpha, \beta)$). Abbildung 3.33 zeigt das schrittweise, verschränkte Vorgehen von RasDaMan bei der Addition der einzelnen Zellen der beiden MDD α und β zu einem Ergebnisobjekt der gleichen Domäne.

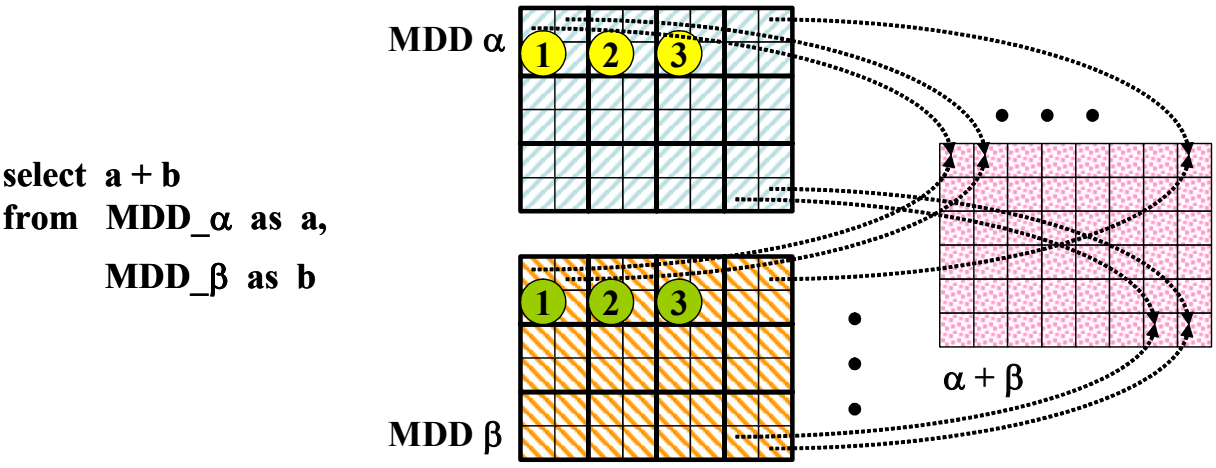


Abbildung 3.33: Addition zweier MDD als Beispiel einer binären, induzierten Operation.

Bei der Berechnung des Zielobjektes werden die beiden Quellobjekte entlang aufsteigender Dimensionen durchlaufen. Entsprechend der Koordinate des aktuell zu berechnenden Zellwertes werden bei Bedarf die Zellwerte der MDD α und β in Kachel-Granularität aus dem von RasDaMan verwendeten konventionellen DBMS in den Hauptspeicher alternierend geladen. Auf diese Art und Weise erfolgt ein kontinuierliches, schrittweises und verschränktes Laden der relevanten Kacheln der beiden MDD in den Hauptspeicher. Das ist begründet in

der grundlegenden Philosophie von RasDaMan, Objekte möglichst spät im Hauptspeicher zu materialisieren. Das bedeutet, erst wenn eine Zelle zur Bearbeitung erforderlich ist wird die zugehörige Kachel geladen. Erste Operationen können dadurch bereits nach dem Laden der ersten Kacheln der MDD α und β ausgeführt werden. Folgeoperationen hingegen müssen mit ihrer Berechnung aufgrund des MDD basierten Iteratorkonzeptes warten, bis die Vorgängeroperation ein vollständiges Ergebnis-MDD berechnet hat. Im Beispiel aus Abbildung 3.33 besteht eine Kachel aus vier Zellen. Um die erste Zelle des Zielobjektes zu berechnen, wird von den Quellobjekten α und β jeweils die Kachel Nr. 1 benötigt. Bei der Operation auf der zweiten Zelle ist kein Laden notwendig, da sich die Daten bereits im Hauptspeicher befinden. Das Berechnen des dritten Zellwertes erfordert bei beiden MDD das Laden der Kachel mit der Nr. 2 in den Hauptspeicher. Dieses Vorgehen wird fortgesetzt, bis die letzte Zelle des Zielobjektes berechnet wurde. Anzumerken ist hier, dass bei den beiden MDD auch unterschiedliche Kachelung vorhanden sein kann.

Wie bereits in Kapitel 3.7.2 beschrieben, ist das alternierende Ladeverhalten von RasDaMan bereits problematisch bei Operationen auf einzelne Objekte, die sich auf TS-Medien befinden. Bei der Ausführung von Operationen auf Objektmengen, ist die alternierende Vorgehensweise bei der Datenhaltung auf TS-Medien weitaus ineffizienter. Das erfordert aus diesem Grund eine andere Vorgehensweise. Gerade das schrittweise, verschränkte Laden von Kacheln (in Super-Kachel-Granularität) unterschiedlicher MDD, führt zu sehr vielen Positionierungsoperationen. Der Lesekopf muss auf dem Magnetband zwischen den beiden MDD oft hin und herbewegt werden. Das liegt an der beim Exportieren durch das Intra- und Inter-Super-Tile-Clustering herbeigeführten Linearisierungsordnung, mit der n-dimensionale Objekte auf die eindimensionale Struktur eines Magnetbandes abgebildet werden. Wegen der zeitaufwändigen Positionierungsoperationen, ist es besonders bei TS-Zugriffen viel versprechend, die Anfragewarteschlange auf eine geeignete Weise vorzusortieren, um mehrerer Ladeaufträge sinnvoll zu kombinieren. Es hätte wenig Sinn, aufwändige Clustering-Verfahren beim Exportieren von multidimensionalen Daten auf TS-Medien anzuwenden und beim Retrieval dieser Daten, die gewählte Ordnung zu vernachlässigen. Dieser Sachverhalt wird anhand der Abbildung 3.34 verdeutlicht. Als Beispielanfrage dient, wie bereits beschrieben, die Addition der zwei MDD α und β.

Die Variante 1 verdeutlicht das bereits in Abbildung 3.33 gezeigte, verschränkte Laden der für die Operation notwendigen Zellen. Einziger Unterschied ist die, bei TS-Systemen eingeführte, höhere Zugriffsgranularität von Super-Kacheln im Vergleich zu Kacheln. Die einzelnen Super-Kacheln der beiden MDD, wurden wie in Abbildung 3.34 ersichtlich, nacheinander in lexikalischer Ordnung auf Magnetband geschrieben. Die lexikalische Ordnung spiegelt die beim Exportieren durch eine Linearisierungsordnung festgelegt Reihenfolge wider. Um für die Beispielanfrage die erste Zelle des MDD α und die erste Zelle des MDD β zu addieren, wird zunächst die Super-Kachel σ_1 des MDD α von TS-Medium geladen. Im Anschluss findet eine Positionierung auf dem TS-Medium statt, um auch die Super-Kacheln σ_1 des MDD β laden zu können. Nach dem Lesen dieser Super-Kachel, muss erneut positioniert

werden, um wiederum die Super-Kachel σ_2 des MDD α zu laden. Dieses Vorgehen wird fortgesetzt, bis die letzte Super-Kachel σ_{12} des MDD β von dem TS-Medium gelesen wurde.

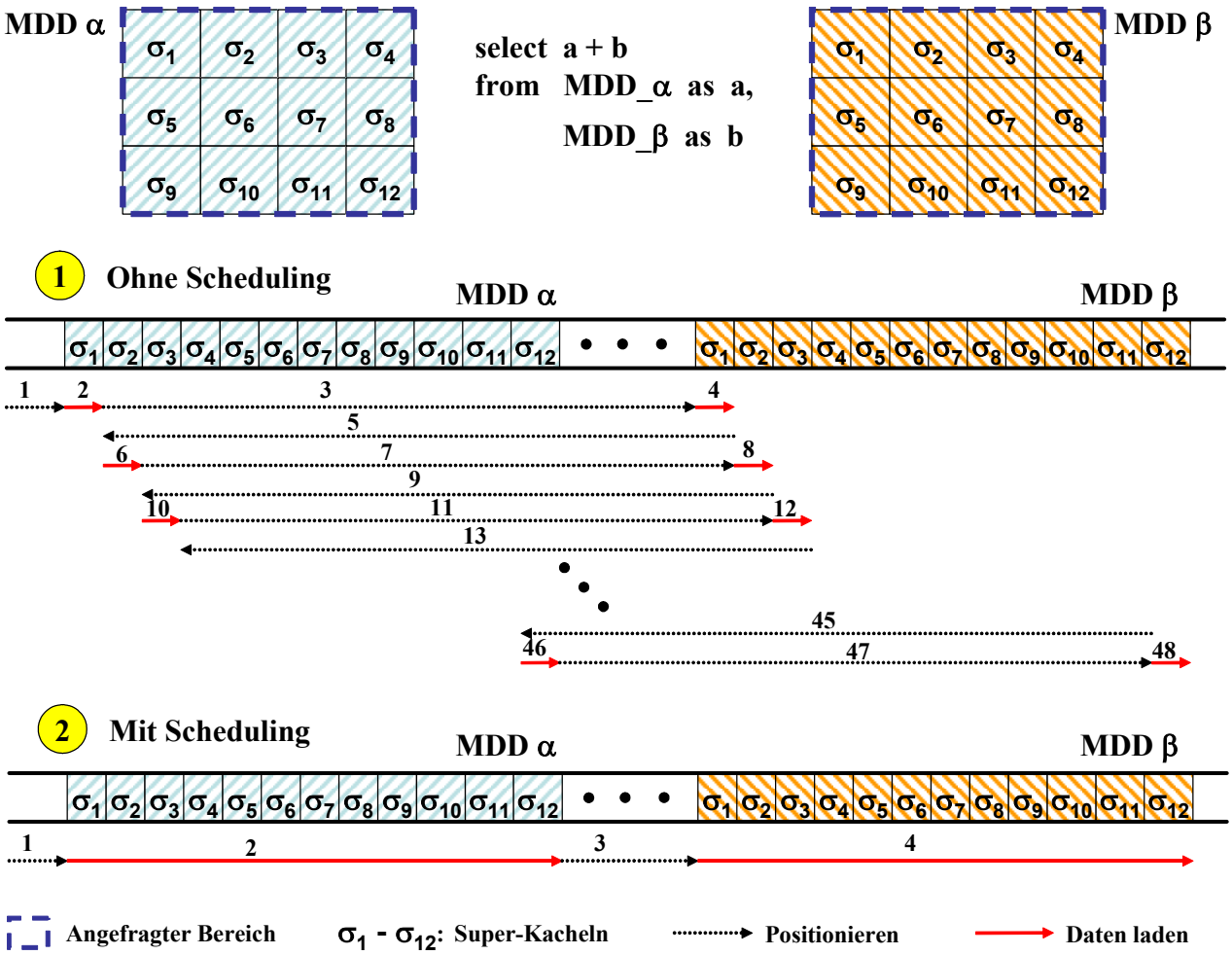


Abbildung 3.34: Anfrage über zwei MDD, ohne, bzw. mit Intra-Query-Scheduling.

Bei der unter Variante 1 beschriebenen Vorgehensweise fällt auf, dass sehr häufig über Objekte, die für die Anfrage relevant sind, hinweg gespult wird, ohne diese sofort zu laden. Die Beobachtung der Variante 1 ohne Anfrageprozessverwaltung (Query-Scheduling) führt zur Variante 2 der Abbildung 3.34 mit Inter-Query-Scheduling, das ein komplettes Vorladen (Demand-Prefetching) der beiden Objekte realisiert. Ähnlich wie bei der beschriebenen Optimierung von TS-Zugriffen mittels Query-Scheduling bei Anfragen auf ein MDD, wird versucht, unnötige Operationen zum Positionieren einzusparen. Die notwendige Information, um die Anfragewarteschlange optimal sortieren zu können, wird den Metadaten entnommen. Zu diesem Zweck wurde der Systemkatalog von RasDaMan entsprechend erweitert. Durch diese Vorgehensweise wird die beim Exportvorgang durch das Intra- und Inter-Super-Tile-Clustering realisierte Linearisierungsordnung auch beim Datenretrieval eingehalten. Wie die Variante 2 der Abbildung 3.34 beweist, reduziert sich die Vielzahl der Positionierungsoperationen erheblich. Auch das mehrfache Überspulen von einzelnen Objekten beim Datenzugriff kann verhindert werden. Anfallende Ladeoperationen werden so weit wie möglich serialisiert

und mit einem gleich bleibenden linearen Lesestrom geladen. Das beschriebene Vorladen kompletter MDD wurde durch die Anpassung der in RasDaMan implementierten alternierenden Ladetechnik bewerkstelligt. Die hier vorgestellte Variante 2 behandelt die Anfrage von kompletten MDD. Werden nur Bereiche dieser MDD, zum Beispiel mit einer binären, induzierten Operation angefragt, so sind zusätzliche Positionierungsoperationen meist nicht zu vermeiden (vergleiche Abbildung 3.28, Variante 4).

Neben der behandelten Reduzierung von Positionierungsoperationen o_{sp}, werden durch die hier gewählte Methode auch Medienwechseloperationen o_{sv} eingespart, falls sich die für eine Operation relevanten MDD auf zwei unterschiedlichen TS-Medien befinden. Die Anfragewarteschlange wird dahingehend sortiert, dass zunächst die Objekte eines TS-Mediums in Speicherordnung gebracht und von TS-Medium geladen werden. Danach folgt der Wechsel auf ein anderes TS-Medium, um die restlichen Objekte entsprechend der Exportordnung zu lesen. Dieses Vorgehen ist ebenfalls sinnvoll, wenn mehrere Schreib-/Lesestationen ($s \in S$) vorhanden sind und kein ständiger Wechsel der TS-Medien notwendig ist. Das verschränkte Lesen bedeutet auch in dieser Konstellation einen Verlust an Performance, da das TS-System weniger zusammenhängende Aufträge abhandeln kann. Weiterhin ist anzumerken, dass trotz der von TS-Systemen eingesetzten Methode des trägen Auswerfens von Medien (Lazy Eject), keine Sicherheit besteht, dass beide TS-Medien wirklich in der Lesestation verbleiben. Die Bestrebung, die häufigen Medienwechsel zu vermeiden, kann durch eine zu geringe Anzahl an Lesestationen oder einer zu frequentierten Nutzung des TS-Systems ausgehebelt werden. Nur bei einer parallelisierten Anfragebearbeitung wird versucht, eine Beschleunigung im Ladevorgang zu erreichen, indem Objekte verschränkt auf mehreren Medien gespeichert werden. Es werden bewusst parallele Hardwarestrukturen ausgenutzt. Dies entspricht dem RAIT Level 0 und wird auch als Striping bezeichnet.

Die Basisanforderung des Intra-Query-Scheduling für einzelne Objekte, bzw. Objektmengen, kann durch eine einfach zu formulierende Bedingung definiert werden: Die Anforderungsreihenfolge der auf TS-Medien gespeicherten Daten muss der beim Exportieren gewählten Linearisierungsordnung entsprechen. Die in *HEAVEN* implementierte Umsetzung dieser Bedingung minimiert die Retrievalkosten durch die Einsparung, bzw. Reduzierung von Positionierungs- und Medienwechselzeiten. Weiteres Einsparungspotential bietet das Inter-Query-Scheduling.

3.7.3.2 Inter-Query-Scheduling

Im Unterschied zum Intra-Query-Scheduling, das den Ablaufplan der Ladeaufträge innerhalb einer Anfrage übernimmt, behandelt das Inter-Query-Scheduling die Organisation mehrerer Anfragen. Durch das Inter-Query-Scheduling soll gewährleistet werden, dass eine optimierte Ausführungsreihenfolge zwischen mehreren Anfragen, durch Sortierung der Ausführungswarteschlange erreicht wird. Auch hierbei sind besondere Überlegungen notwendig, um eine performante Umsetzung auf eine tertiäre Datenhaltung zu erreichen. Wie auch bei der Realisierung des Intra-Query-Scheduling steht die Minimierung von Operationen zum Medienwechsel o_{sv}, bzw. zur Positionierung o_{pv} bei *HEAVEN* im Vordergrund. Das bedeutet, dass die

Anfragen einer Warteschlange entsprechend einer optimalen Retrievalperformance für TS-Medien sortiert werden. Diese zunächst trivial klingende Anforderung, erfordert gegenüber dem Vorgehen beim Intra-Query-Scheduling weiterführende Überlegungen.

Im Gegensatz zum Intra-Query-Scheduling ist beim Inter-Query-Scheduling keine Stapelverarbeitung der Anfragewarteschlange mehr vorauszusetzen. Das liegt an der Tatsache, dass aufgrund des dynamischen Verhaltens der Warteschlange durch neu eintreffende Anfragen, keine exakte Analyse der anstehenden Anfragen möglich ist. Gerade bei DBMS, dominiert die durch interaktiven Betrieb erzeugte Last, die nicht vorhergesagt werden kann. An die Stelle einer Planung tritt daher eine dynamische Zuteilungsstrategie, die oft von der jeweiligen Lastsituation ausgeht. Die Reihenfolge, der in der Warteschlange enthaltenen Anfragen, ist somit nicht mehr statisch. Um eine optimale Abarbeitung zu gewährleisten muss sie nach jedem Eintreffen einer weiteren Anfrage, erneut nach bestimmten Kriterien angepasst und sortiert werden. Für das hier vorgestellte Inter-Query-Scheduling, lassen sich ähnliche Entscheidungskriterien, wie bei der Prozessverwaltung eines Betriebssystems anbringen. Als entscheidender Unterschied gilt, dass Anfragen aus der Sicht von HEAVEN immer komplett (non-preemptive) ausgeführt werden. Ein bei modernen Rechnersystemen übliches klassisches Zeitscheibenverfahren, wie zum Beispiel Round Robin, wird nicht verwendet. Bei HEAVEN wird eine sehr einfache Vorgehensweise eingesetzt:

- *First-Come-First-Served* (FCFS), bzw. *First-In-First-Out* (FIFO)
  Die Anfragen werden entsprechend der Reihenfolge ihres Eintreffens bearbeitet. Vorteil: Faire Zuteilung und einfach zu realisieren, da keine Sortierung notwendig ist. Nachteil: Lange Antwortzeit für kurze Anfragen, wenn eine lange Anfrage früher in der Warteschlange eingetroffen ist. Die Verwaltungskomplexität beträgt O(1). Weiterhin ist es bei HEAVEN möglich, Anfragen zur Lastverteilung (Load Balancing) auf einen Pool an Ressourcen (RasDaMan-Server) zu verteilen. Sobald eine Ressource frei wird, erfolgt die Zuteilung einer Anfrage aus der Warteschlange.

Abbildung 3.35 zeigt das Inter-Query-Scheduling von HEAVEN. Innerhalb der RasDaMan-Architektur übernimmt der RasDaMan-Manager (RasMgr) die Verwaltung und Verteilung der eingehenden Anfragen q auf den oder die RasDaMan-Server. Zunächst werden in Abbildung 3.35 unterschiedliche Architekturvarianten vorgestellt.

Grundsätzlich ist es möglich, mehrere RasDaMan-Server gleichzeitig zu betreiben, die von einem RasMgr verwaltet werden. In Abbildung 3.35 ist ein System mit einem zentralen RasMgr und drei RasDaMan-Server (RasDaMan 1 bis 3) dargestellt. Jeder der Einzelkomponenten kann dabei auf dem gleichen, bzw. auf unterschiedlichen Rechnern betrieben werden. In diesem Beispiel befindet sich der erste RasDaMan-Server zusammen mit dem relationalen DBMS IBM/DB2 auf Rechner 1. Die Datenbasis des DBMS wird durch das angeschlossene HSM-System 1 ergänzt. Weiterhin werden zwei RasDaMan-Server parallel auf dem zweiten Rechner betrieben. Das angebundene relationale DBMS Oracle befindet sich hingegen auf dem separaten Rechner 3. Wie bereits in Kapitel 3.6 beschrieben, ist es ebenfalls möglich, gleichzeitig mehrere HSM-Systeme an einen RasDaMan-Server zu koppeln. Das zeigt der

Server RasDaMan 2, wohingegen an RasDaMan 3 kein HSM-System gekoppelt wurde. Der RasMgr wiederum, kann auf einem dieser Rechner oder wie in diesem Beispiel, auf einem eigenständigen Rechner (Rechner 4) arbeiten. Die Vielzahl der vorgestellten Architekturvarianten zeigt, dass für unterschiedlichste Anwendungsbereiche eine flexible und optimierte Lösung angestrebt werden kann.

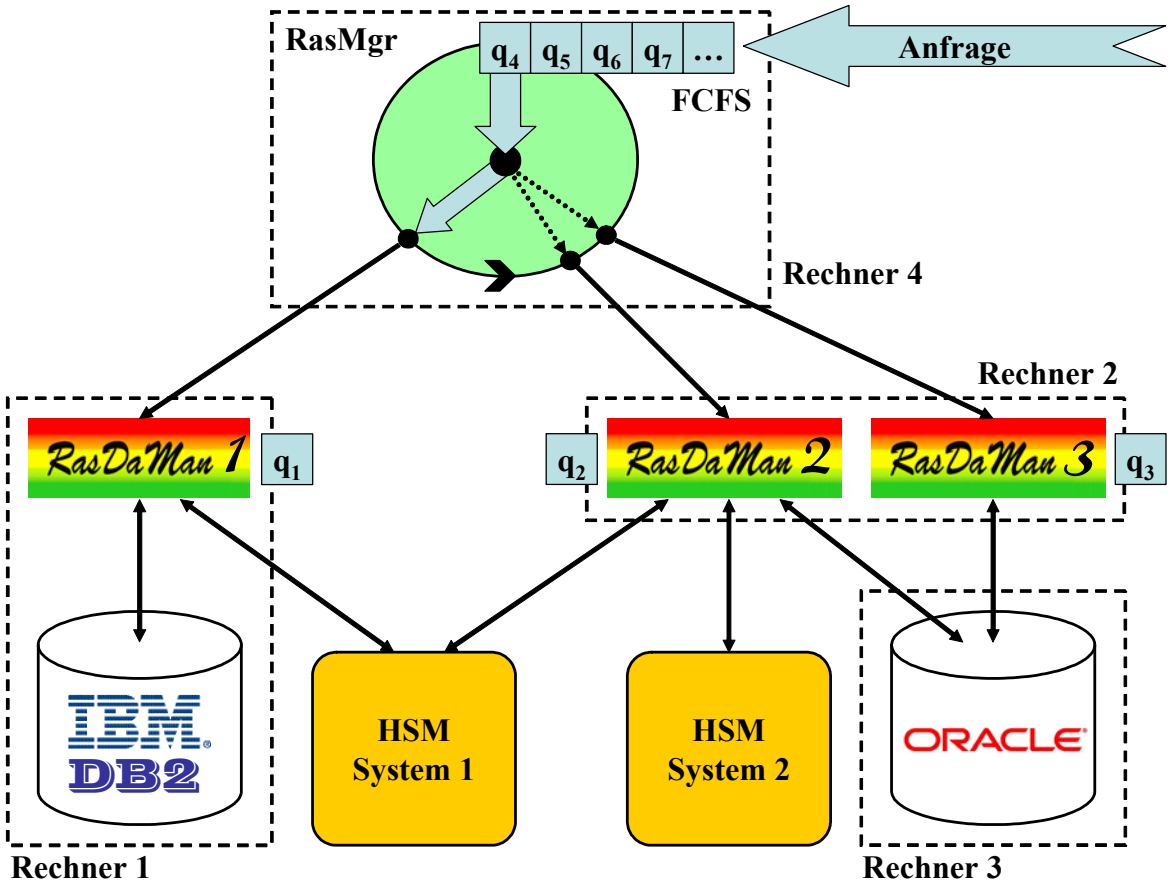


Abbildung 3.35: Inter-Query-Scheduling bei HEAVEN bei mehreren RasDaMan-Servern.

Wurde eine Architektur mit nur einem RasDaMan-Server (z.B. RasDaMan 1) gewählt, so erfolgt das Inter-Query-Scheduling nach dem FCFS-Prinzip. Sobald der Server das Ergebnis einer Anfrage ermittelt und zur Anwendung geliefert hat, schickt der RasMgr die entsprechend der Reihenfolge ihres Einganges nächste Anfrage der Warteschlange an den Server. Wurde ein System mit mehreren RasDaMan-Server installiert, so werden die Anfragen von dem RasMgr auf die einzelnen RasDaMan-Server verteilt. In Abbildung 3.35 wurden die Anfragen q_1, q_2 und q_3 bereits an die entsprechenden RasDaMan-Server geschickt. In bestimmten zeitlichen Abständen wird zyklisch überprüft, ob ein RasDaMan-Server wieder für weitere Anfragen zur Verfügung steht. Ist das der Fall, so wird an diesen Server die nächste Anfrage q_4 aus der Warteschlange geschickt.

Als entscheidende Vorteile des FCFS-Verfahrens, gelten die einfache Realisierung und die faire Abarbeitung der eingehenden Anfragen. Außerdem sind zur Verteilung, der in der Warteschlange befindlichen Anfragen, keine zusätzlichen Informationen notwendig. Komplexere

Scheduling-Algorithmen benötigen für die Sortierung der Anfragewarteschlange systemweite Informationen über den Status der angebundenen TS-Systeme, den Inhalt des Sekundärspeicher-Caches und die Menge der für eine Anfrage zu ladenden Objekte. Solche Informationen sind notwendig, um zum Beispiel prioritätsgesteuerte Algorithmen zu realisieren. Dabei werden Anfragen mit der höchsten Priorität als erstes ausgeführt. Hier besteht allerdings die Möglichkeit, dass Anfragen „verhungern“, wenn immer wieder neue Anfragen eintreffen, die eine höhere Prioritätsstufe aufweisen. Als Beispiel ist der klassische Algorithmus *Shortes-Job-First* (SJF) zu nennen. Es erhält die Anfrage mit der kürzesten (geschätzten) Laufzeit die höchste Priorität zugewiesen und wird somit zuerst ausgeführt. Als Vorteil dieses Algorithmus gilt das Liefern einer minimalen Antwortzeit und die Tatsache, dass kurze Anfragen gegenüber langen Anfragen nicht benachteiligt werden, wie es bei FCFS der Fall ist. Als nachteilig wirkt sich eine hohe Wartezeit für lange Anfragen aus, die erst nach vielen kurzen Anfragen ausgeführt wird. Wird als Datenstruktur für die Anfragen eine Vorrangwarteschlange (Priority Queue) mit N geordneten Einträgen verwendet, beträgt die Komplexität zum Einfügen und Entfernen eines Elements O(log N). Der Zugriff auf das Element mit der höchsten Priorität, kann in konstanter Zeit erfolgen [Knut98]. Das Verfahren *Highest-Response-Ratio-Next* (HRRN) stellt einen Kompromiss zwischen FCFS und SJF dar und beseitigt deren Nachteile. Dies geschieht durch die Berechnung des Response-Ratio-Wertes ((Wartezeit + Ausführungszeit) / Ausführungszeit) der einzelnen Anfragen einer Warteschlange. Die Anfrage mit dem größten Response-Ratio-Wert hat die höchste Priorität und wird als nächstes ausgeführt. Ergeben sich gleiche Response-Ratio-Werte, so wird nach dem FCFS-Prinzip verfahren. Da in die aktuelle Priorität einer Anfrage die Wartezeit einfließt, ist es bei einer Neubewertung erforderlich, diese auf jedes der N Elemente der Warteschlange anzuwenden. Das führt zu einer linearen Komplexität von O(N). Der Zugriff auf das Element mit der höchsten Priorität erfolgt in konstanter Zeit. Für die beiden Verfahren SJF und HRRN ist es notwendig, die voraussichtliche Bearbeitungsdauer zu berechnen. Das erweist sich als sehr schwierig, da vielfältige Einflüsse die Laufzeit verändern. Einflüsse sind zum Beispiel die Selektivität der Anfragen, ein vorzeitiger Abbruch durch Quantoren, die Anzahl der von TS-Medien zu ladenden Objekte und unterschiedliche Medienwechsel- und Positionierungszeiten.

Wie die Verfahren FCFS und SJF, weist auch HRRN Defizite bezüglich des Datenretrieval von TS-Medien auf. Abbildung 3.36 verdeutlicht an einem Beispiel diese Mängel bei Zugriffen auf Magnetbänder. Als Grundannahme soll gelten, dass sich die Anfragen q_1 bis q_5 zum gleichen Zeitpunkt in der Warteschlange befinden und in lexikalischer Ordnung eingetroffen sind. Die rechts neben den Anfragen aufgetragenen Balken, zeigen die jeweilige geschätzte Ausführungsdauer. Diese beinhaltet das Laden der relevanten Daten und die eigentliche Anfragebearbeitung. Für die Ausführung der Anfragen sind entsprechende Datenobjekte von den Magnetbändern v_1, v_2 und v_3 notwendig. Eine Anfrage benötigt allerdings nur Daten von einem Magnetband.

Zunächst werden die Verfahren FCFS, SJF und HRRN mit einem idealen Ladevorgang verglichen. Bei diesem Vergleich stehen nur ein RasDaMan-Server und eine Lesestation für das

Datenretrieval zur Verfügung. Untersucht werden die Anzahl der benötigten Operationen für einen Medienwechsel o_{sv} (hier inkl. initialer Positionierung) und für die Positionierung o_{pv} auf dem Magnetband $v \in V$. Bei der Variante FCFS sind drei Medienwechsel und eine Positionierungsoperation notwendig, da die nacheinander ausgeführten Anfragen q_2 und q_3 Datenobjekte des gleichen Magnetbandes v_2 anfragen. Bei den Varianten SJF und HRRN ist diese glückliche Lage nicht eingetreten und es werden jeweils vier Medienwechsel o_{sv} ausgeführt. Diese Medienwechsel, beinhalten die Positionierung bis zum ersten gewünschten Datenobjekt, das gelesen werden soll. Im Idealfall können zwei Medienwechsel durch Positionierungsoperationen ersetzt werden. Das Laden der Datenobjekte der Anfragen in der „richtigen" Reihenfolge (q_1, q_4) minimiert die Positionierungsdauer (Ende q_1 bis Anfang q_4). Bei umgekehrter Ausführungsreihenfolge (q_4, q_1), muss über eine längere Strecke (Ende q_4 bis Anfang q_1) positioniert werden. Zusammenfassend lässt sich erkennen, dass die bei der Prozessverwaltung eingesetzten Standardverfahren, keine ideale Lösung darstellen. Obwohl zusätzliche Informationen über das System in die Verfahren SJF und HRRN einfließen, wird kein Vorteil in der Retrievalperformance gegenüber der einfach zu realisierenden FCFS-Methode erreicht.

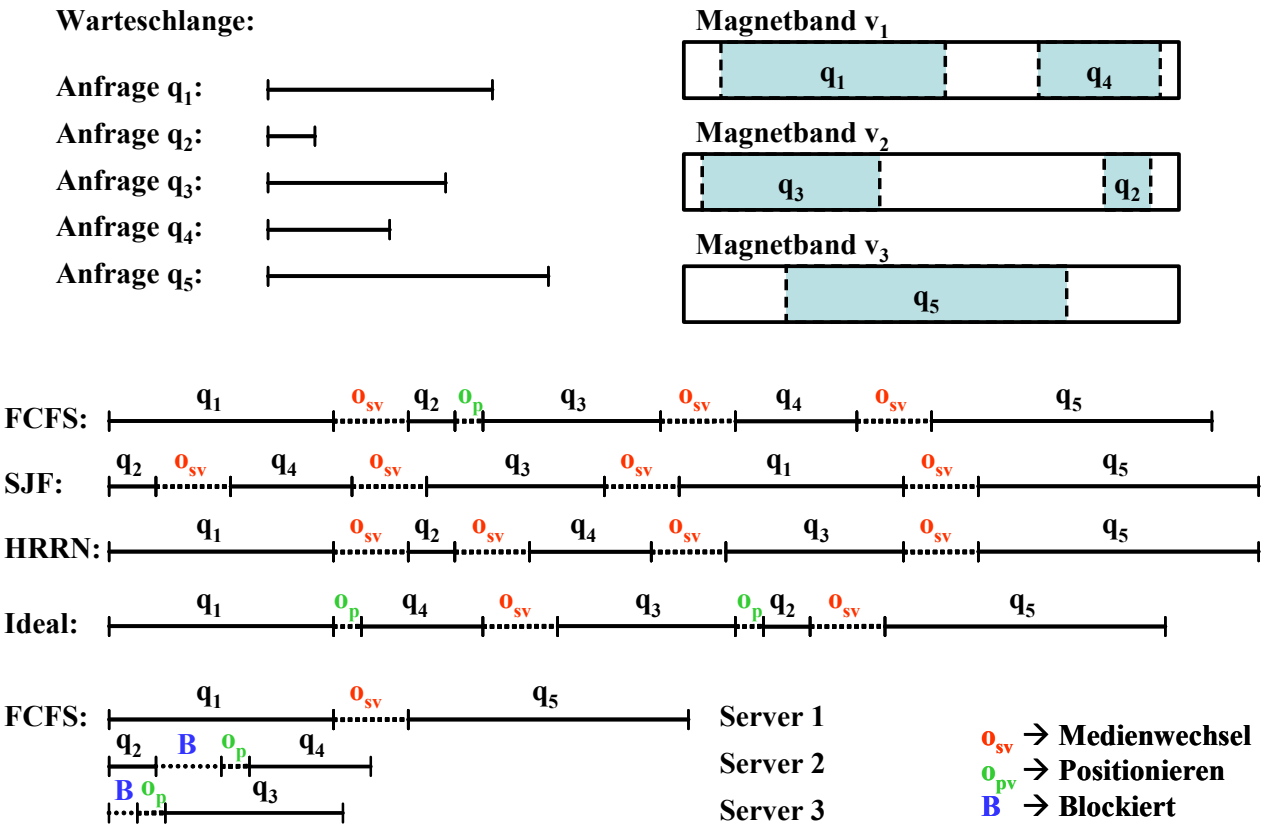


Abbildung 3.36: Untersuchung der Scheduling-Verfahren FCFS, SJF und HRRN.

Das in RasDaMan implementierte Verfahren zur Lastverteilung, mit der Zuordnung der Anfragen auf eine vorgegebene Anzahl an RasDaMan-Server, birgt gegenüber dem Einprozessorbetrieb Vorteile. Abbildung 3.36 stellt die Vorzüge anhand eines Beispiels mit drei Ras-

DaMan-Server und zwei Magnetband-Lesestationen dar. Die Anfragen q_1 bis q_5 werden in der Reihenfolge des Eintreffens in die Warteschlange (FCFS-Verfahren) an die drei RasDaMan-Server verteilt. Nach Eintreffen der Anfragen q_1 und q_2 beginnen die beiden Server 1 und 2 sofort mit den Ladeoperationen. Server 3 ist nicht sofort in der Lage, die Anfrage q_3 zu bearbeiten, da das Magnetband v_2 durch das Laden der Datenobjekte für Anfrage q_2 (von Server 2 bearbeitet) blockiert (B) ist. Erst nach dem Laden der Datenobjekte der Anfrage q_2 und einer Positionierung o_{pv} können die Datenobjekte für Anfrage q_3 gelesen werden. In diesem Beispiel setzt sich die Ausführungsdauer einer Anfrage je zur Hälfte durch das Laden der relevanten Daten und der eigentlichen Anfragebearbeitung zusammen. Daher ist die Blockierung der Lesestation durch den Server 2 bereits nach der halben Ausführungsdauer der Anfrage q_2 aufgehoben. Eine Blockierung liegt ebenfalls für den RasDaMan-Server 2 bei der Bearbeitung der Anfrage q_4 vor. In diesem Fall ist das Magnetband v_1 durch das Laden der Datenobjekte für die Anfrage q_1 belegt. Die Anfrage q_5 kann von RasDaMan-Server 1 nach einem Medienwechsel o_{sv} ohne Blockierung geladen werden. Durch die Ausnutzung nebenläufiger Ressourcen ist nur eine Medienwechseloperation o_{sv} bei der Abarbeitung der fünf Anfragen notwendig. Als negativ sind die in diesem Beispiel auftretenden Blockaden in der Anfragebearbeitung durch das Warten auf freie Ressourcen zu nennen. Allerdings ist zu erwähnen, dass sich Nutzeranfragen, aufgrund der sehr großen Anzahl an TS-Medien eines HSM-System, meist nicht nur auf wenige Medien konzentrieren. Daher treten Verzögerungen häufig nicht durch konkurrierende Anfragen, die vom gleichen Medium lesen wollen auf, sondern insgesamt durch eine zu starke Frequentierung der vorhandenen Schreib-/Lesestation.

Eine Schwierigkeit des Scheduling für eine Menge an Anfragen Q, besteht in der Tatsache, dass meist jede Anfrage $q \in Q$ unterschiedlich und der Verlauf bei der Ausführung nicht vorhersagbar ist. Soll das Ladeverhalten von Anfragen speziell für TS-Zugriffe optimiert werden, sind für das Inter-Query-Scheduling einige alternative Methoden denkbar:

1. MIN-VOLUME: Bevorzugt die Anfrage mit der geringsten Anzahl der zu ladenden Objekte, bzw. mit dem geringsten Datenvolumen, das geladen werden muss: Entspricht dem SJF-Prinzip aus der Sicht des Ladevorganges. Es wird eingesetzt, um die Datenübertragungskosten zu minimieren.
2. MAX-CACHE: Bevorzugt beim Laden die Anfrage, bei der sich die meisten Objekte bereits im Cache befinden ($\tau \in \Theta_{SRC}$). Es wird eine allgemeine Reduzierung der TS-Zugriffskosten erreicht.
3. MAX-ONLINE: Bevorzugt beim Laden die Anfrage, bei der sich die meisten Objekte auf einem Online-Medium befinden ($\tau \in \Theta_{Ton}$): Motiviert durch eine Reduzierung der Medienwechselkosten.
4. MIN-DELAY: Bevorzugt die Anfrage, welche Objekte anfordert, die sich auf einem Online-Medium befinden ($\tau \in \Theta_{Ton}$) und die geringste Zugriffsverzögerung aufweisen. Reduzierung der Medienwechsel- und Positionierungskosten, allerdings abhängig von der aktuellen Position des Schreib-/Lesekopfes.
5. MIN-OPERATION: Bevorzugt die Anfrage, für welche die geringsten Medienwechsel o_{sv} und Positionierungsoperationen o_{pv} notwendig sind.

6. MAX-MATCH: Bevorzugt die Anfrage, welche Objekte enthält, die von den meisten anderen Anfragen ebenfalls benötigt werden. Maximiert das Vorladen von Datenobjekten und steigert die Möglichkeit der parallelen, konkurrierenden Anfrageausführung.

Um diese Möglichkeiten zu realisieren, müssen Systemzustände in den Entscheidungsprozess mit einfließen. Tabelle 3.2 zeigt unterschiedliche Systeminformationen, die als Entscheidungskriterien für die oben genannten Scheduling-Methoden bereits im RasMgr für die Entscheidungsfindung benötigt werden. Tabelle 3.2 ist nach aufzuwendender Komplexität für die Ermittlung der jeweiligen Systemzustände sortiert. Einfach zu beschaffende Zustände stehen am Anfang der Tabelle.

Systemzustand:	**Benötigt für:**
Status der einzelnen RasDaMan-Server	FCFS, SJF, HRRN, MIN-VOLUME, MAX-CACHE, MAX-ONLINE, MIN-DELAY, MIN-OPERATION, MAX-MATCH
Zu ladende Objekte, bzw. zu ladendes Datenvolumen	MIN-VOLUME, MAX-CACHE, MAX-ONLINE, MIN-OPERATION, MAX-MATCH
Zu ladende Objekte, bzw. zu ladendes Datenvolumen von TS-Medien ($\tau \in \{\Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$)	MAX-CACHE, MAX-ONLINE, MIN-OPERATION
Zu ladende Objekte, bzw. zu ladendes Datenvolumen vom Cache (SRC) oder vom DBMS (SR)	MAX-CACHE, MAX-ONLINE, MIN-OPERATION
Ordnung der Datenobjekte auf dem Magnetband	MIN-DELAY, MIN-OPERATION
Geschätzte Bearbeitungsdauer der Anfrage	SJF, HRRN
Magnetbandtechnologie (Serpentin-, bzw. Schrägspurverfahren)	MIN-DELAY
Datentransferrate, Medienwechselkosten und Positionierungskosten (variabel)	SJF, HRRN, MIN-OPERATION
TS-Medien, die sich zum aktuellen Zeitpunkt in einer Schreib-/Lesestation befinden ($v \in V_{on}$)	MAX-ONLINE, MIN-DELAY, MIN-OPERATION
Gegenwärtige Kopfposition bei TS-Medien	MIN-DELAY, MIN-OPERATION
Aktuelle Bewegungsrichtung des Lesekopfes bei Serpentinenverfahren	MIN-DELAY

Tabelle 3.2: Systemzustände, als Entscheidungskriterien für diverse Scheduling-Verfahren.

Die Tabelle zeigt, dass für einige Scheduling-Algorithmen, Informationen benötigt werden, die sehr schwer zu beschaffen sind. Zu diesen Informationen zählen überwiegend die gegen-

wärtige Position des Lesekopfes und die aktuelle Bewegungsrichtung des Lesekopfes. Auch wenn vereinzelte HSM-Systeme die Möglichkeit bieten, diese speziellen Informationen über eine proprietäre Systemschnittstelle auszulesen, wird diese Möglichkeit bei HEAVEN nicht genutzt. Bei der Entwicklung von HEAVEN wurde vor allem Wert darauf gelegt, die Anbindung des multidimensionalen DBMS RasDaMan an verschiedene HSM-Systeme sehr kompatibel zu gestalten. Es wurde bewusst darauf verzichtet, eine Anbindung über proprietäre Schnittstellen der HSM-Systeme zu realisieren. Oft wird im Bereich der HSM-Systeme durch die Verwendung von proprietären Schnittstellen bewusst eine Inkompatibilität zu anderen Systemen geschaffen. Dies soll vor allem den Kunden an das eigene Produkt binden und den Umstieg auf Konkurrenzprodukte erschweren. Weiterhin zeigt sich im Folgenden, dass durch genauere Betrachtung des Einsatzgebietes und des Anwenderverhaltens, weit weniger komplexe Verfahren für das Inter-Query-Scheduling notwendig sind.

Die Erfahrungen aus dem ESTEDI-Projekt haben gezeigt, dass sich Anfragen überwiegend auf bestimmte Objektgruppen beschränken. Das bedeutet, dass normalerweise bei Anfragen keine Operationen auf beliebige Kombinationen aller in RasDaMan gespeicherter Daten ausgeführt werden. Überspitzt ausgedrückt: Es ist sehr unwahrscheinlich, dass eine Anfrage mit Operationen auf Daten einer Klimasimulation kombiniert mit einer chemischen Prozesssimulation ein brauchbares Ergebnis liefert. Bei RasDaMan besteht weiterhin die Einschränkung, dass Operationen nur auf gleichartigen Objekten, mit gleicher Domäne[42] ausgeführt werden können. Auch sind teilweise Operationen auf Objekte mit unterschiedlichem Basisdatentyp nicht sinnvoll, wie zum Beispiel die Addition von Zellen zweier MDD mit den Datentypen RGB und Float. Durch diese Tatsachen ist die Wahrscheinlichkeit relativ hoch, dass die für eine Anfrage benötigten Datenobjekte nicht auf vielen TS-Medien verstreut vorliegen. Es ist eher davon auszugehen, dass sich die Datenobjekte einer Anfrage auf nur einem, bzw. wenigen TS-Medien befinden. Dafür spricht auch die Tatsache, dass bei RasDaMan gleichartige Objekte (MDD) zu Kollektionen zusammengefasst werden. Beim Exportieren wird ferner versucht, diese Objektgruppen auf einem TS-Medium zu speichern. Aus der Sicht der Anwender können in den meisten Fällen zwei typische Zugriffsmuster unterschieden werden:

1. Anwender greifen meist auf unterschiedliche Daten zu:
   Durch die Zugriffe auf unterschiedliche Daten, sind überwiegend verschiedene TS-Medien beim Datenretrieval betroffen. Das bedeutet allerdings, dass eine Änderung der Reihenfolge der Anfragebearbeitung durch das Inter-Query-Scheduling keine Auswirkungen auf die Retrievalperformance hat. Es können keine Medienwechsel und Positionierungsoperationen eingespart werden. Als Beispiel sind hier Wissenschaftler zu nennen, die in unterschiedlichen Bereichen forschen.
2. Anwender greifen auf die gleichen Daten (Hot Spots) zu:
   Fragen mehrere Anwender die gleichen Daten an, so ist ein Inter-Query-Scheduling sinnvoll, da das meist Zugriffe auf das gleiche TS-Medium bedeutet. Die Reihenfolge der Ladeaufträge auf diesem TS-Medium ist für die Retrievalperformance ausschlag-

[42] Mit geometrischen Operationen, wie zum Beispiel Trimming und Shift, kann die Domäne angeglichen werden (Kapitel 2.5.5).

gebend. Allerdings nur solange sich die Datenobjekte noch nicht im Sekundärspeicher-Cache befinden. Ein Anwendungsszenario ist ein gerade neu entstandenes Klimamodell, das für viele Klimaforscher als Grundlage für weiterführende Analysen verwendet wird.

Ergänzend ist zu erwähnen, dass bei einer RasDaMan-Installation, die Anzahl der Anfragen eher überschaubar ist. Dafür ist die jeweilige Bearbeitungsdauer sehr hoch (bis mehrere Stunden oder Tage). Aufgrund der geringen Frequentierung des RasMgr mit Anfragen ist die Anfragewarteschlange nicht sehr lang. Das bedeutet wiederum, dass das Potential, die Retrievalperformance durch das Inter-Query-Scheduling zu steigern, als eher niedrig einzuschätzen ist. Möchte man bei steigender Anzahl der Anfragen dem Anwender eine angemessene Retrievalperformance garantieren, so ist der einzig gangbare Weg, die Anzahl der parallel arbeitenden RasDaMan-Server zu steigern. Sonst wächst die Anfragewarteschlange immer weiter an und die bestehenden RasDaMan-Server, sind aufgrund der langen Bearbeitungszeiten nicht mehr in der Lage, die Aufträge abzuarbeiten. Trotz hoher Retrievalkosten, die beim Laden von TS-Medien auftreten, ist die eigentliche Anfrageberechnung auf dem RasDaMan-Server bei komplexen, berechnungsintensiven Operationen (z.B. Aggregatoperationen und induzierte Operationen) der Engpass. Solche Anfragen ereichen eine hohe Prozessorauslastung und werden auch als *CPU-bound* bezeichnet. Dem gegenüber stehen berechnungsarme Operationen, mit einem stark dominierenden Anteil des Ladevorganges (*I/O-bound*) der Datenobjekte an der gesamten Bearbeitungsdauer der Anfrage. Berechnungsarme Operationen sind geometrische Operationen, da zur Einschränkung der Domäne eines Objektes nur Metadaten erforderlich sind und keine zusätzlichen Berechnungen auf den Datenobjekten stattfinden. Werden klassische CPU- und I/O-bound Operationen kombiniert, so dominieren die berechnungsaufwändigen Operationen, was insgesamt zu einem CPU-bound-Verhalten führt. Diese kombinierten Operationen stellen die typische Klasse an Anfragen dar.

Zusammenfassend lässt sich sagen, dass der Mehrbenutzerbetrieb bei HEAVEN, kein primäres Problem darstellt. Das haben die Erfahrungen aus dem Projekt ESTEDI bewiesen. In der Regel arbeitet mit HEAVEN eine geringe Anzahl an Benutzer, die allerdings sehr berechnungsintensive Anfragen an das System stellen. Durch die Änderung der Ausführungsreihenfolge kann das Inter-Query-Scheduling für bestimmte Anfragen eine erhöhte Ausführungsdauer bedeuten, falls diese in der Warteschlange weiter hinten eingereiht wurden. Besonders in Hinsicht auf die meist hohe Bearbeitungsdauer (mehrere Stunden bis Tage) einer Anfrage, ist eine weitere Verzögerung, durch Bevorzugung anderer, in der Warteschlange befindlicher Anfragen, hinderlich. Die Verteilung der Anfragen nach dem FCFS-Prinzip, stößt bei den Anwendern auf eine gute Akzeptanz. Zur Beschleunigung der Anfragebearbeitung wird bei HEAVEN vielmehr auf eine parallele Bearbeitung von Anfragen durch mehrere RasDaMan-Server gebaut.

Beim Inter-Query-Scheduling wird nur die Reihenfolge der in der Warteschlange befindlichen Anfragen entsprechend bestimmter Kriterien (z.B. FCFS) sortiert und in dieser Reihenfolge ausgeführt. Weiterhin ist es sinnvoll, die einzelnen zur Ausführung anstehenden Anfragen zu

optimieren. Das ist mit Hilfe des Intra-Query-Scheduling beim Retrieval von Daten möglich, die sich auf TS-Medium befinden. Eine Kombination der beiden Scheduling-Verfahren ist ohne Einschränkung möglich und bringt eine weitere Verbesserung der Anfragebearbeitung. Allerdings ist das Scheduling auf der Basis ganzer Anfragen nicht immer ausreichend. Eine verfeinerte Möglichkeit bietet die Verschränkung der Ladevorgänge der einzelnen Anfragen.

3.7.3.3 Anfrageverschränkung

Die Anfrageverschränkung (Query Interleaving) ist die konsequente Weiterentwicklung des Intra- und Inter-Query-Scheduling. Bei diesem Verfahren wird versucht, die Retrievalperformance bei TS-Zugriffen weiter zu optimieren. Die grundlegende Idee hinter diesem Konzept ist, alle in einer Warteschlange befindlichen Ladeaufträge unabhängig von der Zugehörigkeit zu den einzelnen Anfragen, entsprechend einer idealen Ladeperformance zu sortieren. Bisher wurden lediglich die Reihenfolge der zur Ausführung anstehenden Anfragen selbst (Inter-Query-Scheduling) und die einzelnen Ladeaufträge innerhalb dieser Anfragen (Intra-Query-Scheduling) sortiert. Abbildung 3.37 vergleicht nun die unterschiedlichen Varianten des Query-Scheduling gegenüber der naiven Ausführung von mehreren Anfragen einer Warteschlange. In diesem Beispiel werden drei Anfragen an das System gestellt. Anfrage q_1 benötigt die Super-Kacheln σ_{1a}, σ_{1b}, σ_{1c} und σ_{1d}, die sich ausschließlich auf Magnetband v_1 befinden. Die Objekte σ_{2a}, σ_{2b} und σ_{2c} werden von Anfrage q_2 angefragt, welche auf beiden Magnetbändern v_1 und v_2 verteilt vorliegen. Anfrage q_3 wiederum benötigt die Objekte σ_{3a}, σ_{3b} und σ_{3c} des ersten Magnetbandes. Die lexikalische Ordnung der Anfragen und der angeforderten Objekte, spiegelt die naive Ausführungsreihenfolge (Variante 1) der ursprünglichen Anfragen wider. Die enthaltenen unnötigen Medienwechsel- (o_{sv}) und Positionierungsoperationen (o_{pv}) bieten ein gutes Potential zur Anfrageoptimierung.

Die Möglichkeiten des Intra- und des Inter-Query-Scheduling, beeinflussen bei der Optimierung nur jeweils komplette Anfragen. So sortiert das Intra-Query-Scheduling (Abbildung 3.37, Variante 2) die Ladereihenfolge der Objekte einer Anfrage, entsprechend der Lokalität auf dem Magnetband. Bei der Anfrage q_2 wird die Reihenfolge σ_{2b}, σ_{2c} und σ_{2a} gewählt. Die Reihenfolge σ_{2c}, σ_{2a}, σ_{2b} ist ebenso denkbar. Es ist zu erkennen, dass bereits dieser Optimierungsschritt die Anzahl der notwendigen Medienwechsel- und Positionierungsoperationen erheblich reduziert. Die Kombination von Intra- und Inter-Query-Scheduling (Variante 3) optimiert weiterhin die Reihenfolge der Anfragen selbst. Im Beispiel der Abbildung 3.37 wird die Ausführungsreihenfolge der Anfragen q_2 und q_3 getauscht, um einen Medienwechsel einzusparen (entspricht nicht FCFS). Mit der Methode der Anfrageverschränkung kann eine weitere Verbesserung des Ergebnisses in Bezug auf die Ladeperformance erreicht werden. Betrachtet man bei Variante 2 die Ladevorgänge der Anfragen q_1 und q_3, so fällt auf, dass beim Laden der Objekte σ_{1b}, σ_{1a}, σ_{1d} und σ_{1c} der Anfrage q_1, die für die Anfrage q_3 benötigten Objekte σ_{3b}, σ_{3c} und σ_{3a} überlesen werden. Erst nach erneuter Positionierung auf dem Magnetband, werden die Objekte der Anfrage q_3 geladen. Beim Query-Interleaving findet eine Sortierung der Anfragewarteschlange auf Objektebene statt, was zu einer weiteren Minimierung von Medienwechsel- (o_{sv}) und Positionierungsoperationen (o_{pv}) führen kann.

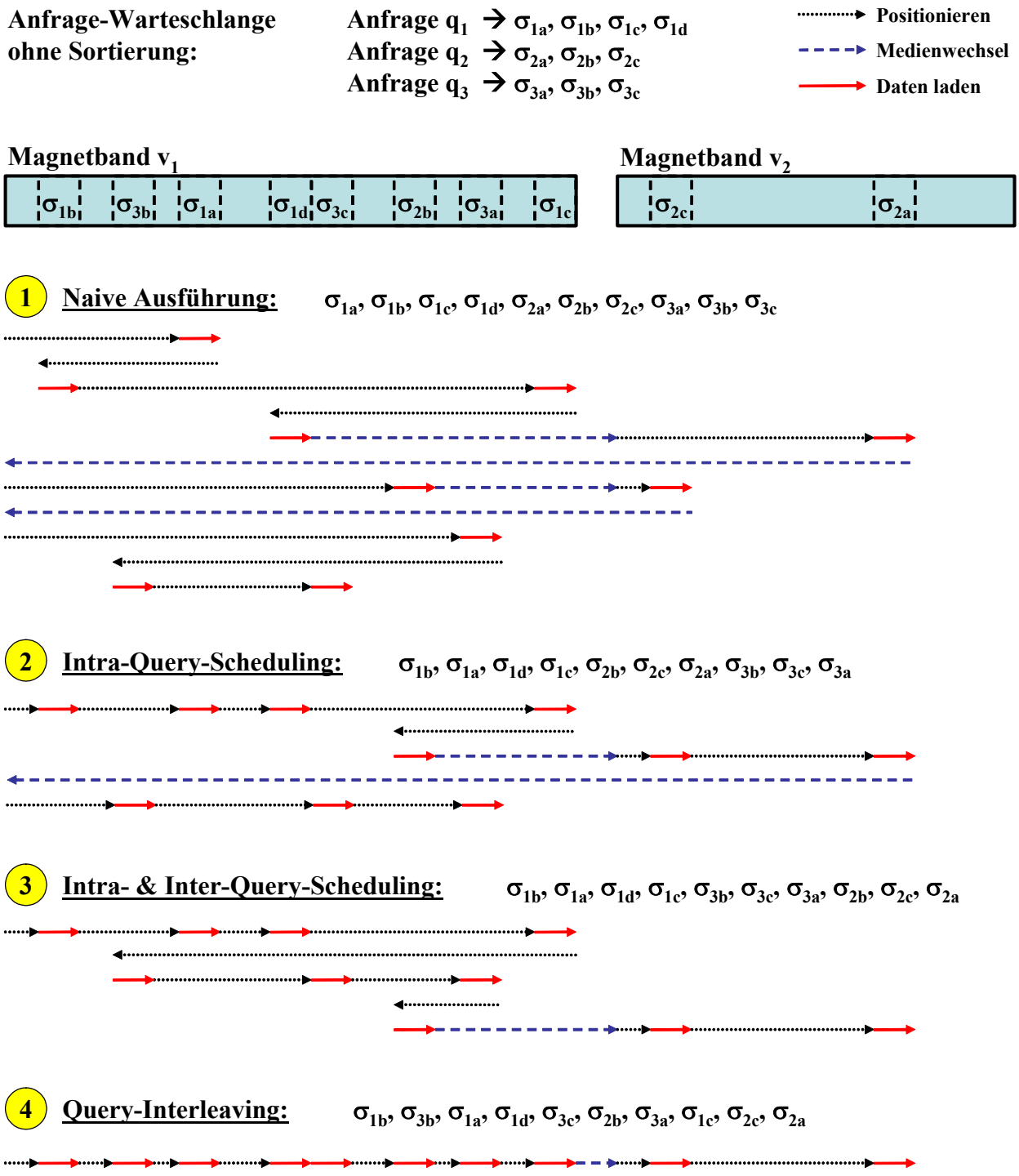


Abbildung 3.37: Vergleich der unterschiedlichen Möglichkeiten des Query-Scheduling.

Es muss eine klare Unterscheidung zwischen optimaler Ladeperformance und optimaler Anfrageperformance getroffen werden. Eine optimale Ladeperformance kann durch die Umsortierung der Anfragewarteschlange eine Verzögerung gewisser Anfragen darstellen, wenn einzelne Ladeaufträge in der Warteschlange nach hinten geschoben werden. Hier greifen wieder die bei Inter-Query-Scheduling beschriebenen Erfahrungen aus dem ESTEDI-Projekt. Aufgrund der meist berechnungsintensiven (CPU-bound) Anfragen, die nicht selten eine

Laufzeit von mehreren Stunden bis Tagen aufweisen, ist eine weitere Verzögerung der Anfragebearbeitung nicht erwünscht. Durch die Möglichkeit der gleichzeitigen Abarbeitung mehrerer Anfragen auf parallel arbeitenden RasDaMan-Servern, wird die Anzahl der in einer Warteschlange befindlichen Anfragen gering gehalten. Steigt die Anzahl der zur Bearbeitung anstehenden Aufträge dauerhaft, sollte die Überlegung angestrebt werden, einen weiteren RasDaMan-Server zu betreiben. Gerade die parallele Abarbeitung mehrerer Anfragen auf verschiedenen RasDaMan-Servern erschwert, bzw. verhindert, die Durchführung einer auf den jeweiligen Ladeaufträgen basierenden Anfrageverschränkung. Das liegt an der unabhängigen Arbeitsweise der RasDaMan-Server. Um eine entsprechende anfrageübergreifende Sortierung der Ladeaufträge durchzuführen, wäre ein entsprechender Informationsaustausch zwischen den einzelnen RasDaMan-Server notwendig. Neben diesem zusätzlichen Verwaltungsaufwand sind für den Entscheidungsprozeß zur Durchführung eines optimierten Query-Interleaving spezifische Informationen über den Systemzustand des verwendeten TS-System notwendig. Mögliche Systemzustände, die zur Entscheidungsfindung beitragen, sind in Tabelle 3.2 aufgelistet. Solche Informationen sind allerdings nur über proprietäre Schnittstellen der HSM-Systeme zu erhalten, was eine Inkompatibilität zu anderen HSM-Systemen bedeutet.

Konventionelle HSM-Systeme selbst verfügen über solche Informationen und realisieren eigene ausgefeilte Scheduling-Algorithmen, die für eine optimale Auslastung der Ressourcen und Ladeperformance sorgen. Diese arbeiten auf der Granularität der gespeicherten Objekte (Dateien). Somit wird jede von HEAVEN angeforderte Super-Kachel als ein separater Ladeauftrag behandelt. Ein bei HEAVEN durchgeführtes Query-Interleaving würde unter Umständen durch das Scheduling-Verfahren des HSM-System wirkungslos werden. Wie bei konventionellen Produkten üblich, sind keine detaillierten Informationen über die eingesetzten Verfahren zu bekommen. Allerdings sind in der einschlägigen Literatur einige Forschungsarbeiten veröffentlicht worden. Hier werden beispielhaft vier speziell für TS-Medien entwickelte Verfahren des E/A-Scheduling mit steigender Komplexität vorgestellt [SM99, TG99, MS98, PAA97, PAAS97, PAAS96, HS96]:

- SCAN: Geeignet für Magnetbänder in Serpentintechnik mit Vorwärts- und Rückwärtsspuren (Kapitel 2.2.2). Alle N Ladeaufträge, die ein TS-Medium betreffen, werden durch einen einzelnen Lesevorgang (Scan) in Vorwärts- und Rückwärtsrichtung über das gesamte Magnetband durchgeführt. Die Ladeaufträge werden zunächst in zwei Mengen, entsprechend der momentanen Leserichtung des Magnetbandes partitioniert und aufgrund ihrer physikalischen Lage auf dem Medium sortiert. Nun werden zunächst alle angefragten Objekte zum Beispiel der Vorwärtsspur und anschließend alle Objekte der Rückwärtsspur gelesen. Die Zeitkomplexität des Algorithmus liegt bei O(N log N).
- MPScan (Multi Pass SCAN): Weiterentwicklung des SCAN-Algorithmus, die mehrere Banddurchläufe erlauben. Es wird ein Nachteil von SCAN beseitigt, indem die Kosten der Spurwechsel berücksichtigt werden. Durch diesen zusätzlichen Einfluss steigt die Zeitkomplexität für N Ladeaufträge auf $O(N^2)$.

- SLTF (Shortest Locate Time First): Beginnend bei einer beliebigen Position des Lesekopfes eines TS-Mediums, wird das Objekt zuerst gelesen, das den geringsten Positionierungsaufwand verursacht. Ausgehend von der neuen Position des Lesekopfes wird wieder das Objekt mit der geringsten Positionierungszeit gelesen, usw. Die Zeitkomplexität beträgt bei N Ladeaufträgen $O(N^2)$.
- OPT (Optimal Scheduler): Der Algorithmus durchläuft N_v TS-Medien in nicht aufsteigernder Ordnung, entsprechend dem Verhältnis der Anzahl N an Ladeaufträgen zur Summe der Ausführungszeit und der Medienwechselzeit. OPT liefert immer die kürzest mögliche Gesamtausführungszeit. Bei nur einer Schreib-/Lesestation, wird eine Zeitkomplexität von $O(N_v \log N_v + N)$ erreicht. Bei mehreren Laufwerken führt das zu einer exponentiellen Zeitkomplexität. Dadurch ist dieser Algorithmus nur für geringe Anzahl an Ladeaufträge geeignet.

Die ersten drei Verfahren berücksichtigen nur das momentan eingelegte TS-Medium und eine Schreib-/Lesestation. Im Gegensatz dazu ermittelt der OPT-Algorithmus eine optimale Ausführungsreihenfolge für mehrer Medien und mehrere Schreib-/Lesestationen. Weiterführende Betrachtungen über Scheduling-Verfahren im Bereich des Datenretrieval von tertiären Speichermedien sind bei [Lijd03, SM99, TG99, MS98, PAA97, PAAS97, PAAS96, HS96, Sara96, Sara95b] zu finden.
Bei dem im Rahmen von HEAVEN entwickelten Tape-Controller-Toolkit (TCT, Kapitel 3.3) wurde eine Anfrageverschachtelung nach dem SCAN-Verfahren implementiert. Das TCT nimmt die Ladeaufträge der parallel arbeitenden RasDaMan-Server entgegen und sortiert die einzelnen Ladeaufträge entsprechend der Lokalität auf dem Magnetband. Dabei wird sowohl die aktuelle Position des Lesekopfes, als auch die momentane Leserichtung berücksichtigt. Im Gegensatz zu RasDaMan verfügt TCT über solche gerätespezifische Informationen. Um zeitaufwändige Medienwechsel zu vermeiden werden bei TCT zunächst die Ladeaufträge abgearbeitet, die das gerade eingelegte Magnetband betreffen. Erst im Anschluss erfolgt ein Medienwechsel. Die Tatsache, dass TCT in der gegenwärtigen Version keine parallelen Schreib-/Lesestationen verwaltet, hält die Zeitkomplexität des Algorithmus zur Verschachtelung der Ladeaufträge mehrerer Anfragen in einem vertretbaren Rahmen.

Zusammenfassend lässt sich sagen, dass die Realisierung des Query-Interleaving, aufgrund der parallelen Abarbeitung von Anfragen durch unabhängige RasDaMan-Server, sehr aufwändig ist. Auch fehlende Informationen über den Systemzustand der angebundenen TS-Systeme, erlauben keine Ermittlung einer optimalen Ausführungsreihenfolge der Ladeaufträge. Vielmehr wird auf ausgefeilte Verfahren des E/A-Scheduling der konventionellen HSM-Systeme, bzw. der Eigenentwicklung des TCT zurückgegriffen. Die dort eingesetzten Algorithmen können zur Entscheidungsfindung die vorhandenen Systemzustände mit einfließen lassen.

3.7.4 Erreichtes Optimierungspotential

Sind bei einer Anfrage Zugriffe auf TS-Medien involviert, kann nicht entsprechend der traditionellen Art und Weise von RasDaMan verfahren werden. Das führt aufgrund zufälliger

Zugriffe auf TS-Medien zu einer schlechten Performance der Anfragebearbeitung: Es liegt an der Vielzahl der benötigen Positionierungsoperationen. Teilweise sind weiterhin zeitaufwändige Medienwechsel notwendig (mehrere Minuten). Um eine Performancesteigerung bei TS-Zugriffen zu erreichen, sind unbedingt Maßnahmen zu ergreifen, um unnötige Positionierungs- und Medienwechseloperationen einzusparen. Um eine Verbesserung zu erreichen, wurden Techniken, wie das Intra-Query-Scheduling, das Inter-Query-Scheduling und das Query-Interleaving untersucht. Dabei werden durch eine geschickte Vorsortierung der Anfragewarteschlange zufällige Ladevorgänge von TS-Medien reduziert und möglichst durch kombinierte Operationen ersetzt. Bei allen Verfahren ist eine klare Unterscheidung zwischen optimaler Ladeperformance und optimaler Anfrageperformance zu treffen. Eine optimale Ladeperformance kann durch die Umsortierung der Anfragewarteschlange eine Verzögerung gewisser Anfragen darstellen, wenn diese in der Warteschlange nach hinten geschoben werden.

Das realisierte Intra-Query-Scheduling passt die Anforderungsreihenfolge für die auf TS-Medien gespeicherten Datenobjekte (Super-Kacheln) entsprechend der beim Exportieren gewählten Linearisierungsordnung an. Die in HEAVEN implementierte Umsetzung, minimiert die Retrievalkosten bereits bei Einzelobjekten (MDD) um über 53,3 % (Speed-Up: 2,14) durch die Einsparung, bzw. Reduzierung von Positionierungs- und Medienwechselzeiten. Bei Zugriffen auf Objektmengen wird ein weitaus höheres Einsparungspotential erreicht. Eine Performance Evaluierung folgt in Kapitel 4.
Inter-Query-Scheduling wird bei HEAVEN nach dem FCFS-Prinzip, im Zusammenspiel mit der Verwendung paralleler RasDaMan-Server durchgeführt. Dadurch ist ein Mehrbenutzerbetrieb bei HEAVEN kein primäres Problem, was auch die Erfahrungen aus dem Projekt ESTEDI beweisen. In der Regel arbeitet mit HEAVEN eine geringe Anzahl an Benutzer, die allerdings sehr berechnungsintensive Anfragen an das System stellen. Durch die Änderung der Ausführungsreihenfolge, kann eine andere Strategie beim Inter-Query-Scheduling für bestimmte Anfragen eine erhöhte Ausführungsdauer bedeuten, falls diese in der Warteschlange weiter hinten eingereiht wurden. Besonders in Hinsicht auf die meist hohe Bearbeitungsdauer (mehrere Stunden bis Tage) einer Anfrage ist eine weitere Verzögerung durch Bevorzugung anderer in der Warteschlange befindlicher Anfragen hinderlich.

Innerhalb RasDaMan wurde auf eine Realisierung des Query-Interleaving, der konsequenten Weiterentwicklung des Intra- und Inter-Query-Scheduling, aus zweierlei Gründen verzichtet: Einerseits ist Query-Interleaving, aufgrund der parallelen Abarbeitung von Anfragen durch unabhängige RasDaMan-Server, sehr aufwändig zu realisieren. Andererseits erlauben fehlende Informationen über den Systemzustand der angebundenen TS-Systeme keine Ermittlung einer optimalen Ausführungsreihenfolge von Ladeaufträgen. Vielmehr wird hier auf ausgefeilte Verfahren des E/A-Scheduling der konventionellen HSM-Systeme zurückgegriffen. Bei der Eigenentwicklung des TCT wurde das SCAN-Verfahren realisiert. Die dort eingesetzten Algorithmen können zur Entscheidungsfindung die dort vorhandenen Systemzustände mit einfließen lassen.

3.8 Delete / Update / Re-Import und Prefetching von Daten

Neben der Funktionalität des Exportierens und des Retrievals von multidimensionalen Daten, auf, bzw. von tertiären Speichermedien, sind das Löschen (Delete), Updaten, Re-Importieren und das Vorladen (*Prefetching*) von Daten essentielle Bestandteile einer Speicherverwaltung und somit von HEAVEN. Um eine konsistente Datenbasis zu erhalten, ist es erforderlich, Lösch- und Update-Vorgänge von der höchsten, bis zur niedrigsten Speicherebene zu propagieren. Die jeweiligen Speicherinhalte werden entsprechend angepasst. Zunächst werden die genannten Funktionen formal definiert:

Definition 3.19: Funktionen *delete*(object, source), *update*(object, source), *reimport*(object, source), und *prefetch*(object, source)

Die Funktion *delete*(object, source) löscht ein Objekt (object) auf der angegebenen Speicherebene (source):
delete : object, source → $\mathbb{B}$,
wobei gilt object ∈ Θ_α und source ∈ {P, SR, SRC, SHC, T_{on}, T_{near}, T_{off}}

Die Funktion *update*(object, source) aktualisiert ein geändertes Objekt (object) einer Speicherebene (source):
update : object, source → $\mathbb{B}$,
wobei gilt object ∈ Θ_τ und source ∈ {P, SR, SRC, SHC, T_{on}, T_{near}, T_{off}}

Die Funktion *reimport*(object, source) verschiebt ein Objekt (object) komplett von den Speicherebenen SRC, SHC, T_{on}, T_{near}, bzw. T_{off} in den Speicherbereich SR. Zur Realisierung werden die Funktionen *move*, *locate* und *delete* herangezogen:
reimport : object, source → $\mathbb{B}$, bzw.
reimport(object, source) = *move*(object, *locate*(object), target); *delete*(object, {SRC, SHC, T_{on}, T_{near}, T_{off}}),
wobei gilt object ∈ Θ_α und source ∈ {SRC, SHC, T_{on}, T_{near}, T_{off}} und target ∈ {SR}. Nach dem Re-Import gilt $\alpha \in \{\Theta_P, \Theta_{SR}\}$ und $\alpha \notin \{\Theta_{SRC}, \Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$.

Die Funktion *prefetch*(object, source) ist einen Spezialfall der *replicate*(object, source, target) Funktion aus Definition 3.18, mit eingeschränktem Wertebereich für source und target:
prefetch : object, source = *replicate* : object, source, traget → $\mathbb{B}$,
wobei gilt object ∈ Θ_τ und source ∈ {SHC, T_{on}, T_{near}, T_{off}} und target ∈ {SRC}

Im Folgenden werden die Funktionen *delete*(object, source), *update*(object, source), *reimport*(object, source), und *prefetch*(object, source) näher beschrieben:

Delete-Funktionalität
Zum Entfernen multidimensionaler Objekte aus der Speicher- und Archivierungsumgebung HEAVEN wurde die Löschfunktionalität von RasDaMan erweitert und angepasst. Um den

Löschvorgang für den Anwender transparent zu gestalten, ist die bereits in RasDaMan vorhandene RasQL-Anweisung zum Löschen von Objekten beibehalten worden:

```
DELETE  FROM        <Kollektion>
        [ WHERE     <Selektionsbedingung> ]
```

Der Anwender benötigt keinerlei Information über den tatsächlichen Speicherort der Objekte. Die kleinste Objektgranularität zum Löschen ist ein einzelnes MDD α. Das kann durch die Angabe einer Selektionsbedingung aus einer Kollektion ausgewählt werden. Fehlt die Selektionsbedingung, so werden alle MDD, die sich in einer Kollektion befinden, gelöscht. Soll das Kollektionsgerüst ebenfalls entfernt werden, so ist dies mit DROP COLLECTION zu erreichen. Wird ein Objekt aus HEAVEN entfernt, so wird die Funktion *delete*(object, source) schrittweise auf alle Speicherebenen (source $\in$ {P, SR, SRC, SHC, T_{on}, T_{near}, T_{off}}) angewendet, in denen das Objekt gespeichert vorliegt. Ist $\alpha \in \Theta_{SR}$, wurde keine Auslagerung des Datensatzes auf TS-Medien vorgenommen ($\alpha \notin \{\Theta_{SRC}, \Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$) und das Entfernen des Datenobjektes entspricht einem herkömmlichen Löschvorgang von RasDaMan. Das bedeutet, alle betroffenen Kacheln (BLOBs) eines MDD und der zugehörige Index werden aus dem konventionellen DBMS gelöscht. Wurde das MDD exportiert ($\alpha \in \{\Theta_{SRC}, \Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$), wird die Löschoperation sofort auf alle entsprechenden Speicherebenen propagiert, um einen konsistenten Datenbestand zu erhalten. Der Löschvorgang von RasDaMan und dem HSM-System, ist ebenso wie beim Export entkoppelt. Das bedeutet, sobald das MDD aus Θ_P, Θ_{SRC} und Θ_{SHC} entfernt wurde, steht der RasDaMan-Server für weitere Aufgaben zur Verfügung. Danach übernimmt das HSM-System eigenständig das Löschen der Datenobjekte aus den Speicherbereichen Θ_{Ton}, Θ_{Tnear} oder Θ_{Toff}. Befindet sich zum Beispiel ein MDD nur auf TS-Medium ($\alpha \in \{\Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$), so sind aus der Sicht von RasDaMan nur der Verweis (Stub-File oder Anchor) auf dem HSM-Cache (SHC) und die entsprechenden Indexeinträge zu entfernen. Dieses typische Szenario ist für RasDaMan ideal, da sehr schnell neue Auftrage bearbeitet werden können. Beim Löschen von Daten eines Magnetbandes, ist anzumerken, dass die Datensätze nicht wirklich gelöscht werden, sondern nur als gelöscht markiert werden. Begründet ist dies durch die Tatsache, dass es bei Magnetbändern nicht möglich ist, in die Lücken gelöschter Bereiche neue Datensätze zu schreiben. Somit entsteht mit der Zeit eine Fragmentierung. Um diese Fragmentierung zu beseitigen und den ungenutzten Speicherplatz wieder freizugeben, besitzen die meisten HSM-Systeme automatisierte Methoden. Überschreitet die Fragmentierung auf einem Magnetband eine einstellbare Schwelle, so werden die gültigen Datensätze von diesem Magnetband auf ein neues Magnetband geschrieben. Das ursprüngliche Magnetband wird als leeres Medium gekennzeichnet und zum erneuten Beschreiben freigegeben.

Update-Funktionalität

Das Ändern von Datensätzen wurde in HEAVEN transparent für den Anwender realisiert. Das bedeutet, dass der Anwender keinerlei Information über den tatsächlichen Speicherort des entsprechenden Datensatzes benötigt und die ursprüngliche Update-Anweisung verwenden kann:

UPDATE <Kollektion>
SET <Update- Ausdruck>
[WHERE <Selektionsbedingung>]

Jedes MDD, das der Selektionsbedingung genügt, wird verändert. Dabei überschreiben die im Update-Ausdruck definierten Angaben die entsprechenden Zellen der gewählten Regionen eines MDD. Als Randbedingung gilt, dass die Domäne des Update-Ausdrucks maximal gleich groß der Domäne des MDD sein darf. Mit dem Update-Ausdruck können Einschränkungen bezüglich des multidimensionalen Intervalls angegeben werden [Rasd02]. Ist $\alpha \in \Theta_{SR}$, so ist verglichen mit der ursprünglichen RasDaMan-Version, kein zusätzlicher Aufwand notwendig, da keine Datensätze auf Tertiärspeicher betroffen sind. Die Update-Granularität im Speicherbereich SR basiert auf Kacheln τ.
Wurde das MDD α auf Tertiärspeicher exportiert ($\alpha \in \{\Theta_{SRC}, \Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$), so müssen die Änderungen auf die weiteren Speicherebenen (SHC, T_{on}, T_{near}, T_{off}) in Granularität von Super-Kacheln σ propagiert werden. Eine solche Synchronisation findet direkt beim Auftreten einer Änderungsoperation statt. Das bedeutet, es werden Änderungen sofort von der höchsten Speicherebene (SRC) bis zur niedrigsten Speicherebene (T_{on}, T_{near}, bzw. T_{off}) durchgeschrieben. Dieses Vorgehen wird auch als Write-Through-Verfahren bezeichnet. Es bestehen keine Konsistenzprobleme, da andere RasDaMan-Server auf den gleichen Cache-Bereich zugreifen und immer den aktuellsten Wert lesen. Hier greift ebenfalls das Transaktionskonzept des verwendeten konventionellen DBMS, das zum Beispiel einen so genannten Dirty-Read verhindert. Obwohl ein sofortiges Durchschreiben von Änderungen aus Performancegründen teilweise vermieden wird, bereitet dieses Verfahren bei *HEAVEN* aus zweierlei Hinsicht geringe Probleme: Einerseits ist der Bedarf im HPC-Bereich nicht sehr hoch, Datensätze zu ändern. In diesem Anwendungsgebiet bleiben Originaldaten überwiegend in ihrer ursprünglichen Form erhalten. Veränderte Simulationsergebnisse mit angepassten Parametern werden meist zusätzlich abgespeichert. Andererseits ist der Updatevorgang ebenso, wie der Export- und Löschvorgang, zwischen RasDaMan und dem HSM-System entkoppelt. Das bedeutet, dass RasDaMan die Änderungen im Primärspeicher (P), im Sekundärspeicher-Cache von RasDaMan (SRC) und im Sekundärspeicher-Cache des HSM-Systems (SHC) durchführen muss. Sobald die Änderungen den HSM-Cache erreicht haben, propagiert das HSM-System das Update der Daten in dem entsprechenden Speicherbereich (T_{on}, T_{near} oder T_{off}).

Aufgrund der bei arbiträrer Kachelung auftretenden Mehrfachspeicherung von Kacheln τ in unterschiedlichen Super-Kacheln σ, sind konsistenzerhaltende Maßnahmen notwendig. Wurden Kacheln in unterschiedlichen Super-Kacheln doppelt gespeichert, so sind alle betroffenen Super-Kacheln der jeweiligen Speicherebenen (Θ_{SHC}, Θ_{Ton}, Θ_{Tnear}, Θ_{Toff}) anzupassen. Wurde reguläre, ausgerichtete oder irreguläre Kachelung gewählt, sind diese Maßnahmen nicht erforderlich. Auch bei einem Update, das ein ganzes Objekt (MDD) einschließt, sind keine besonderen Maßnahmen erforderlich. In diesem Fall müssen sowieso alle Super-Kacheln angepasst werden.

Da auf dem TS-Medium einzelne Speicherbereiche nicht überschrieben werden können, wird der alte Datensatz als gelöscht markiert und der geänderte Datensatz nach dem letzten auf dem Band befindlichen Datensatz geschrieben. Reicht der Platz auf dem Magnetband nicht mehr aus, so muss ein neues Magnetband verwendet werden. Dieses Vorgehen zeigt deutlich, dass bei jedem Updatevorgang, das Inter-Super-Tile-Clustering, zumindest zum Teil, aufgebrochen wird. Die Antwortzeit zukünftiger Anfragen steigt aufgrund zusätzlicher Operationen zum Positionieren. Wie bereits bei der Delete-Funktionalität beschrieben, verfügen HSM-Systeme über Mechanismen zum Defragmentieren von TS-Medien. Allerdings wird nicht die ursprüngliche Reihenfolge, die dem Inter-Super-Tile-Clustering entspricht, auf dem Magnetband wieder hergestellt, sondern die aktuelle Sequenz beibehalten. Die Defragmentierung wird nur eingesetzt, um ungenutzten Speicherplatz auf den Magnetbändern zu reaktivieren.
Eine Möglichkeit, um das ursprüngliche Inter-Super-Tile-Clustering wieder zu erhalten, ist eine eigenständige Verwaltung des Fragmentierungsgrades zu realisieren. Diese Verwaltung kann durch einen Systemkatalog übernommen werden. Wird ein bestimmter Grenzwert überschritten, müssen die betroffenen Super-Kacheln einer Kollektion, bzw. eines MDDs, auf Magnetband reorganisiert werden. Diese Reorganisation bedeutet allerdings, dass die betroffenen Datenobjekte von TS-Medien geladen und erneut auf Magnetband geschrieben werden müssen. Bei den im HPC-Bereich vorhandenen großen Datensätzen, bedeutet das natürlich einen hohen Aufwand für das HSM-System bei der Aus- und Einlagerung. Die Erfahrungen aus dem ESTEDI-Projekt zeigten allerdings, dass gerade im HPC-Bereich ein Update von Datensätzen eher selten vorkommt. Originaldaten bleiben in der Regel in der ursprünglichen Form erhalten. Simulationsergebnisse werden entweder aus den Originaldaten wieder reproduziert oder werden zusätzlich gespeichert. Aus diesem Grund wurde auf die Realisierung einer eigenständigen Komponente zur Defragmentierung innerhalb von HEAVEN verzichtet. Bei Bedarf ist eine solche Funktionalität einfach zu integrieren, da auf vorhandene Funktionen, wie zum Beispiel das Ein- und Auslagern von Daten, zurückgegriffen werden kann. Die Erweiterung der Anfragesprache RasQL könnte wie folgt aussehen:

```
DEFRAG FROM       <Kollektion>
       [ WHERE    <Selektionsbedingung> ]
```

Eine Defragmentierung bedeutet zunächst einen Re-Import der entsprechenden Objekte und einen erneuten Export. Wie im Folgenden beschrieben, impliziert eine Re-Import-Anweisung das Entfernen der Datensätze auf tertiäre Speichermedien. Durch gespeicherte Heuristiken über den Defragmentierungsgrad in einem Systemkatalog, ist es möglich, in Zeiten niedriger Systemlast (Nacht, Wochenende), eine automatisierte Defragmentierung durchzuführen.

Re-Import-Funktionalität
Um MDDs, die auf TS-Medium exportiert wurden, wieder dauerhaft in dem von RasDaMan verwendeten konventionellen DBMS zu speichern, ist der Re-Import von Daten implementiert worden. Dadurch wird das Exportieren von MDDs rückgängig gemacht und der ursprüngliche Zustand wieder hergestellt. Die Anfragesprache RasQL wurde um das entsprechende Konstrukt erweitert:

```
REIMPORT  FROM     <Kollektion>
          [ WHERE  <Selektionsbedingung> ]
```

Mit dem Befehl REIMPORT können Kollektionen, bzw. durch Angabe einer Selektionsbedingung, einzelne MDD α dauerhaft von einem HSM-System in den Sekundärspeicher (SR) von RasDaMan verschoben werden. Das Verschieben eines MDD erfolgt in zwei Phasen: In einer ersten Phase wird das MDD vom Tertiärspeicher in den Sekundärspeicher (SR) von RasDaMan geladen. In einer zweiten Phase erfolgt das Löschen der entsprechenden Super-Kacheln dieses MDD auf dem HSM-Cache (SHC). Das HSM-System entfernt daraufhin automatisch die betroffenen Datenelemente vom TS-Medium. Während des Löschvorganges des HSM-Systems, kann RasDaMan bereits neue Anfragen bearbeiten. Nach ausgeführtem Re-Import gilt: $\alpha \in \{\Theta_P, \Theta_{SR}\}$ und $\alpha \notin \{\Theta_{SRC}, \Theta_{SHC}, \Theta_{Ton}, \Theta_{Tnear}, \Theta_{Toff}\}$.

Prefetching-Funktionalität

Die Prefetching-Funktionalität wurde eingeführt, um einem Anwender die Möglichkeit zu bieten, bestimmte Datenobjekte vor der eigentlichen Anfragebearbeitung vorzuladen. Während der Laufzeit des ESTEDI-Projektes, hat sich gezeigt, dass Anwender oft bereits im Voraus wissen, dass sie in naher Zukunft (Stunden, Tage) Operationen auf bestimmten Datenobjekten ausführen möchten. Handelt es sich bei diesen Datenobjekten um auf TS-Medium ausgelagerte Daten, so kann die Retrievalzeit bei der tatsächlichen Bearbeitung erheblich reduziert werden: Es werden die relevanten Daten einfach vom TS-Medium in den Sekundärspeicher-Cache von RasDaMan (SRC) vorgeladen. Zu diesem Zweck wurde die Anfragesprache RasQL um folgendes Konstrukt erweitert:

```
PREFETCH  SELECT   <Geometrische Operation>
          FROM     <Kollektion>
          [ WHERE  <Selektionsbedingung> ]
```

Durch das Schlüsselwort PREFETCH werden relevante Datenobjekte von den Speicherbereichen SHC, T_{on}, T_{near}, oder T_{off} eines HSM-Systems in den Speicherbereich SRC repliziert. Im Zusammenhang mit dem Schlüsselwort PREFETCH werden im SELECT-Zweig nur geometrische Operationen ausgewertet, da sie die Domäne einschränken und das zu übertragende Datenvolumen eines MDDs reduzieren. Es werden keine weiteren Berechnungen auf dem RasDaMan-Server ausgeführt. Um die Ausführung in Zeiten mit geringer Serverlast (Nacht, Wochenende) zu legen, ist in der aktuellen Version eine entsprechende Anfrage mittels eines so genannten Cron-Job (Command Run On) zu starten. Verschiedene Betriebssysteme (z.B. Unix oder Linux) erlauben, Kommandos und Dienste durch einen Cron-Job zeitbasiert zu starten. Es ist auch denkbar, ein zeitbasiertes Starten in den RasDaMan-Server zu integrieren. Zusätzlich ist es möglich, das Konstrukt mit einer Startzeit (Datentyp: datetime) und einer Verweildauer zu ergänzen. Durch die Angabe einer Verweildauer könnte eine vorzeitige Verdrängung aus dem Sekundärspeicher-Cache-Bereich von RasDaMan (SRC) verhindert werden.

Neben der hier beschriebenen Variante, das Vorladen explizit durch einen Anwender anzustoßen, ist es denkbar, automatisierte Strategien zu entwickeln. Da das Zugriffsverhalten von Anwendern nicht deterministisch ist und sowohl die Zugriffssequenz auf bestimmte Objekte, bzw. Teilobjekte, als auch die eigentliche Nutzungsdauer, kein typisches Verhalten aufweisen, ist ein zeitliches Verhalten nur stochastisch zu beschreiben. Diese stochastische Beschreibung basiert auf Wahrscheinlichkeiten für das Auftreten bestimmter Zugriffsmuster und der Nutzungsdauer von Objekten. Durch diese Wahrscheinlichkeiten, soll das zukünftige Nutzerverhalten stochastisch vorhergesagt werden. So ist es denkbar, bei einem Ladevorgang vom TS-Medium, nicht nur die für eine Anfrage relevanten Datenobjekte zu laden, sondern ebenfalls zusätzliche Datenobjekte, die mit einer gewissen Wahrscheinlichkeit in naher Zukunft benötigt werden. Zu beachten ist, dass für das zusätzliche Laden von Daten auch zusätzliche, kostenintensive Operationen zum Positionieren notwendig werden. Nicht vermeidbare Fehlentscheidungen beim Vorladen von Datenobjekten haben gravierende Auswirkungen auf die Retrieval-Performance des Systems. Um eine maximale Retrieval-Performance zu erreichen und um die Netzwerklast möglichst gering zu halten, wurde aufgrund des großen Datenvolumens, bewusst auf das automatische, zusätzliche Laden von Datenobjekte verzichtet. Stattdessen sind Caching-Strategien entwickelt worden, die Eigenschaften multidimensionaler Daten und die Charakteristika von TS-Systemen berücksichtigen.

Anzumerken ist noch, dass der durch das Super-Kachel-Konzept überschüssig gelesene Datenbereich, im weitesten Sinn als eine Art Vorladen (Prefetching) von Kacheln angesehen werden kann. Durch das Super-Kachel-Konzept, findet bezogen auf die Domäne der Anfrage, meist eine Überladung von Kacheln im Speicher-Bereich SRC statt. Als Maßzahl für den Anteil der zusätzlich geladenen Kacheln kann der Verschnitt $\upsilon_{\sigma\setminus\tau}$ angeführt werden (Definition 3.8). Bei einer Folgeanfrage, die diesen zusätzlich vorgeladenen Datenbereich benötigt, entfallen zeitaufwändige TS-Zugriffe. Der größte Effizienzgewinn wird dabei beim Laden von physisch benachbarten Seiten erreicht, da sich hierbei das im Vergleich zu wahlfreien Zugriffen wesentlich günstigere Zugriffsverhalten von sequentiellen Leseoperationen ausnutzen lässt.

3.9 Caching von Array-Daten

In diesem Kapitel wird untersucht, wie sich das Retrieval von multidimensionalen Daten, die auf TS-Medien gespeichert wurden, durch Caching nennenswert beschleunigen lässt. Die grundlegende Funktionsweise eines schnellen Zwischenspeichers beruht auf einer vorhandenen Lokalitätseigenschaft. In der Informatik wird darunter verstanden, dass in einem gewissen Zeitabschnitt nur auf einen relativ kleinen Bereich der gesamten Datenmenge zugegriffen wird. Diese Aussage wird durch Erfahrungen aus dem Projekt ESTEDI bestätigt. Nach [Laut95] wird in 80 – 90 Prozent aller Retrievalfälle auf nur 10 Prozent des gesamten Datenvolumens zugegriffen.

Das Zwischenspeichern von Datenobjekten auf einer schnelleren Speicherebene hat vor allem das Ziel, zeitaufwändige Zugriffe auf TS-Medien zu vermeiden. Eine korrekte Verwaltung der

zwischengespeicherten Objekte bedarf durchdachter Algorithmen. Diese stellt die notwendige Konsistenz zwischen den Kopien und den Originalen her. Idealerweise befindet sich jedes Datenobjekt zum Zeitpunkt des Zugriffs im Cache des Primär- oder Sekundärspeichers. Da der Cache im Normalfall um Größenordungen kleiner ist, als der vorhandene Hintergrundspeicher, können nicht alle Objekte vorgehalten werden. So müssen nicht im Cache vorhandene, aber für eine Anfrage benötigte Datenobjekte, aus dem Hintergrundspeicher geladen werden. Im Falle von Änderungen sind diese in den Originalbereich zu propagieren, um einen konsistenten Datenbestand zu wahren. Weiterhin werden Strategien zum Verdrängen von Datenobjekten aus dem Cache-Bereich eingesetzt. Die Absicht dabei ist, dasjenige Objekt als Verdrängungskandidat zu wählen, das den geringsten „Schaden" verursacht. Um die Funktionsweise eines Caches zu beurteilen, wird als Bewertungskriterium die so genannte Cache-Trefferrate eingeführt:

Definition 3.20: Cache-Trefferrate (Cache-Hit-Ratio) H

Die Cache-Trefferrate (Cache-Hit-Ratio) H beschreibt das Verhältnis der Anzahl an Treffern (Hits) beim Versuch, Datenobjekte aus dem Cache-Bereich zu laden, gegenüber der Anzahl der bei Anfragen insgesamt angeforderten Objekte:

$$H = \frac{\text{Anzahl der Treffer (Hits)}}{\text{Anzahl der angeforderten Objekte}};\ \text{es gilt: } 0 \leq H \leq 1$$

In der Literatur wird zur Bewertung eines Caches teilweise auch die Cache-Fehlerrate (Cache-Fault-Ratio) herangezogen. Die Cache-Fehlerrate $F = 1 - H$ steht für das Verhältnis der Anzahl der Fehlversuche (Miss) beim Versuch, Datenobjekte aus dem Cache-Bereich zu laden, gegenüber der Anzahl der bei Anfragen angeforderten Objekte ($0 \leq F \leq 1$).

Nach dieser allgemeinen Einführung wird die konkrete Umsetzung einer Caching-Hierarchie bei HEAVEN vorgestellt. Dabei liegt das Hauptaugenmerk nicht auf dem Caching von Objekten im Primärspeicher, sondern vor allem auf dem Caching von Datenobjekten im Sekundärspeicher-Cache, die sich normalerweise auf TS-Medien befinden.

3.9.1 Umsetzung einer Caching-Hierarchie

Bei der in HEAVEN realisierten Fusion des DBMS RasDaMan mit konventionellen HSM-Systemen, wurde neben der Erweiterung der Speicherhierarchie um TS-Medien, zusätzlich eine Caching-Hierarchie entwickelt. Dabei werden Datenobjekte über mehrere Stufen durchgereicht, bzw. vorgehalten. Das vermindert die operative Distanz der Verarbeitungseinheit (RasDaMan-Server) zu den Datenobjekten. Abbildung 3.38 verdeutlicht die in HEAVEN realisierte Caching-Hierarchie. Die abgebildeten Pfeile zeigen den Datenfluss mit der entsprechenden Zugriffsgranularität einer Kachel τ, bzw. Super-Kachel σ.

Die dargestellte, dreistufige Caching-Hierarchie besteht aus den Speicherbereichen P, SRC und SHC. Die im Primärspeicher (P) befindlichen Datenobjekte, werden nur während der

Laufzeit einer Anfrage zwischengespeichert. Im Gegensatz dazu handelt es sich bei dem TS-Cache-Bereich (SRC) des von RasDaMan verwendeten konventionellen DBMS (z.B. Oracle) und dem Sekundärspeicher-Cache des HSM-Systems (SHC) um persistente Speicher. Die Datenobjekte bleiben auch bei einem Neustart von RasDaMan, bzw. des Rechners, erhalten. Bei der Ausführung einer Anfrage, wird bei einem Objektzugriff *access*(τ) mittels *locate*(τ), die in der Speicherhierarchie am höchsten gelegene Speicherebene ermittelt, auf der sich das Datenobjekt befindet. Wurde das Datenobjekt weder im Cache-Bereich SRC oder SHC entdeckt, erfolgt ein TS-Zugriff im HSM-System. Das erfordert zunächst das Einlegen des gesuchten TS-Mediums in eine freie Lesestation. Danach wird das Datenobjekt von diesem TS-Medium (T_{on}) in den Sekundärspeicher-Cache des HSM-Systems (SHC) geladen. Ein Zugriff erfolgt in der eingeführten Granularität einer Super-Kachel σ. Im Anschluss wird das Datenobjekt vom Speicherbereich SHC in den TS-Cache-Bereich (SRC) des von RasDaMan verwendeten konventionellen DBMS (z.B. Oracle) überführt. Die Zugriffsgranularität in Form einer Super-Kachel ändert sich im RasDaMan-Server. Dort werden aus diesem Container die darin enthaltenen Kacheln rekonstruiert und in das DBMS (SRC) eingebettet. Zur Ausführung der Operationen wird das Datenobjekt im Primärspeicher (P) materialisiert. Nach der Bearbeitung der Anfrage erfolgt der in Abbildung 3.38 nicht mehr dargestellte Transfer des Ergebnisses zur Applikation.

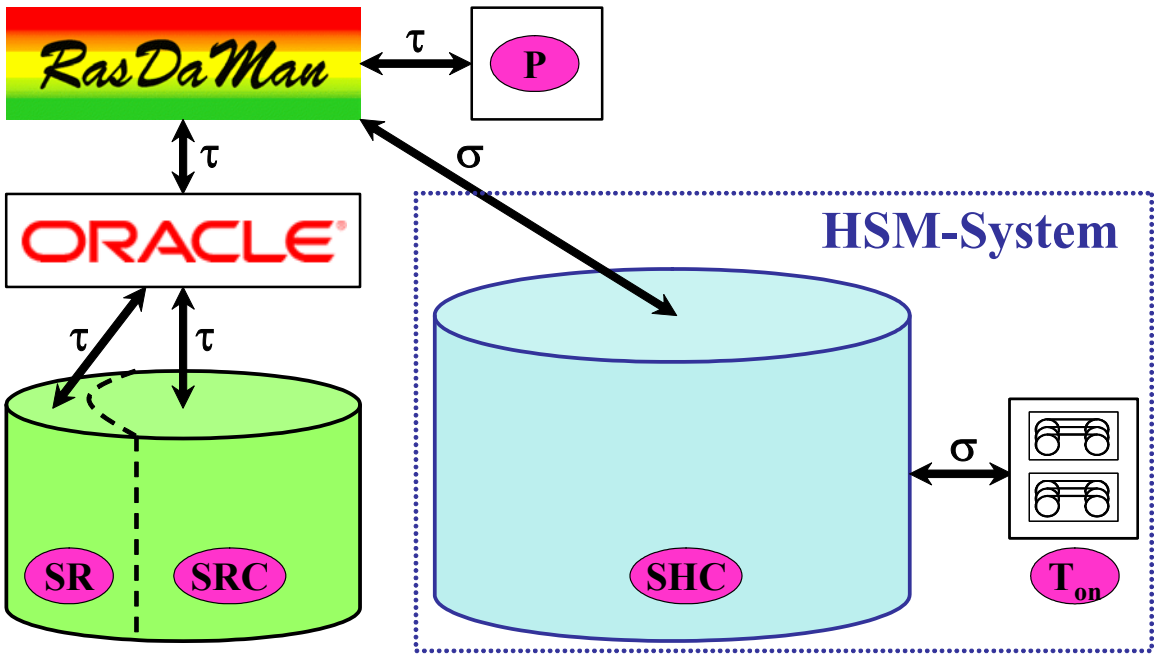


Abbildung 3.38: In *HEAVEN* realisierte Caching-Hierarchie, bestehend aus P, SRC und SHC.

Der in Abbildung 3.38, links unten, dargestellte Speicherbereich SR spiegelt die ursprüngliche Datenhaltung von RasDaMan ohne TS-Anbindung wider. Bei *HEAVEN* wird dieser Speicherbereich für Datenobjekte genutzt, die sehr häufig angefragt und dementsprechend nicht auf TS-Medien ausgelagert werden. Die Speicherinhalte Θ_{SR} und Θ_{SRC} sind disjunkt ($\Theta_{SR} \cap \Theta_{SRC} = \varnothing$). Um das Zugriffsverhaltens auf die unterschiedlichen Speicherbereiche SRC, SHC

und T_{on} zu untersuchen, wird der Speicherbereich SR als Referenzmarke herangezogen. Die Abbildung 3.39 vergleicht die Gesamtlaufzeit einer Bereichsanfrage bei unterschiedlichen Speicherebenen. Für die Anfrage werden 1.200 Kacheln mit einem Datenvolumen von insgesamt 562,5 MByte angefragt (Δ_τ = 480 KByte). Bei einem Zugriff auf den Speicherbereich SHC, bzw. T_{on} steigt aufgrund der größeren Zugriffsgranularität das zu ladende Speichervolumen in diesen Bereichen auf 585 MByte. Das entspricht zwölf Super-Kacheln mit einer Kapazität Δ_σ von je 48,7 MByte. Der Verschnitt υ_σ beträgt 3,84 %.

Wie zu erwarten, zeigt das Laden von Datenobjekten aus dem Speicherbereich SRC, eine ähnliche Performance, wie das Laden aus dem Bereich SR (Abbildung 3.39). Das liegt an der Tatsache, dass die beiden Bereiche nur aus logischer Sichtweise zwei disjunkte Teilmengen darstellen. Eine Unterscheidung erfolgt durch zusätzliche Metadaten. Auf physikalischer Ebene liegen die Daten in der gleichen Tabelle des konventionellen DBMS. Die geringfügig bessere Laufzeit bei einer Anfragebearbeitung aus dem Speicherbereich SRC, lässt sich durch ein etwas besseres Clustering der Datenobjekte auf dem Sekundärspeicher erklären. Die Datenobjekte liegen aufgrund des verwendeten Intra-Query-Scheduling direkt in Anfrageordnung auf der Festplatte. Es können Operationen zum Positionieren des Leskopfes beim Festplattenzugriff eingespart werden. Im Gegensatz dazu, werden die Datenobjekte im Speicherbereich SR in Einfügeordnung in der Datenbank abgelegt, was nicht unbedingt der Anfrageordnung entspricht.

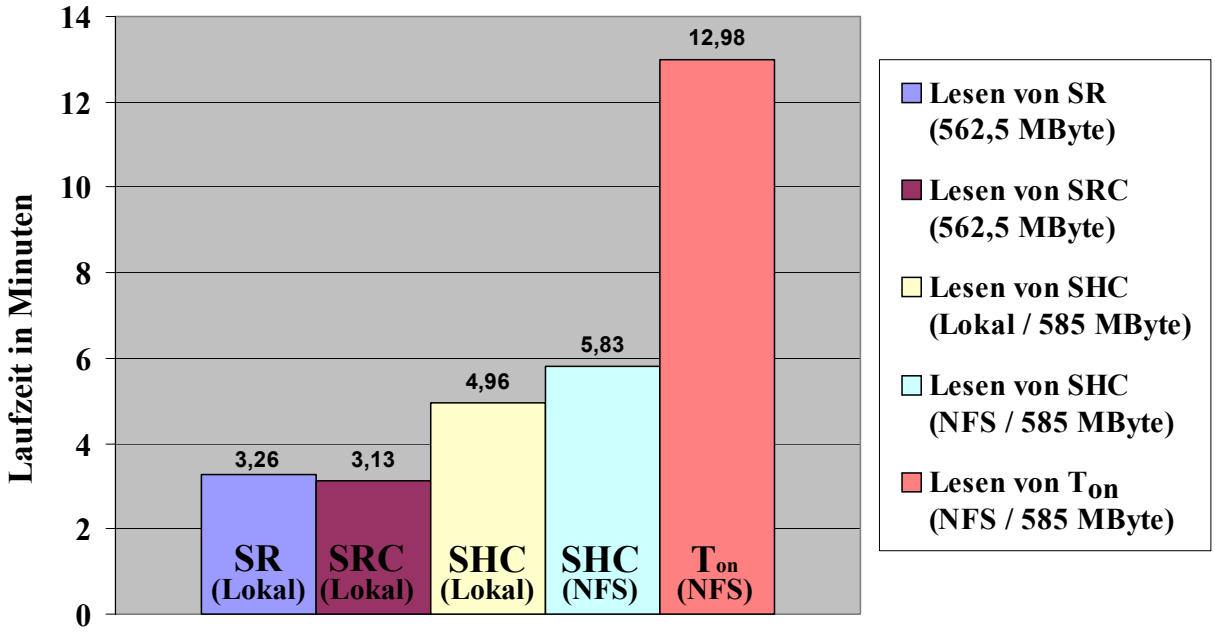


Abbildung 3.39: Laufzeit einer Bereichsanfrage bei unterschiedlichen Speicherebenen.

Beim Laden von Datenobjekten aus dem Cache-Speicher des HSM-Systems (SHC), sind zwei Möglichkeiten zu unterscheiden. Einerseits kann sich dieser Sekundärspeicherbereich auf dem lokalen Rechner befinden oder über NFS (Network File System) angebunden sein. Abbildung 3.39 zeigt beide Varianten. Bei einer NFS-Anbindung ist aufgrund der notwenigen Übertra-

gung über das Netzwerk eine höhere Laufzeit festzustellen. In der Messung erfolgte die Anbindung über ein 100 MBit *Ethernet* Netzwerk mit einer erreichten Transferrate von 8 MByte/s. Mit speziellen, im HPC-Bereich eingesetzten Technologien, ist nur noch ein sehr geringer Unterschied zwischen einem Zugriff auf eine lokale Festplatte oder zum Beispiel über *InfiniBand* (Infinite Bandwidth) zu erkennen. InfiniBand stellt eine E/A-Architektur zur Verfügung, mit der Übertragungsraten von 500 MByte/s bis zu 6 GByte/s erreicht werden [Mell02, Jni01]. Eine durchgeführte Messung mit einem InfiniBand-Cluster, hat eine Übertragungsrate von ca. 700 MByte/s bestätigt. Der bestehende Flaschenhals beim Netzwerk wird somit verringert, bzw. weitgehend beseitigt. Der zeitliche Mehraufwand beim Laden aus dem Speicherbereich SHC, gegenüber SRC, ist durch den Transfer der angeforderten Super-Kacheln in RasDaMan und das Extrahieren der enthaltenen Kacheln im DBMS zu erklären.
Befindet sich keines der angeforderten Datenobjekte in einem Cache-Bereich, so müssen die Super-Kacheln aufwändig von dem entsprechenden TS-Medium geladen werden. Diesen Sachverhalt zeigt der rechte Balken der Abbildung 3.39. Die Messung wurde mit dem Fünffach-Wechsler DLT-LXLS der Firma Overland Data mit einem integrierten DLT 4000 Laufwerk durchgeführt. Angebunden über NFS. Bei der Verwendung moderner Bandlauftechnologien, verringert sich die Laufzeit dieser Anfrage aufgrund der besseren Transferraten.

Der durchgeführte Vergleich der Ausführungszeiten, spricht klar für die Realisierung einer Caching-Hierarchie. Werden Datenobjekte im Speicherbereich SRC vorgehalten, so verbessert sich die Retrievalperformance um ca. 75,9 % (Speed-Up: 4,15) gegenüber dem Ladevorgang von tertiären Speichermedien. Die angeforderten Datenelemente sind nicht im HSM-Cache SHC zwischengespeichert. Der Speed-Up$_{SRC}$ kann unter Einbeziehung der Kosten für den Cacheverwaltung t_{SRC_Algo} wie folgt berechnet werden:

$$\text{Speed-Up}_{SRC} = \frac{t_{T_{on}}}{t_{SRC\_Algo} + t_{SRC}} = \frac{t_a^{T_{on}} + \sum_{i=1}^{N_\sigma}\left(t_{p_i}^{T_{on}} + \frac{\Delta_\sigma}{\Gamma^{T_{on}}} + t_{p_i}^{SHC} + \frac{\Delta_\sigma}{\Gamma^{SHC}}\right) + \sum_{j=1}^{N_\tau}\left(t_{p_j}^{SRC} + \frac{\Delta_\tau}{\Gamma^{SRC}}\right)}{t_{SRC\_Algo} + \sum_{j=1}^{N_\tau}\left(t_{p_j}^{SRC} + \frac{\Delta_\tau}{\Gamma^{SRC}}\right)}$$

wobei gilt: $sdom(\tau_j) \subseteq sdom(\sigma_i); sdom(\tau_j) \cap sdom(\sigma_i) \neq \varnothing; i \leq j$

Im Zähler werden alle Zeiten zum Laden der erforderlichen Datenobjekte von einem TS-Medium über den Speicherbereich SHC und SRC in den Primärspeicher P berücksichtigt. Allerdings wurde die Messung ohne Anlaufkosten ($t_a^{Ton} = 0$) durchgeführt. Bei anfallenden Anlaufkosten steigt der Speed-Up$_{SRC}$ weiter an. Der Nenner enthält die Zeitdauer t_{SRC_Algo} für die Verwaltung des Caches und das Laden der Datenobjekte vom Speicherbereich SRC in den Primärspeicher P. Wird die Cache-Trefferrate H_{SRC} mit einbezogen, so berechnet sich der Speed-Up$_{SRC}$ wie folgt:

$$\text{Speed-Up}_{SRC} = \frac{t_{T_{on}}}{t_{SRC\_Algo} + t_{SRC}\, H_{SRC} + t_{T_{on}}\left(1 - H_{SRC}\right)}$$

es gilt: $0 \leq H_{SRC} \leq 1$

Der hier gemessene Speed-Up$_{SRC}$ von 4,13 wird bei einer Cache-Trefferrate $H_{SRC} = 1$ erreicht. Das bedeutet, dass sich alle für die Anfrage notwendigen Datenobjekte im Speicherbereich SRC befinden und nichts von langsameren Speicherebenen nachgeladen werden muss. Die effektive Zugriffszeit (Access Time) t_a berechnet sich aus der Zeitdauer für die Verwaltung des Caches t_{SRC_Algo}, der Zeitdauer t_{SRC} und t_{Ton}, um Datenobjekte aus dem Cache-Bereich SRC oder von tertiären Speichermedien zu laden:

$$t_a = t_{SRC\_Algo} + t_{SRC}\ H_{SRC} + t_{T_{on}} \left(1 - H_{SRC}\right)$$

Neben der Nutzung des Cache-Bereiches SRC kann die effektive Zugriffszeit durch eine Reduzierung der Fehlzugriffszeit t_{Ton} weiter minimiert werden. Dies wird durch die Zwischenschaltung des HSM-Cache-Bereiches SHC erreicht. Somit ergibt sich für die effektive Zugriffszeit:

$$t_a = t_{SRC\_Algo} + t_{SRC}\ H_{SRC} + \left(t_{SHC\_Algo} + t_{SHC}\ H_{SHC} + t_{T_{on}} \left(1 - H_{SHC}\right)\right) \left(1 - H_{SRC}\right)$$

Der Performancegewinn bei einem Caching im Speicherbereich SHC, beträgt in Abhängigkeit von den Übertragungskosten (lokal Festplatte oder NFS) gegenüber dem Speicherbereich T_{on} zwischen 55,2 % (Speed-Up: 2,23) und 61,8 % (Speed-Up: 2,62). Die Laufzeitbeschleunigung beim Speicherbereich SRC, gegenüber SHC, erreicht einen Wert zwischen 36,7 % (Speed-Up: 1,58) und 46,2 % (Speed-Up: 1,86). Nach Erläuterung der Umsetzung der Caching-Hierarchie in *HEAVEN* folgen allgemeine Randbedingungen.

3.9.2 Allgemeine Randbedingungen bei *HEAVEN*

Eine der wichtigsten Bedingungen bei der Realisierung von Caching, ist die Erhaltung eines konsistenten Datenbestandes. Bei *HEAVEN* werden vorhandene Kopien auf allen Speicherebenen (SRC, SHC, T_{on}, T_{near}, T_{off}) direkt bei einer Änderungs- oder Löschoperation synchronisiert. Das bedeutet, es werden Änderungen sofort von der höchsten Speicherebene bis zur niedrigsten Speicherebene durchgeschrieben (Write-Through-Verfahren). Dadurch werden Konsistenzprobleme vermieden. Alle RasDaMan-Server greifen auf den gleichen Cache-Bereich SRC zu und lesen dadurch immer die aktuellen Werte. Cache-Kohärenz-Probleme werden vermieden. Weiterhin greift das Transaktionskonzept des verwendeten konventionellen DBMS, das zum Beispiel bei Änderungen durch die Verwendung von Sperren einen so genannten Dirty-Read verhindert. Das Verlagern des Cache-Bereiches SRC in das konventionelle DBMS (z.B. Oracle oder IBM/DB2), sichert durch die integrierte Transaktionsverwaltung die ACID-Eigenschaften bei Zugriffen auf die Datenbasis.

Als Nachteil gilt bei dem verwendeten Write-Through-Verfahren, dass der Performancegewinn durch einen schnellen Cache-Speicher bei Schreiboperationen entfällt. Das bedeutet, ein Änderungsvorgang dauert ebenso lang, wie bei einem System ohne Zwischenspeicher, da die Geschwindigkeit der langsamsten Speichereinheit ausschlaggebend ist. Bei *HEAVEN* bereitet

dies aus zweierlei Hinsicht keinerlei Probleme: Zum einen besteht im HPC-Bereich nur ein geringer Bedarf, Datensätze zu ändern oder zu löschen. In diesem Anwendungsgebiet verbleiben Originaldaten meist in ihrer ursprünglichen Form erhalten. Neue Simulationsergebnisse mit angepassten Parametern werden zusätzlich abgespeichert. Andererseits ist der Updatevorgang, ebenso wie der Export- und Löschvorgang, zwischen RasDaMan und dem HSM-System entkoppelt. Das bedeutet, dass RasDaMan die Änderungen im Primärspeicher (P), im Sekundärspeicher-Cache von RasDaMan (SRC) und im Sekundärspeicher-Cache des HSM-Systems (SHC) durchführen muss. Sobald die Änderungen den HSM-Cache erreicht haben, übernimmt das HSM-System automatisch die Anpassung der entsprechenden Speicherbereiche T_{on}, T_{near} oder T_{off}. Es reicht also aus, die Änderungen auf schnelle Sekundärspeicher zu propagieren.

Wie bereits im vorherigen Kapitel besprochen, sind im Rahmen von Änderungsoperationen bei Objekten mit arbiträrer Kachelung, zusätzliche konsistenzerhaltende Maßnahmen notwendig. Hier kann eine Mehrfachspeicherung von Kacheln τ in unterschiedlichen Super-Kacheln σ auftreten. Ist das der Fall, werden alle betroffenen Super-Kacheln der jeweiligen Speicherebenen angepasst. Wurde reguläre, ausgerichtete oder irreguläre Kachelung gewählt, sind diese Maßnahmen nicht erforderlich. Auch bei einem Update, das ein ganzes Objekt (MDD) betrifft, sind keine besonderen Maßnahmen erforderlich.

Die Einrichtung eines Cache, bietet nur unter der Annahme Vorteile, dass der erreichbare Performancegewinn durch einen schnelleren Speicherzugriff, den Aufwand für die Verwaltung des Ein- und Auslagern übertrifft. Bei steigender Aufenthaltsdauer eines Elementes im Cache, erhöht sich die Wahrscheinlichkeit, dass dieses Element bei einem erneuten Zugriff noch im Zwischenspeicher anzutreffen ist. Um eine gute Funktionsweise der Zwischenspeicherung zu garantieren, ist es notwendig, die Cachegrößen ausreichend zu dimensionieren. Dabei sind auch parallel arbeitende RasDaMan-Server zu beachten, die mehrere Anfragen gleichzeitig ausführen und auf den gleichen Cache-Bereich SRC zugreifen. Bei einem unterdimensionierten Cache-Bereich SRC, besteht die Möglichkeit einer zeitlich begrenzten Blockierung einer Anfrage, da der gesamte Cache durch Datenelemente konkurrierender RasDaMan-Server vollständig belegt ist. In diesem Fall können die für die Anfrage benötigten Datenelemente nicht in den Speicherbereich SRC geladen werden, solange keine anderen Datenobjekte zur Verdrängung freigegeben werden. Die Entstehung einer Verklemmung (Deadlock) wird durch die Verwendung von Freigabemarken (Flags) verhindert. Als grobe Richtlinie für eine Abschätzung einer minimalen Cachegröße Δ_{SRC} kann folgende Berechnung dienen:

$$\Delta_{SRC} = \frac{\sum_{i=1}^{n} \Delta_{q_i^{max}}}{L_{SRC}} = \frac{\sum_{i=1}^{n} \left(\Delta_{P_i} + \Delta_{Swap_i} \right)}{L_{SRC}}; \text{ es gilt } 0 < L_{SRC} \leq 1$$

Bei dieser Abschätzung, wird im Zähler das maximal mögliche Volumen einer Anfrage Δq^{max} für jede Instanz $i \in \{1, \ldots, n\}$ von RasDaMan berücksichtigt. Das maximale Volumen einer

Anfrage Δq^{max}, ist bei RasDaMan durch die Größe des Hauptspeichers Δ_P und des Swap-Speicher Δ_{Swap} begrenzt (Kapitel 3.7.1). Durch diese Abschätzung im Zähler wird sichergestellt, dass zumindest alle Instanzen von RasDaMan die für die Anfragebearbeitung benötigten Daten in den Cache-Bereich SRC laden können. Die beschriebenen, zeitlichen Verzögerungen werden vermieden. Durch die Angabe eines geeigneten Füllgrades L_{SRC} (Cache-Level) des Caches, für das maximal mögliche Anfragevolumen aller parallelen RasDaMan Instanzen, wird die Wirkungsweise des Caches weiter gesteigert. Es wird sichergestellt, dass bei der parallelen Bearbeitung von Anfragen nicht der komplette Cacheinhalt ausgetauscht wird, sonder auch ältere Datenobjekte im Cache verbleiben. Einzelne Anfragen können so den Cache-Bereich nicht monopolisieren. Der Cache-Level sollte kleiner 0,1 gewählt werden. Das bedeutet, dass bei einer Maximalbelastung des Caches bei paralleler Abarbeitung von Anfragen durch die RasDaMan-Server, der Cache-Bereich nur zu 10 % gefüllt ist. Bei Einhaltung der hier vorgestellten Abschätzung der minimalen Cache-Größe Δ_{SRC}, kann ein so genanntes Objektflattern oder Seitenflattern (Trashing) verhindert werden. Trashing tritt auf, wenn durch eine ungünstige Referenzfolge die Häufigkeit der Ersetzungsoperationen überproportional ansteigt. Das ist zum Beispiel der Fall, wenn viele Datenobjekte, die als nächstes benötigt werden, vor diesem Zugriff aus Platzmangel aus dem Cache entfernt werden und wieder neu geladen werden müssen.

Wie in den meisten Fällen gilt auch hier, dass der Cache-Bereich nie groß genug gewählt werden kann. Mit steigender Größe des Cache-Bereichs SRC, erhöht sich die Wahrscheinlichkeit, TS-Zugriffe einzusparen. Tabelle 3.3 gibt reale Steigerungsfaktoren der Speichergröße unterschiedlicher Speicherebenen mit typischen Speichervolumen im HPC-Bereich an.

Speicherübergang	Steigerungsfaktor	Typische Werte
P → SRC	100 – 200	10 GByte → 1 TByte
SRC → SHC	10 – 100	1 TByte → 20 TByte
SHC → T_{on}, T_{near}, T_{off}	10 – 1000	20 TByte → 500 TByte

Tabelle 3.3: Typische Größenordnung zwischen den unterschiedlichen Speicherebenen.

Aufgrund der großen Datenvolumen von multidimensionalen Array-Daten, ist das angegebenen Volumen für den Primärspeicher P von 10 GByte eher als mittlere oder sogar untere Schranke anzusehen. Bei der Pufferverwaltung ist ein besonderes Augenmerk auf das Auffinden von Datenobjekten im Cache zu legen. Es sind sehr effiziente Suchstrategien erforderlich, da diese Ereignisse bei jedem Datenzugriff auftreten. Generell sind zwei Möglichkeiten zu klassifizieren. Einerseits kann direkt in sequentieller Reihenfolge im Puffer-Bereich nach den entsprechenden Einträgen gesucht werden. Im Erfolgsfall (Hit) sind dabei durchschnittlich die Hälfte und bei Misserfolg (Miss) alle Einträge zu durchsuchen. Dieser Aufwand ist vor allem bei einem Cache mit vielen Einträgen nicht zu unterschätzen. Bei der zweiten Möglichkeit wird eine indirekte Suche über Hilfsstrukturen durchgeführt. Neben den eigentlichen Cache-

inhalten müssen zusätzliche Verwaltungsinformationen gespeichert werden. Dazu sind für einen Cache mit N Einträgen, die gleiche Anzahl an entsprechenden Strukturinformationen notwendig. Durch die Einführung einer sortierten Liste, wird bei Misserfolg (Miss) die Anzahl der Zugriffe auf N/2 reduziert. Bei der Verwendung einer balancierten Baumstruktur, verkürzt sich die Suchkomplexität auf $O(\log_m N)$, wobei m für die Anzahl der Einträge pro Knoten steht. Bei *HEAVEN* wurde die Information des Speicherortes auf der Basis von Super-Kacheln, bzw. Super-Knoten in den bestehenden R^+-Baum mit integriert. Das hat den Vorteil, dass keine zusätzliche Verwaltungsstruktur aufgebaut werden muss. Bei einem Indexzugriff steht die Information, ob sich ein Datenobjekt im Puffer befindet oder nicht, sofort ohne weiteren Aufwand zur Verfügung.

Nach der Beschreibung der allgemeinen Randbedingungen bei *HEAVEN*, folgen die für den Cache-Bereich SRC umgesetzten Verdrängungsstrategien. Der Cache-Bereich SHC unterliegt der Verwaltung des konventionellen HSM-Systems.

3.9.3 Verdrängungsstrategien

Wie bereits besprochen, befindet sich idealerweise jedes Datenobjekt zum Zeitpunkt des Zugriffs im Primärspeicher P oder im Cache-Bereich SRC. Da der Cache im Normalfall um Größenordungen (Faktor 100 bis 1000) kleiner ist, als der vorhandene tertiäre Hintergrundspeicher, können nicht alle Objekte vorgehalten werden. Aus diesem Grund werden Strategien zum Verdrängen von Datenobjekten aus dem Cache-Bereich eingesetzt. Die Absicht dabei ist, dasjenige Objekt als Verdrängungskandidat zu wählen, dessen Erwartungswert für eine Wiederverwendung minimal ist. Als Kriterium kann zum Beispiel die geringste Erhöhung der mittleren Antwortzeit herangezogen werden. Konventionelle Caching-Strategien zwischen Primär- und Sekundärspeicher, können prinzipiell auch auf das Caching zwischen Sekundär- und Tertiärspeicher übertragen werden. Eine wesentliche Besonderheit ist dabei jedoch der signifikante Unterschied in der resultierenden Antwortzeit bei TS-Zugriffen.

Grundsätzlich sind die realisierbaren Verdrängungsstrategien durch die beiden Verfahren OPT (Optimal [Bela66]) und RANDOM eingegrenzt. OPT ersetzt das Datenobjekt aus dem Puffer, dessen zeitlicher Abstand bis zur nächsten Referenz maximal ist. Da das zukünftige Referenzverhalten zur Entscheidungsfindung mit verwendet wird, ist es nicht realisierbar. Aus theoretischer Sicht beschreibt das OPT-Verfahren die obere Schranke in Bezug auf die Trefferrate H. Die untere Schranke wird durch RANDOM vorgegeben, das keinerlei Kenntnis über das Referenzverhalten ausnutzt. Abbildung 3.40 zeigt die Cache-Trefferrate H_{SRC} für die Fälle OPT, LRU, RANDOM und RANDOM ohne Lokalitätseigenschaften in Abhängigkeit von der Puffergröße Δ_{SRC}.

Je größer der Pufferbereich gewählt wurde, umso mehr Treffer (Hits) werden erreicht. Ist der Cache-Bereich Δ_{SRC} ebenso groß, wie der Hintergrundspeicher Δ_{Ton}, so beträgt die Trefferrate 100 Prozent. Ohne Lokalitätseigenschaften würde für RANDOM eine lineare Abhängigkeit der Trefferrate bestehen (Abbildung 3.40, RANDOM ohne Lokalität). Eine korrekt imple-

mentierte Verdrängungsstrategie, wie zum Beispiel das dargestellte LRU-Verfahren, liegt quantitativ zwischen OPT und RANDOM.

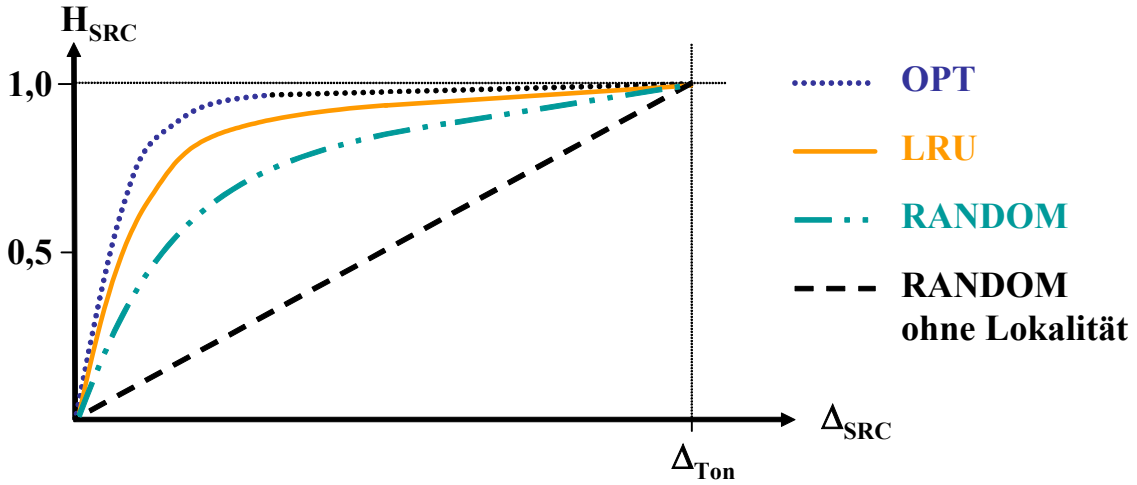


Abbildung 3.40: Grenzen der Trefferrate H_{SRC} gegeben durch OPT und RANDOM.

Die gemeinsame Herausforderung jedes Verdrängungsalgorithmus, ist die Entscheidung, welche Datenobjekte als nächstes aus dem Cache-Bereich verdrängt werden sollen und wie viel Aufwand für die Entscheidungsfindung des Verdrängungskandidaten aufgewendet werden darf. Die eigentliche Granularität der Einlagerung, bzw. der Verdrängung von Datenobjekten, basiert auf Super-Kacheln. Allerdings sind die im Cache-Bereich befindlichen Objekte in Kachel-Granularität abgespeichert. Das bedeutet, sobald ein Datenobjekt (Super-Kachel) vom Speicherbereich SHC in den Cachespeicher SRC eingelagert wird ($\sigma \in \Theta_{SRC}$), werden die in einer Super-Kachel (Containerobjekt) enthaltenen Kacheln extrahiert und gespeichert ($\tau_i \in \sigma$; $\tau_i \in \Theta_{SRC}$; $i \in \{1, \ldots, n\}$). Ist für das Einlagern neuer Objekte $\sigma_j \in I_\sigma$ mit $j \in \{1, \ldots, N_{I\sigma}\}$ einer Anfrage q nicht ausreichend Speicherplatz vorhanden, so muss vor dem Laden eine Menge von Verdrängungskandidaten $\Theta_{SRCvictims}$ ermittelt werden:

Definition 3.21: Menge $\Theta_{SRCvictims}$ der Verdrängungskandidaten

Menge $\Theta_{SRCvictims}$ aller durch einen Verdrängungsalgorithmus ermittelten Kandidaten (Opfer, Victims) $\sigma \in \Theta_{SRC}$. Es gilt: $\Theta_{SRCvictims} \subseteq \Theta_{SRC}$.

Die Kandidatenermittlung erfolgt auf der Basis von Super-Kacheln $\sigma \in \Theta_{SRCvictims}$. Bei einer anstehenden Verdrängung werden alle Kacheln ($\tau \in \sigma$) der Verdrängungskandidaten (Super-Kacheln) aus dem Cache-Bereich SRC gelöscht ($\sigma \notin \Theta_{SRC}$; $\tau \notin \Theta_{SRC}$).

In HEAVEN wurden mit FIFO (First-In-First-Out) und LRU (*Least-Recently Used*) zunächst zwei bekannte Verdrängungsstrategien realisiert [EH84]. Gerade das LRU-Verfahren hat sich sowohl in zahlreichen Betriebssystemen, als auch in der Pufferverwaltung von DBMS bewährt [HR99]. In einem weiteren Schritt ist das implementierte LRU-Verfahren um spezifi-

sche Besonderheiten von tertiären Speichermedien angereichert und zum H-LRU (*HEAVEN-LRU*) weiterentwickelt worden. Welches der genannten Verfahren bei HEAVEN zum Einsatz kommt, entscheidet der Systemverwalter durch einen entsprechenden Eintrag in der Konfigurationsdatei *TertiaryStorage.conf*. Eine nachträgliche Umstellung des gewählten Verfahrens, ist durch eine implementierte Überführungsfunktion ohne Probleme möglich. Der Cache-Bereich kann ohne Neuinitialisierung weiter verwendet werden.

Die einfachste Methode zur Ermittlung von Verdrängungskandidaten $\sigma \in \Theta_{SRCvictims}$, ist das FIFO-Verfahren. Als verketteter Ringpuffer realisiert, weist das Verfahren eine Zeitkomplexität von O(1) auf. Obwohl sich dieses Verfahren durch eine sehr einfache Verwaltung auszeichnet, ist es für den Praxiseinsatz aufgrund der hohen Fehlerrate nicht besonders gut geeignet. Ohne Beachtung der Frequentierung von Datenobjekten, wird immer das älteste Objekt aus dem Cache-Bereich entfernt. Ein in der Praxis sehr weit verbreitetes Verfahren, ist die Verdrängung von Kandidaten nach der LRU-Methode. Dies ist die konsequente Weiterentwicklung des LFU- (*Least-Frequently Used*) Verfahren. Beim LFU-Verfahren wird für jedes Objekt die Anzahl der Zugriffe registriert. Darauf basierend, wird die stationäre Zugriffswahrscheinlichkeit geschätzt und Kandidaten mit der geringsten Zugriffswahrscheinlichkeit verdrängt. Als nachteilig wirkt sich aus, dass sich die Schätzung nach langer Betriebszeit nur langsam an eine Veränderung des Zugriffsverhaltens anpasst. Das LRU-Verfahren beseitigt diesen Nachteil, indem die Reihenfolge der Zeitpunkte der letzten Zugriffe als Entscheidungskriterium für die Bestimmung der Verdrängungskandidaten herangezogen wird. Es wird das Objekt aus dem Cache-Bereich verdrängt, auf das am längsten nicht zugegriffen wurde.

Bei vielen Anwendungen, wie auch bei HEAVEN, ist man nicht nur primär an der Maximierung der Cache-Trefferrate H_{SRC} interessiert. Vielmehr steht die Minimierung der mittleren Antwortzeit für Zugriffe auf Datenobjekte, die auf TS-Medien gespeichert vorliegen, an erster Stelle. Verfahren wie LRU und LFU verwenden lediglich ein Kriterium zur Entscheidungsfindung von Verdrängungsopfer $\sigma \in \Theta_{SRCvictims}$. Diese Feststellung führt zu der Tatsache, dass beim Caching eine entstehende Erhöhung der Antwortzeit durch das Verdrängen von Objekten zusätzlich berücksichtig werden sollte. Objekte, deren Fehlen zu hohen Antwortzeiten führt, sollten eher im Cache gehalten werden, als Objekte, deren Fehlen zu geringeren Antwortzeiten führt. Dazu ist es nötig, mehrere Kriterien in die Entscheidungsfindung von Verdrängungskandidaten $\sigma \in \Theta_{SRCvictims}$ einfließen zu lassen. Diese Erkenntnis war der Anreiz für die Realisierung der speziell für TS-Zugriffe ausgelegten Verdrängungsstrategie H-LRU.

Verdrängungsstrategie H-LRU

Im Gegensatz zu Ersetzungsverfahren in Betriebssystemen, die oft durch Hardware-Bausteine direkt unterstützt werden, bieten eigene Entwicklungen größere Freiheiten, da alle Einzelheiten softwaretechnisch realisiert werden können. So besteht die Möglichkeit, Vorteile unterschiedlicher Ersetzungsstrategien zu kombinieren und für die speziellen Bedürfnisse anzupassen. Bei der im Rahmen von HEAVEN entwickelten Verdrängungsstrategie H-LRU gehen folgende Überlegungen in eine Entscheidungsfindung mit ein:

1. Nutzung der typischen Verfahrensweise der LRU-Methode, durch Einbeziehung der Zeitpunkte der letzten Referenzierung von Datenobjekten.
2. Beachtung der möglichen Varianz in Datenvolumen von Objekten, auf der Basis von Super-Kacheln in Zusammenhang mit der Transferrate des verwendeten TS-Systems.
3. Vermeidung der Verdrängung eines Datenobjektes, das sich gegenwärtig in einer Anfragewarteschlange befindet. Hierbei werden alle aktuell auszuführenden Anfragen der einzelnen parallelen RasDaMan-Server berücksichtigt, die auf dem gleichen Cache-Bereich SRC arbeiten. Es soll gelten: $\Theta_{SRCvictims} \cap I_\sigma = \varnothing$.
4. Sonderbehandlung von Datenobjekte, die durch einen Prefetching-Auftrag (RasQL-Anweisung: PREFETCH) in den Cache-Bereich SRC vorgeladen wurden.

Zu Punkt 1 (LRU-Verfahren): Als Basis für das in *HEAVEN* realisierte Verdrängungsverfahren H-LRU wird der LRU-Algorithmus verwendet. Der Zeitpunkt des letzten Zugriffes auf ein Datenobjekt, das sich im Cache-Bereich SRC befindet, wird als Entscheidungskriterium zur Schätzung der stationären Zugriffswahrscheinlichkeit herangezogen. Die Reihenfolge der Verdrängungszeitstempel (Eviction-Time-Stamps) E_{LRU} der einzelnen Datenobjekte ist für die Suche von Verdrängungskandidaten $\sigma \in \Theta_{SRCvictims}$ ausschlaggebend. Um Platz für neue Datenobjekte zu schaffen, werden Objekte mit den ältesten Zeitstempeln E_{LRU} aus dem Cache-Bereich entfernt.

Zu Punkt 2 (Datenvolumen und Transferrate): Wie bereits erwähnt, steht bei *HEAVEN* nicht unbedingt die Maximierung der Cache-Trefferrate H_{SRC} im Vordergrund. Sondern vielmehr die Minimierung der mittleren Antwortzeit bei Zugriffen auf Datenobjekte. Um dies zu erreichen, werden neben dem vom LRU-Verfahren verwendeten Zeitstempel E_{LRU} weitere Kriterien berücksichtigt, die zu einer besseren Verdrängungsentscheidung führen. Grundsätzlich sollen möglichst keine Datenobjekte verdrängt werden, deren Fehlen zu einer höheren Antwortzeit führt, als das Fehlen anderer Objekte. Entscheidende Einflussfaktoren für die Antwortzeit beim Laden von Objekten vom HSM-System sind vor allem das Datenvolumen eines Objektes Δ_σ und die Transferrate Γ des TS-Systems. Bei *HEAVEN* können die Datenobjekte, die in den Cache-Bereich SRC geladen werden, unterschiedlich groß sein. Durch die bei *HEAVEN* gegebene Möglichkeit, gleichzeitig mehrere HSM-Systeme als tertiären Hintergrundspeicher zu betreiben, ist die jeweilige Transferrate der eingesetzten TS-Systeme zu berücksichtigen. Der Einfluss unterschiedlicher TS-Technologien wirkt sich durch variierende Transferraten Γ auf die mittlere Zugriffzeit aus. Diese Überlegungen führen zu einem gewichteten LRU-Verfahren. Dabei geht der beim originalen LRU-Verfahren zur Entscheidung verwendete Zeitstempel E_{LRU} der einzelnen Datenobjekte, durch die Einbeziehung der Varianz in der mittleren Zugriffszeit, zu einem gewichteten Verdrängungsstempel $E_{H\text{-}LRU}$ über und wird wie folgt ermittelt:

$$E_{H\text{-}LRU} = E_{LRU}\, E_{\Delta_\sigma/\Gamma} = E_{LRU} \left(\frac{\Delta_\sigma}{\Gamma}\right)^{c_4 - 1};\ \text{es gilt: } E_{\Delta_\sigma/\Gamma} \geq 1;\ c_4 \geq 1$$

Zunächst betrachten wird das Verdrängungskriterium E_{LRU} des ursprünglichen LRU-Verfahrens. In der hier implementieren Version wird das Datenobjekt mit dem kleinsten E_{LRU}-Wert (= ältester Zeitstempel) als erster Kandidat aus dem Cache-Bereich verdrängt. Dieser Zeitstempel E_{LRU} wird mit dem Einflussfaktor $E_{\Delta\sigma/\Gamma}$ multipliziert. Der Wert $E_{\Delta\sigma/\Gamma}$ ist abhängig von der mittleren Zugriffszeit und somit von der Objektgröße Δ_σ und der Transferrate Γ. Bei steigender Zugriffzeit erhöht sich der Wert des Multiplikators $E_{\Delta\sigma/\Gamma}$. Das wiederum bewirkt eine Erhöhung des für die Kandidatensuche relevanten, gewichteten Verdrängungsstempel $E_{H\text{-}LRU}$.

Bei einem Datenobjekt mit einer höheren mittleren Zugriffszeit steigt der Wert $E_{H\text{-}LRU}$, bedingt durch einen höheren $E_{\Delta\sigma/\Gamma}$-Wert stärker an, als bei einem vergleichbaren Objekt mit geringeren Zugriffskosten. Das bewirkt, dass Datenobjekte mit höherer mittlerer Zugriffzeit in der Verdrängungsreihenfolge nach hinten verschoben werden. Als Verdrängungskandidat wird das Datenobjekt $\sigma \in \Theta_{SRCvictims}$ mit dem kleinsten $E_{H\text{-}LRU}$-Wert gewählt. Die Konstante[43] c_4 bestimmt dabei die Einflussnahme der mittleren Zugriffszeit. In Abbildung 3.41 wird der Einflussfaktor $E_{\Delta\sigma/\Gamma}$ in Abhängigkeit von c_4 und Δ_σ, bzw. von Γ und Δ_σ näher betrachtet. Beispielsweise errechnet sich für eine Transferrate Γ von 20 MByte/s, einem Datenvolumen Δ_σ von 260 MByte und einem Wert von 1,2 für die Konstante c_4 ein Multiplikator von $E_{\Delta\sigma/\Gamma}$ = 1,67.

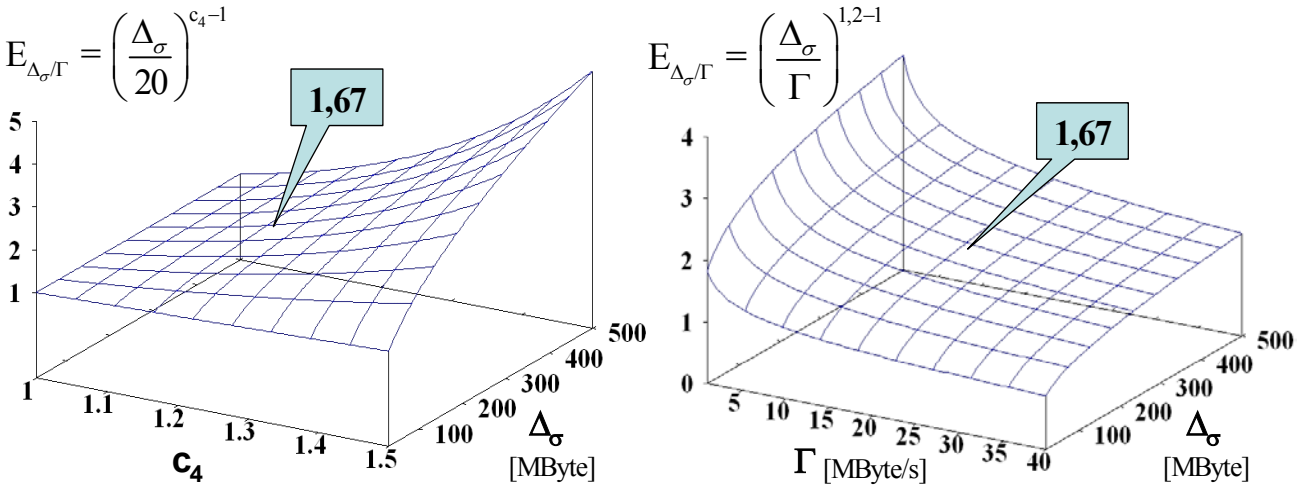


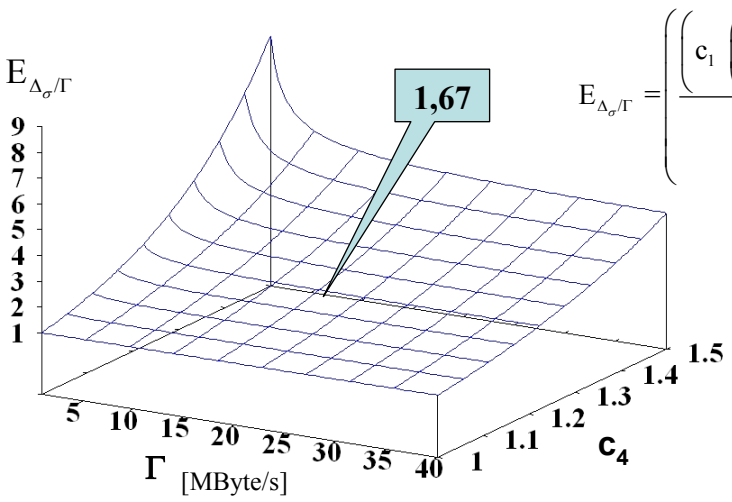
Abbildung 3.41: Betrachtung von $E_{\Delta\sigma/\Gamma}$ in Abhängigkeit von c_4 und Δ_σ, bzw. Γ und Δ_σ.

Wird die Konstante auf $c_4 = 1$ gesetzt, erhalten wir eine Verdrängung von Datenobjekten nach dem originalen LRU-Verfahren, ohne jegliche Einwirkung von Datenvolumen Δ_σ und Transferrate Γ. Bei steigender Konstante c_4 nimmt der Einfluss der mittleren Zugriffszeit, in Abhängigkeit von der Transferrate Γ und dem Datenvolumen Δ_σ, gegenüber dem LRU-

[43] Es wird die Indexnummer 4 verwendet, um eine Verwechslung mit den Konstanten c_1 bis c_3 bei der Berechnung der Super-Kachel-Größe (Kapitel 3.5.4) zu vermeiden.

Verfahren zu (Abbildung 3.41, linkes Diagramm). Anzumerken ist, dass die Wahl eines zu großen Wertes für c_4 die positive Eigenschaft des LRU-Verfahrens zerstört und die Einflussnahme durch die mittlere Zugriffszeit überhand nimmt. Gute Ergebnisse lassen sich bei dem voreingestellten Wert $c_4 = 1{,}2$ erreichen. Das rechte Diagramm in Abbildung 3.41 zeigt die Abhängigkeit des Multiplikators $E_{\Delta\sigma/\Gamma}$ von der Transferrate Γ und dem Datenvolumen Δ_σ einer Super-Kachel σ. Kleine Transferraten in Verbindung mit großen Datenvolumen ergeben einen hoher Wert für $E_{\Delta\sigma/\Gamma}$. In diesem Fall ist ein eventuelles Nachladen sehr zeitintensiv und daher wird dieses Datenobjekt verzögert verdrängt.

Aufgrund der automatischen Berechnung des Datenvolumens einer Super-Kachel σ (Kapitel 3.5.4) besteht eine direkte Korrelation zwischen dem Datenvolumen Δ_σ und der Transferrate Γ. Abbildung 3.42 zeigt die Abhängigkeit des Multiplikators $E_{\Delta\sigma/\Gamma}$ von Γ und c_4, unter Einbeziehung der Fromel zur automatischen Berechnung des Datenvolumens einer Super-Kachel Δ_σ. Bei einer angenommenen Datenübertragungsrate Γ von 20 MByte/s, errechnet sich eine Super-Kachel-Größe Δ_σ von ca. 260 MByte. Für $c_4 = 1{,}2$ ergibt sich ebenfalls ein Wert von 1,67 für den Multiplikator $E_{\Delta\sigma/\Gamma}$.



$$E_{\Delta_\sigma/\Gamma} = \left(\frac{\left(c_1 \left(\frac{t_p}{\Psi} - t_p \right) \Gamma + c_2 \, \Delta_\sigma^{min} \log\left(\frac{\Delta_\tau}{\Delta_\delta} \right) \right) \left(1 - c_3 \, \upsilon_\sigma \right)}{\Gamma} \right)^{c_4 - 1}$$

$$E_{\Delta_\sigma/\Gamma} = \left(\frac{0{,}8 \left(\frac{68}{0{,}8} - 68 \right) \Gamma + 20 \log\left(\frac{128}{8} \right)}{\Gamma} \right)^{c_4 - 1}$$

Abbildung 3.42: Abhängigkeit des Multiplikators $E_{\Delta\sigma/\Gamma}$ von Γ und c_4, unter Einbeziehung der automatischen Berechnung des Datenvolumens einer Super-Kachel Δ_σ.

Die rechte Seite der Abbildung 3.42 zeigt die resultierende Formel für die Bestimmung des Multiplikators $E_{\Delta\sigma/\Gamma}$, ohne und mit eingesetzten Werten. Da Abweichungen bei der tatsächlichen Super-Kachel-Größe gegenüber der berechneten Wunschgröße, besonders bei Rand-Super-Kacheln eines MDD α auftreten können, wird bei der Ermittlung des Multiplikators $E_{\Delta\sigma/\Gamma}$ die tatsächliche Größe einer Super-Kachel verwendet. Das wirkliche Datenvolumen lässt sich leicht aus der Domäne D und dem Basisdatentyp T der Super-Kachel σ ermitteln. Das Diagramm auf der linken Seite der Abbildung 3.42 zeigt die Abhängigkeit des Multiplikators $E_{\Delta\sigma/\Gamma}$ von der Transferrate Γ und der Konstanten c_4. Es ist gut zu erkennen, dass bei

kleinen Datenübertragungsraten der Wert von $E_{\Delta\sigma/\Gamma}$ ansteigt. Das führt zu einer verzögerten Verdrängung dieses Datenobjektes $\sigma \in \Theta_{SRCvictims}$. Bei Datenobjekte (Super-Kacheln), die das gleiche Datenvolumen Δ_σ aufweisen und vom gleichen TS-System mit identischer Transferrate Γ stammen, erfolgt die Verdrängungsentscheidung alleine durch den Zeitpunkt des Eintreffens in den Cache-Bereich SRC. In diesem Fall dominiert bei der Kandidatenermittlung der E_{LRU}-Faktor.

Zu Punkt 3 (Anfragewarteschlange): Die implementierte H-LRU-Strategie vermeidet bei der Bestimmung von Verdrängungskandidaten $\sigma \in \Theta_{SRCvictims}$ die Auswahl von Datenobjekten einer aktuellen Anfragewarteschlange ($\Theta_{SRCvictims} \cap I_\sigma = \varnothing$). Es werden alle aktuell auszuführenden Anfragen der einzelnen parallelen RasDaMan-Server berücksichtigt, die auf dem gleichen Cache-Bereich SRC arbeiten. Realisiert wird dies über einen mitgeführten Zählereintrag (Counter-Flag). Sobald ein Datenobjekt, das sich im Cache-Bereich SRC befindet, von einem RasDaMan-Server angefragt wird, erfolgt eine Inkrementierung dieses Zählereintrages. Fordert ein weiterer RasDaMan-Server dieses Datenobjekt an, so erhöht sich der Zählerstand weiter. Solange der Zählerstand eines Datenobjektes größer als Null ist, wird es von mindestens einer Anfrage benötigt und kommt als Verdrängungskandidat nicht in Frage ($\sigma \notin \Theta_{SRCvictims}$). Wird ein Datenobjekt nicht mehr von einer Anfrage verwendet, so wird der Zählerstand dekrementiert. Besteht keine aktuelle Referenz auf einem Datenobjekt (Zählerstand = 0) kann eine Verdrängung aus dem Cache-Bereich SRC durchgeführt werden. Bei jeder auftretenden Referenz auf ein Datenobjekt, wird neben der Zählerverwaltung, entsprechend dem implementierten LRU-Verfahren, in Abhängigkeit von der Objektgröße, das Verdrängungskriterium angepasst. Die Realisierung eines Verdrängungsalgorithmus mit der Einbeziehung der Anfragewartschlange in die Entscheidungsfindung, verhindert die frühzeitige Entfernung eines Datenobjektes aus dem Cache-Bereich und das sonst notwendige, zeitintensive Nachladen aus einer langsameren Speicherhierarchie. Ein bei ungünstiger Referenzfolge eventuell auftretendes Objektflattern (Trashing) durch einen überproportionalen Anstieg von Ersetzungsoperationen, kann durch diese Maßnahme verhindert werden.

Zu Punkt 4 (Prefetching): Bei *HEAVEN* erfolgt ein bedarfsgerechtes Laden von Datenobjekten und kein automatisches Prefetching (Vorladen). Durch Prefetching sollen Datenobjekte, die mit hoher Wahrscheinlichkeit in naher Zukunft referenziert werden, schon vorab in den Cache-Bereich geladen werden. Wie im vorherigen Kapitel beschrieben, wurde auf ein automatisches Prefetching verzichtet und ein manuelles Vorladen von Datenobjekten eingeführt. Durch die Erweiterung der Anfragesprache von RasDaMan RasQL um die PREFETCH-Anweisung, sind Anwender in der Lage, Datenobjekte zeitbasiert in den Cache-Bereich SRC vorzuladen. Bei der Realisierung des Verdrängungsalgorithmus H-LRU wurde diese Möglichkeit berücksichtigt, indem die in den Cache-Bereich SRC vorgeladenen Datenobjekte σ normalerweise nicht frühzeitig vor der erstmaligen Verwendung, verdrängt werden ($\sigma \notin \Theta_{SRCvictims}$). Dies wird durch eine spezielle Kennzeichnung der entsprechenden Datenobjekte erreicht. Die Möglichkeit eines mehrfach vorgeladenen Datenobjektes wird ebenfalls berücksichtigt. Um allerdings keine Blockierung der aktuellen Anfragebearbeitung durch vorgeladene, markierte Datenobjekte im Cache-Bereich zu erhalten, werden Ladeaufträge von gerade

laufenden Anfragen vorrangig behandelt. In diesem Fall wird von der Markierung „nicht zur Verdrängung freigeben“ abgesehen und die ältesten markierten Datenobjekte nach dem in HEAVEN implementierten FIFO-Verfahren aus dem Cache-Bereich verdrängt. Nach der ersten Referenzierung von markierten Datenobjekten, werden diese in den normalen Cache-Betrieb eingereiht und entsprechend dem verwendeten H-LRU-Verfahren behandelt. Wird das PREFETCH-Konstrukt um eine Startzeit und eine Verweildauer ergänzt, so ist bei der Einreihung der markierten Datenobjekte in den herkömmlichen Verdrängungsablauf eine etwaige Verweildauer zu berücksichtigen.

Eine weitere Verbesserung des vorgestellten H-LRU-Verfahrens, kann durch eine Berücksichtigung der letzten k Referenzpunkte erfolgen. Da das auf LRU basierende Verfahren, nur einzelne Zeitpunkte zur Schätzung der stationären Zugriffswahrscheinlichkeit heranzieht, unterliegt die Schätzung einer gewissen Varianz. Um diese Varianz abzuschwächen, wurde das LRU-k-Verfahren entwickelt [OOW93]. Hier werden zur Schätzung die Zeitpunkte der letzten k Zugriffe verwendet. Durch die Bestimmung der mittleren Zeitabstände zwischen den letzten k Referenzen, wird eine Art Referenzdichte zur Entscheidungsfindung von Verdrängungsopfern herangezogen. Allerdings ist für dieses Verfahren ein zusätzlicher Aufwand für die Verwaltung und Speicherung der Historie der k letzten Referenzen erforderlich. Außerdem muss eine Sonderbehandlung für Datenobjekte mit weniger als k Referenzen durchgeführt werden. Um diesen Zusatzaufwand möglichst gering zu halten, wird bei [OOW93] vorgeschlagen, nur die letzten beiden Referenzen zu berücksichtigen. LRU-2 zeigt ähnlich gute Ergebnisse wie bei höheren k-Werten. Dadurch unterliegt die Schätzung einerseits nicht mehr so starken Schwankungen und ist andererseits flexibel genug, um schnell auf langfristige Änderungen im Zugriffsverhalten zu reagieren.

3.9.4 Erreichtes Optimierungspotential

Das Zwischenspeichern von Datenobjekten auf einer schnelleren Speicherebene, hat vor allem das Ziel, zeitaufwändige Zugriffe auf TS-Medien zu vermeiden. Aus diesem Grund wurde in HEAVEN eine mehrschichtige Caching-Hierarchie, bestehend aus dem Primärspeicher P, dem Cache-Bereich SRC und dem HSM-Cache-Bereich SHC realisiert. Durch die mehrstufige Caching-Hierarchie, wird die Wahrscheinlichkeit gesteigert, dass sich angefragte Datenelemente in einem dieser Cache-Bereiche aufhalten.

Weiterhin wurde bei HEAVEN mit dem H-LRU-Verfahren eine speziell für angebundene TS-Systeme optimierte Verdrängungsstrategie entwickelt. Hier steht vor allem die Minimierung der mittleren Antwortzeit für Zugriffe auf Datenobjekte, die sich auf TS-Medien befinden, an erster Stelle. Es wird beim Caching eine entstehende Antwortzeiterhöhung durch das Verdrängen von Datenobjekten zusätzlich berücksichtig. Objekte, deren Fehlen zu hohen Antwortzeiten führt, werden eher im Cache gehalten, als Objekte, deren Fehlen zu geringeren Antwortzeiten führt. Um das zu erreichen, wurde das ursprüngliche LRU-Verfahren, das lediglich ein Kriterium zur Entscheidungsfindung von Verdrängungskandidaten verwendet, um weitere Entscheidungshilfen angereichert. Es werden Unterschiede in der mittleren

Zugriffzeit, durch die Einbeziehung der Datenvolumen von Objekten in Zusammenhang mit der entsprechenden Datenübertragungsrate des verwendeten TS-Systems beachtet. Weiterhin werden keine Datenobjekte aus dem Cache-Bereich SRC entfernt, die von einer aktuell ausgeführten Anfrage eines der parallel arbeitenden RasDaMan-Server benötigt werden. Außerdem wird vermieden, Datenobjekte aus dem Cache-Bereich zu verdrängen, die durch einen Prefetching-Auftrag vorgeladen wurden. Gerade die Kombination dieser Entscheidungskriterien bringt Vorteile in Bezug auf die Minimierung der Zugriffszeit, gegenüber herkömmlicher Verdrängungsstrategien.

3.10 Object-Framing für Array-Daten

In diesem Abschnitt wird Object-Framing als ein neues Verfahren vorgestellt. Es ermöglicht, multidimensionale Bereichsanfragen weit genauer zu spezifizieren, als es in RasDaMan möglich war. Bisher waren der Zugriff und auch die Analyse auf multidimensionale Hyperquader beschränkt. Mit Object-Framing wurde diese Beschränkung aufgehoben. Eine Spezifizierung von Polygonen in einer zweidimensionalen Ebene, die in allen Dimensionen abgebildet werden, erlaubt einen besser definierten Zugriff. Diese detaillierte Beschreibung des Anfragebereiches führt zu einer vielfältigen Erweitung der Möglichkeiten im Bereich der Datenanalyse. Darüber hinaus lassen sich in vielen Fällen deutliche Performancegewinne erreichen. Durch ein reduziertes Datenvolumen, das bei Anfragen geladen werden muss gehört Object-Framing, entsprechend dem Optimierungspotential für TS-Zugriffe aus Kapitel 3.4, in die Kategorie „Vermeidung von TS-Zugriffen“. Die Entwicklung und Implementierung von Object-Framing zum adaptiven Zugriff auf das multidimensionale DBMS RasDaMan erfolgte in der Diplomarbeit [Lavs03], die im Rahmen dieser Dissertation betreut wurde.

Aus der Sicht, der in RasDaMan realisierten Operationen, ist die Methode des Object-Framing den geometrischen Operationen zuzuordnen (Kapitel 2.5.5.1). Geometrische Operationen entsprechen einer Abbildung (<D, T> → <D', T>) eines MDD auf ein anders MDD. Diese Operationen verändern weder den Typ, noch den Wert der Zellen eines MDD. Geometrische Operationen bewirken ausschließlich eine Änderung bezüglich der Domäne D eines MDD. Object-Framing ermöglicht ebenso, wie die Trimming-Operation, eine Einschränkung dieser Domäne. Bei Object-Framing kann der Anfragebereich weit flexibler spezifiziert werden und ist nicht wie bei Trimming, zwingend auf multidimensionale Hyperquader beschränkt. So generalisiert Object-Framing die Trimming-Operation und wird definiert als:

Definition 3.22: Object-Framing (rahmenbasiertes Zuschneiden) eines MDD α

Durch diese Operation wird die Domäne D eines MDD α auf eine Teildomäne D_F eingeschränkt, die nicht notwendigerweise eine rechteckige Form aufweisen muss. Die Dimensionalität d und der Basisdatentyp T des MDD bleiben erhalten. Durch die Definition eines multidimensionalen Rahmens (Frame) F wird die Beschränkung auf multidimensionale Hyperquader aufgehoben. Der Rahmen F wird definiert durch eine Menge von Polygonen P[44],

[44] Das Polygon ist in diesem Kontext als ein einfaches, nicht überschneidendes Polygon definiert.

die durch eine Menge räumlicher Punkte p in einer zweidimensionalen Ebene aufgespannt werden:

frame $_F$: <D, T> → <D_F, T>, α → frame(α, D_F)

mit D = [l_1:h_1, ..., l_d:h_d] und $D_F := \bigcap_{i=1}^{n} P_i[j, k]$

wobei gilt: $P_i[j, k] := \{ p_1[j, k]; \ldots; p_m[j, k] \}$,

für alle $i \in \{1, ..., n\}$ und $j, k \in \{1, ..., d\}; j \neq k; p[j, k] \in D; p[j, k] \in \mathbb{Z}^2$

Die zweidimensionalen Bereichsbeschreibungen einer Menge von Polygonen, werden in den d-dimensionalen Raum zur Spezifikation des Anfragebereiches fortgesetzt. Durch diesen multidimensionalen Rahmen F können komplexe, zugeschnittene Bereiche entstehen. Zur Definition eines Rahmens in einer Ebene, aufgespannt durch die Koordinatenachsen der Dimensionen[45] j und k, wird folgende Notation verwendet:

FRAME <j, k> $[p_1[j], p_1[k]; ...; p_m[j], p_m[k]]$

Dabei wird durch eine Zuordnungsvorschrift die Menge der multidimensionalen Punkte auf eine zweidimensionale Menge abgebildet. Der vierdimensionale Zellpunkt (23, 120, 5, 733) wird durch die Angabe der beiden Koordinatenachsen, durch den Ausdruck <2, 4> auf den Punkt (120, 733) abgebildet. Diese Abbildung wird im Folgenden als zweidimensionale Projektion bezeichnet. Innerhalb dieser 2D-Projektion wird eine Grenzlinie (Polygon) definiert, die den Anfragebereich umrandet. Ein Polygon wird durch eine Menge von Punkten $p_i[j, k]$ im zweidimensionalen Koordinatensystem angegeben. Diese werden in aufeinander folgender Ordnung durch gerade Linien (Kanten, engl.: edges) e miteinander verbunden. Um ein Polygon zu erhalten, wird der letzte spezifizierte Punkt mit dem ersten Punkt verbunden. Das entspricht einem zyklischen, gerichteten Graphen mit folgenden Eigenschaften:

1. Der gerichtete Graph hat m Kanten
2. Für alle Kanten $e_i \in \{e_1, e_2, \ldots, e_{m-1}\}$ gilt: Kante e_i hat Knoten (Punkt) p_i als Anfangsknoten und Knoten p_{i+1} als Endknoten ($i \in \{1, ..., m\}$).
3. Die Kante e_m hat Knoten p_m als Anfangsknoten und p_1 als Endknoten.

Es können beliebig viele Begrenzungslinien bestimmt werden. Jede liegt in einer durch zwei Koordinatenachsen spezifizierten Ebene. Mit der Methode des Object-Framing kann durch Angabe eines rechteckigen Polygons eine Trimming-Operation nachgebildet werden. So entspricht z.B. die Object-Framing Operation FRAME <1,2> [30, 10; 30, 20; 40, 20; 40, 10] der Trimming-Operation [10:20, 30:40]. Object-Framing ist als Generalisierung der Trimming-Operation zu verstehen. Die RasDaMan-Anfragesprache RasQL, wurde um das ent-

[45] Bei der Anfragesprache RasQL verläuft der Wertebereich der Dimensionen von 0 bis d – 1, nicht wie hier angegeben von 1 bis d.

sprechende Konstrukt erweitert. Die Funktionsweise von Object-Framing wird nun anhand der folgenden Beispielanfrage näher erläutert:

```
SELECT  a[ FRAME <1, 2> [ 60, 0;   10, 63;   50, 63;   100, 0 ],
                        <3, 2> [   0, 0;     0, 63;   59, 63 ] ]
FROM  climate_coll  as  a
```

Als Beispiel dient der bereits bekannte dreidimensionale Datensatz einer Klimasimulation des Deutschen Klimarechenzentrums, Hamburg. Entsprechend der Anfrage, wird die Domäne des ursprünglichen Objektes durch zwei definierte Rahmen in Form eines Parallelogramms und eines Dreiecks in unterschiedlichen Ebenen beschnitten. Das schrittweise Vorgehen der Operation Object-Framing ist in Abbildung 3.43 dargestellt.

Die Nummer 1 der Abbildung 3.43 zeigt das ursprüngliche MDD[46] mit der Domäne [0:119, 0:63, 0:127]. Um einzelne Teilschritte besser zeigen zu können, wird dieses Objekt als ein abstrakter, dreidimensionaler Würfel dargestellt (Nummer 2). Die beiden schraffierten Ebenen <1, 2> und <3, 2>, welche durch die Dimensionen 1 und 2, bzw. 3 und 2 aufgespannt werden, bilden die Grundlage der Anfrage mittels Object-Framing. Die in der RasQL-Anfrage spezifizierten zweidimensionalen, gerichteten, zyklischen Graphen werden nun in einem zweidimensionalen Koordinatensystem betrachtet, zusammen mit der 2D-Projektion der MDD-Domäne. In der Ebene <1, 2> wird ein Rahmen in Form eines Parallelogramms mit den Eckpunkten p1(60, 0), p2(10, 63), p3(50, 63), p4(100, 0) gezeichnet (Nummer 3). Diese zweidimensionale Grenzlinie wird in den d-dimensionalen Raum zur Spezifikation des Anfragebereiches fortgesetzt (Nummer 5). Alle Zellpunkte eines MDD, die von der 2D-Projektion auf einen Bildpunkt abgebildet wurden und innerhalb des von der zweidimensionalen Grenzlinie umrandeten Gebietes liegen, befinden sich im Anfragebereich. Analog dazu wird in der Ebene <3, 2> ein Rahmen in Form eines Dreieckes mit den Eckpunkten p5(0, 0), p6(0, 63), p7(59, 63) festgelegt (Nummer 4) und in den d-dimensionalen Raum fortgesetzt (Nummer 6). In dieser FRAME-Operation werden mehrere Grenzlinien spezifiziert. Deshalb müssen die jeweiligen Bildpunkte eines multidimensionalen Punktes bei allen 2D-Projektionen innerhalb dieses zweidimensionalen Bereiches liegen, um in den Anfragebereich zu fallen. Daraus resultiert der unter Nummer 7 dargestellte Anfragebereich mit der eingeschränkten Domäne D_F. Als Ergebnis unserer Anfrage mittels Object-Framing, erhalten wir das Teilobjekt Nummer 8.

[46] Simulation zur Klimaerwärmung (Treibhauseffekt) des Deutschen Klimarechenzentrums entsprechend dem IS92a Szenarium des IPCC'92 (Intergovernmental Panel on Climate Change) mit einem jährlichen Anstieg der Konzentration des Treibhausgases CO_2 um 1 %. Zu erkennen ist die atmosphärische Temperaturverteilung der Erde, zwei Meter über der Oberfläche, in einer Zeitspanne von 10 Jahren (2000 – 2009).

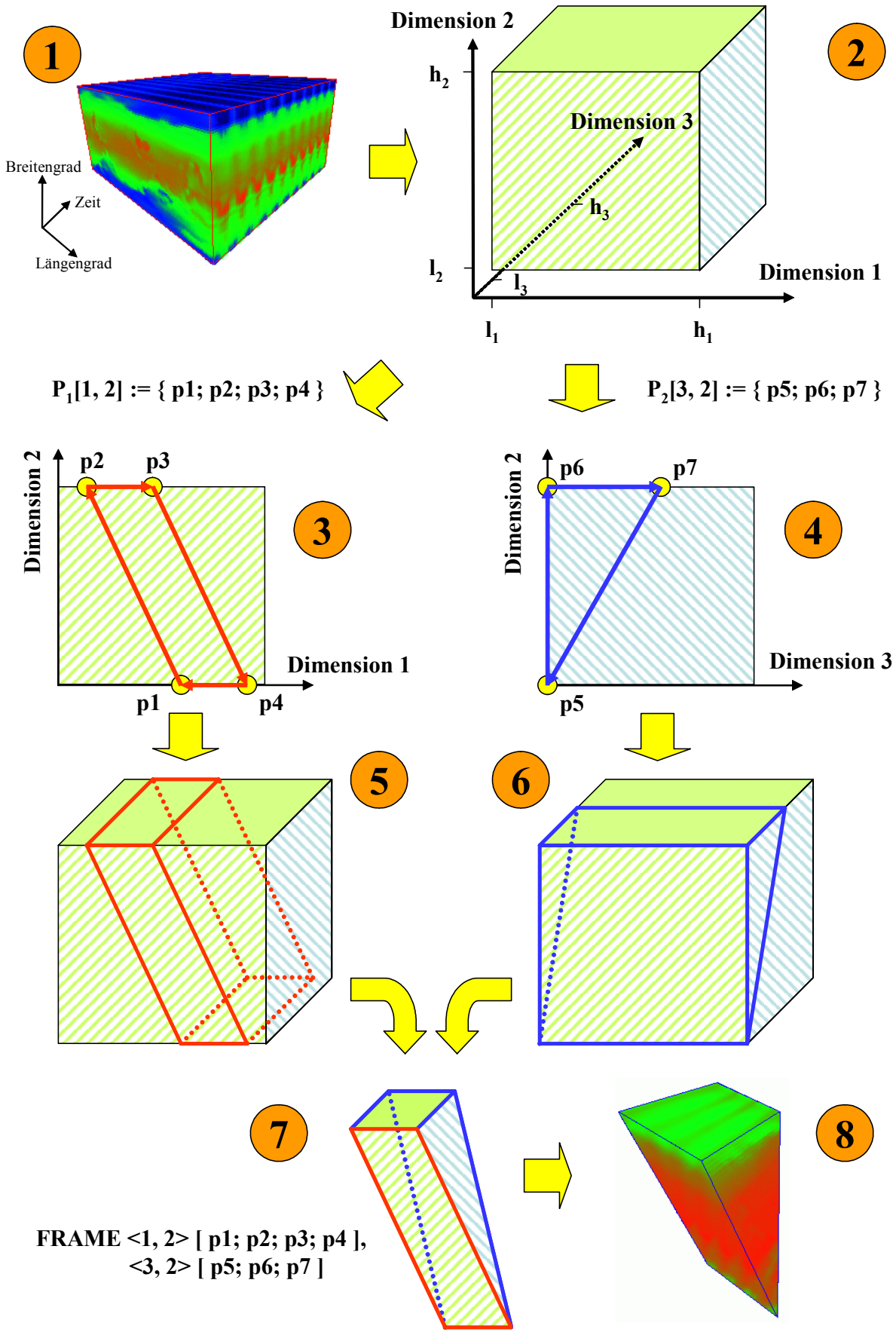


Abbildung 3.43: Object-Framing am Beispiel eines dreidimensionalen MDD.

Die Entscheidung über die Lage einer Kachel, hinsichtlich des definierten Anfragebereiches, kann durch die Betrachtung der 2D-Projektion der Domäne einer Kachel und der spezifizierten Grenzlinie erfolgen. Zur Bestimmung der relevanten Kacheln müssen vier Fälle unterschieden werden (Abbildung 3.44).

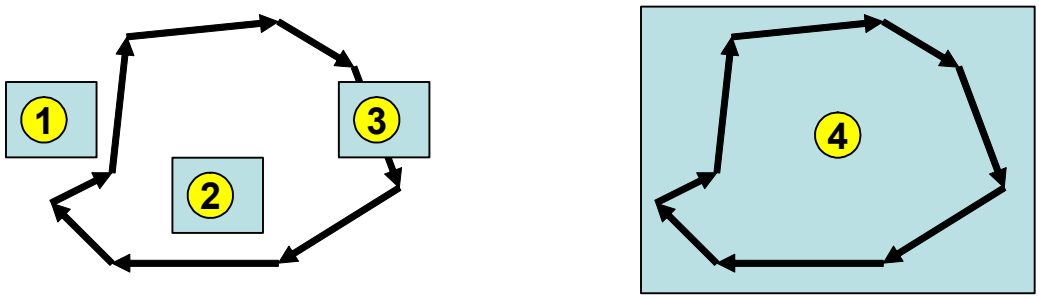


Abbildung 3.44: Fallunterscheidungen bei der Bestimmung der resultierenden Kacheln.

Liegt eine Kachel (Fall 1) außerhalb des gerichteten Graphen, so ist sie nicht Gegenstand der Anfrage und wird nicht weiter berücksichtigt. Ist eine Kachel (Fall 2) vollständig in der Grenzlinie enthalten, so kann sie ohne weitere Maßnahme für die Anfrage verwendet werden. Der Fall 3 und Fall 4 sind Sonderfälle, da eine Kachel nur partiell im Rahmen enthalten ist. In diesem Fall ist eine Bitmaske zu erstellen, die es ermöglicht, die nicht benötigten Bereiche (Zellen) für die Anfragebearbeitung auszublenden. Die Grenzlinie selbst ist Bestandteil des Anfragebereiches. Bei den hier angestellten Überlegungen ist noch zu berücksichtigen, ob bei einer Anfrage mehrere Grenzlinien angegeben wurden. In diesem Fall kann eine Kachel hinsichtlich der ersten Grenzlinie Gegenstand der Anfrage sein, allerdings entsprechend der Definition 3.22 durch eine weitere Grenzlinie aus dem Anfragebereich fallen. Weiterführende Betrachtungen und Implementierungsdetails sind bei [Lavs03] zu finden.

In seiner Anwendbarkeit im Zusammenspiel mit anderen Operationen, ist Object-Framing mit der in RasDaMan bereits vorhandenen Trimming-Operation vergleichbar. So werden beliebig viele Frame-Operationen, in Zusammenhang mit anderen geometrischen Operationen, sowie mit induzierten Operationen, Zelloperationen und Aggregatoperationen voll unterstützt. Object-Framing ist besonders in Zusammenhang mit Aggregatoperationen, wie zum Beispiel der Extremwert-, Summen- oder Durchschnittsbildung, von besonderer Bedeutung. Es können dadurch die Werte eines komplexen Anfragebereiches effizient und vor allem korrekt aggregiert werden. Um eine nicht auf einen Hyperquader basierende Aggregatoperation durchzuführen, musste in RasDaMan bisher das MDD mit einer Maske überlagert werden. Diese Maske war ebenfalls ein MDD mit gleicher Domäne und Größe, wie das ursprüngliche MDD. Die für die Anfrage relevanten Bereiche der Maske wurden auf den Wert 1 und die auszublendenden Bereiche auf den Wert 0 gesetzt. Eine Multiplikation des MDD mit der Maske ergab ein resultierendes MDD, mit den Werten des ursprünglichen MDD an den relevanten Stellen und dem Wert 0 an allen übrigen. Das Problem dieser Methode ist, dass der gesetzte Wert 0 der ausgeblendeten Bereiche einen gültigen Zellwert darstellt, der auch in nicht ausge-

blendeten Bereichen auftreten kann. RasDaMan unterstützt in diesem Fall nicht, die ausgeblendeten Bereiche, z.B. auf NULL[47] zu setzten. Auf diese Art und Weise konnten somit Summenbildungen und bedingt Extremwertbestimmungen für Anfragebereiche, die nicht auf einen Hyperquader beschränkt sind, durchgeführt werden. Eine Extremwertbestimmung liefert allerdings nur das korrekte Ergebnis, wenn kleinere, bzw. größere Werte in dem Objekt vorhanden sind, als der durch die Multiplikation gesetzte Wert 0. Anderenfalls wird 0 als „nicht korrektes" Ergebnis zurückgegeben. Eine Durchschnittsberechnung kann auf diese Weise nie korrekt durchgeführt werden, da die mit 0 besetzten, ausgeblendeten Bereiche mit in die Berechnung einfließen. Operationen mittels Object-Framing schließen diese Lücke der Anfragebearbeitung und lösen die hier genannten Probleme korrekt und effizient. So sind erweiterte Möglichkeiten der Datenanalyse in RasDaMan geschaffen worden. Als weiterer, positiver Effekt ist zu nennen, dass die früher benötigten Masken nicht mehr vorgehalten werden müssen, die das gleiche Speichervolumen wie das ursprüngliche Objekt aufweisen. Für jeden unterschiedlichen Anfragebereich mussten eigenständige Masken generiert und gespeichert werden.

Neben den erweiterten Analysemöglichkeiten bietet Object-Framing sehr interessante Konzepte, die für ein effizientes Datenretrieval von TS-Medien ausgenutzt werden können. So kann durch Object-Framing das zu ladende Datenvolumen reduziert werden. Bei der Beschränkung der Anfragebereiche auf Hyperquader, mussten bisher alle Kacheln geladen werden, die zumindest partiell im Bereich dieses Hyperquaders liegen. Dabei konnte nicht verhindert werden, dass ein großer Teil der Lade- und Berechnungszeiten für Bereiche aufgewendet wurden, die für den Anwender nicht von Bedeutung sind. Durch die Möglichkeit einer genaueren Definition des Anfragebereiches, ist ein enormes Potential gegeben, Lade- und Berechnungszeit einzusparen. Das zu ladende Datenvolumen beschränkt sich auf die Kacheln, die innerhalb der Begrenzungslinie liegen, bzw. von dieser geschnitten werden. Der Verschnitt (Definition 2.11) wird dadurch minimiert. Dieser Zusammenhang soll am Beispiel der im August 2002 an der Elbe stattgefundenen Jahrhundertflut verdeutlicht werden. Das Satellitenbild[48] (Abbildung 3.45) zeigt die Überschwemmungen der Elbe nordwestlich der Stadt Dresden. Weiterhin ist durch einen roten Rahmen, der für die Anfrage relevante Uferbereich der Elbe dargestellt.

Abbildung 3.46 zeigt das auf die Speicherebene reduzierte Satellitenbild mit den enthaltenen Kacheln und dem Anfragebereich. Anhand dieser Darstellung kann leicht der Unterschied des zu ladenden Datenvolumens im Fall einer hyperquaderbasierten Anfragebearbeitung (Trimming-Operation) und einer Anfragebearbeitung mittels Object-Framing verdeutlicht werden. Bei der hyperquaderbasierten Variante sind alle Kacheln (525 Stück) zu laden, die innerhalb des gestrichelten Rechteckes liegen. Im Gegensatz dazu sind bei Object-Framing nur die farbig markierten 95 Kacheln zu laden. Das entspricht in dieser Beispielanfrage einer Einsparung im Datenvolumen, das geladen werden soll, um den Faktor 5,5.

[47] NULL bezeichnet in der EDV den Wert eines Datenfeldes, der nicht vorhanden oder undefiniert ist.
[48] Aufgenommen am 18. August 2002 von einem NASA-Satelliten, mittels eines Moderate Resolution Imaging Spectroradiometer (MODIS; http://modis.gsfc.nasa.gov), der über 36 Spektralbänder verfügt.

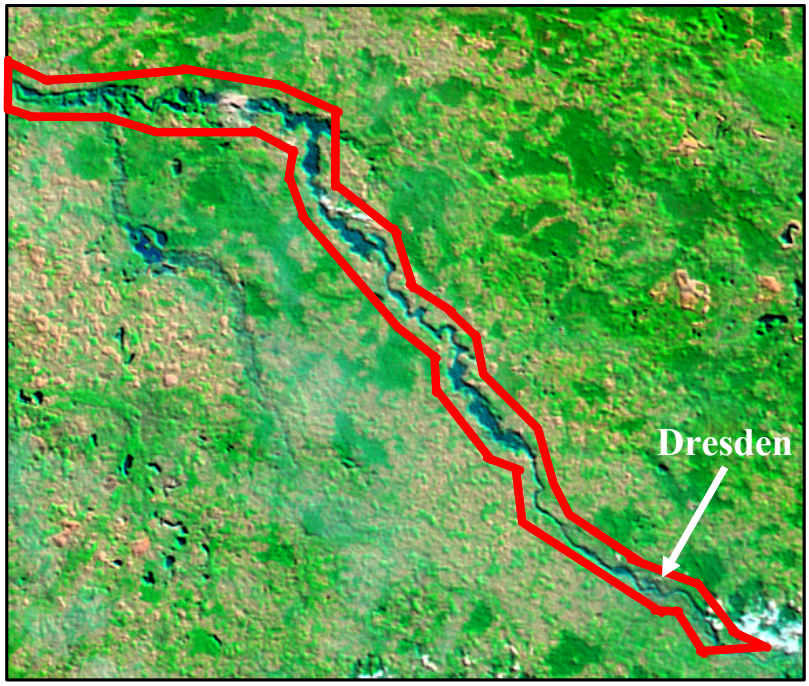


Abbildung 3.45: Beispiel einer Bereichsanfrage mittels Object-Framing.

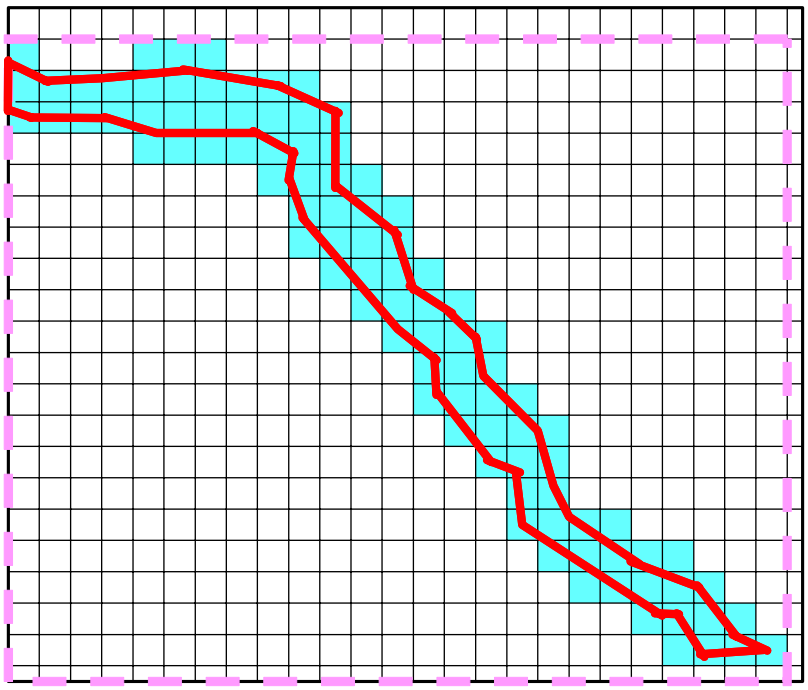


Abbildung 3.46: Vergleich der zu ladenden Kacheln bei Trimming, bzw. Object-Framing.

Die Minimierung des zu ladenden Datenvolumens macht Object-Framing für das Datenretrieval von TS-Medien besonders interessant. Wie bereits in Kapitel 3.4 erwähnt, steckt das stärkste Optimierungspotential in der Vermeidung, bzw. Reduzierung von TS-Zugriffen. Bei

Zugriffen auf TS-Medien erhöht sich ausschließlich die Zugriffsgranularität von Kachel auf Super-Kachel. Diese veränderten Bedingungen werden in Abbildung 3.47 gezeigt.

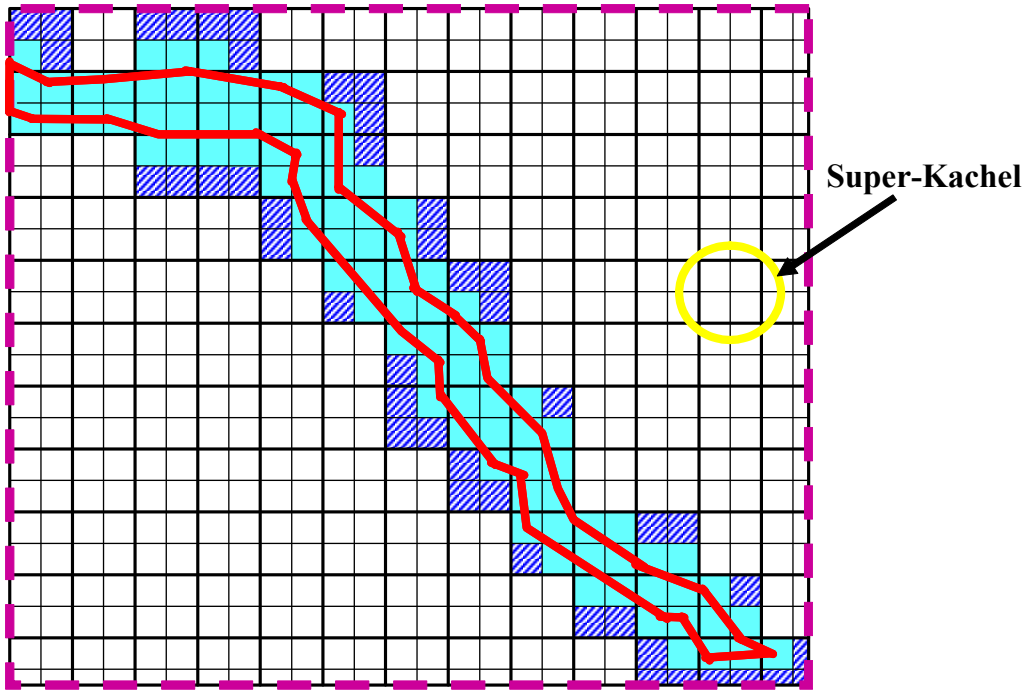


Abbildung 3.47: Vergleich der zu ladenden Super-Kacheln bei Trimming, bzw. Object-Framing.

In diesem Beispiel wird eine Super-Kachel aus vier Kacheln gebildet. Aufgrund der größeren Granularität bei TS-Zugriffen, die durch die Super-Kachel vorgegeben ist, erhöht sich der Verschnitt υ_σ gegenüber dem Verschnitt υ_τ bei der kachelbasierten Anfragebearbeitung. Das ist bei Trimming als auch bei Object-Framing der Fall. Der zusätzliche Verschnitt bei der Object-Framing-Operation ist blau-schattiert dargestellt. Allerdings wird dieser zusätzliche Verschnitt $\upsilon_{\sigma\setminus\tau}$ nach dem Laden der Super-Kachel von tertiären Speichermedien bei der Anfragebearbeitung nicht weiter berücksichtigt. Hier sind ausschließlich die farbig markierten Kacheln aus der Abbildung 3.46 relevant, da nur diese Gegenstand des Anfragebereiches sind. Ein zusätzlicher Aufwand für die Bestimmung der Super-Kachel ist nicht erforderlich, da durch die Bestimmung der relevanten Kacheln über den Index, die entsprechenden Super-Kacheln bereits bekannt sind.

Abbildung 3.48 zeigt eine konkrete Anfrage mittels Object-Framing. Durchgeführt wurde diese RasQL-Anfrage mit RView, einem speziell für RasDaMan entwickelten Bildbetrachter für ein- bis dreidimensionale Daten. Das linke Ergebnisfenster zeigt einen durch Object-Framing spezifizierten Anfragebereich der mexikanischen Golfküste. Die entsprechende Anfrage ist im Query-Editor von RView, in der unteren Bildmitte enthalten. Die rechte Darstellung zeigt nicht den spezifizierte Anfragebereich, sonder die Kachelgrenzen des zu ladenden Datenvolumens. Ermittelt wurden diese Kachelgrenzen, über die Operation eFRAME

(experimental FRAME). Die Operation eFRAME ist ein Nebenprodukt der Implementierung der FRAME-Operation und dient ausschließlich zur Veranschaulichung und Überprüfung der Konzepte. Gegenüber einer Anfrage mittels Trimming erreicht Object-Framing in diesem Bespiel einen Speed-Up von 3,11.

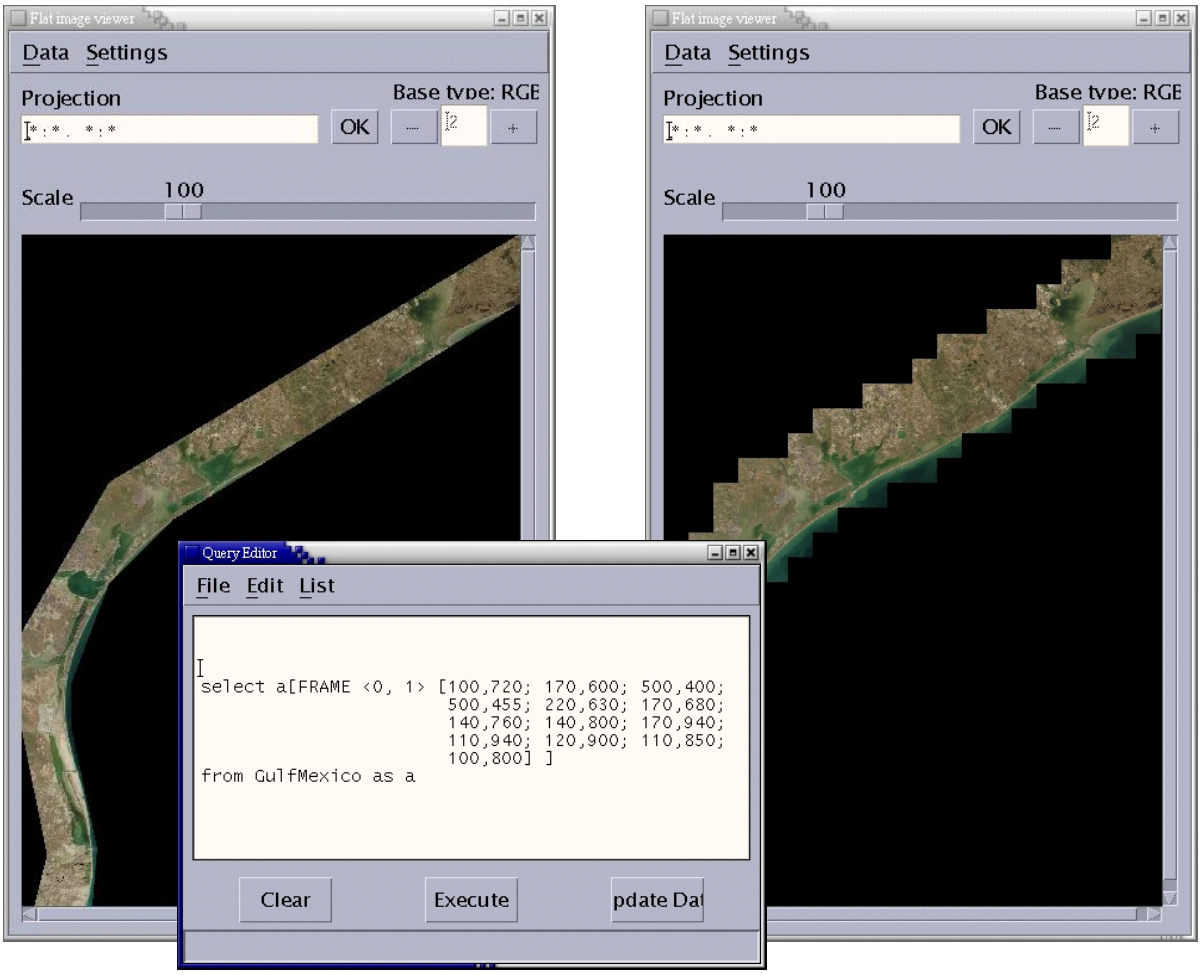


Abbildung 3.48: FRAME- (links) und eFRAME-Operation (rechts), dargestellt mit RView.

Erreichtes Optimierungspotential

Zusammenfassend lässt sich sagen, dass durch Object-Framing neue Wege in der Datenanalyse beschritten werden können. Eine Vielzahl von Analysemöglichkeiten waren bisher bei RasDaMan nicht oder nur sehr umständlich möglich. In der einschlägigen Literatur ist kein vergleichbares Verfahren für die Datenanalyse vertreten. Neben dem Mehrwert in der Datenanalyse ist Object-Framing ein ressourcenschonendes Verfahren, da das zu ladende Datenvolumen für einen spezifizierten Anfragebereich minimiert wird. Es wird eine performante Anfragebearbeitung ermöglicht. Speziell aus der Sicht von HEAVEN kann durch das verminderte Datenvolumen, das für eine Anfrage zu laden ist, die Anzahl der zeitintensiven TS-Zugriffe reduziert werden. Zeitintensive Medienwechsel und Positionierungsoperationen können eingespart werden.

3.11 Systemkatalog für vorberechnete Operationsergebnisse

Aus der Sicht der Anfrageoptimierung wird in diesem Kapitel die Möglichkeit besprochen, wie sich bei der Anfragebearbeitung das Zurückgreifen auf vorberechnete Ergebnisse von multidimensionalen Operationen auswirkt. Durch einen erweiterten Systemkatalog, der vorberechnete Ergebnisse enthält, kann die Performance der Anfragebearbeitung erheblich gesteigert werden. Die Ergebnisse komplexer und berechnungsintensiver Operationen, werden bei erstmaliger Berechnung in der Datenbank abgelegt und bei weiteren Anfragen berücksichtigt. Das Vorhalten der Ergebnisse kann für weitere Anfragen in zweifacher Hinsicht eine Einsparung mit sich bringen:

- Einsparung komplexer und kostenintensiver Berechnungen.
- Reduzierung des zu ladenden Datenvolumens und Minimierung von TS-Zugriffen.

Die Tatsache, dass bei RasDaMan die Bearbeitungsdauer von Anfragen, durch sehr komplexe und berechnungsintensive Operationen (z.B. Aggregatoperationen und induzierte Operationen), nicht selten im Bereich von mehreren Stunden bis Tage liegt, prädestiniert die Verwendung vorberechneter Ergebnisse. Weiterhin kann durch das Zurückgreifen auf vorgehaltene Ergebnisse das für eine Anfrage zu ladende Datenvolumen erheblich reduziert werden. Die Einsparung von Ladeoperationen macht diese Optimierung besonders für HEAVEN interessant, da damit TS-Zugriffe komplett entfallen können. Das optimierte Haushalten mit Rechenkapazität und E/A birgt viel Potential für die Verbesserung der Anfrageperformance. Zunächst ist allerdings zu klären, welche der multidimensionalen Operationen von RasDaMan (Kapitel 2.5.5) ein gutes Potential für diese Art der Optimierung bieten:

Geometrische Operationen

Die geometrischen Operationen Trimming (Zuschneiden), Shift (Verschieben) und Section (Schnitt bzw. Fixieren einer Dimension), zählen zu den berechnungsarmen Operationen. Sie verändern weder den Typ, noch den Wert der Zellen eines MDD. Sie bewirken ausschließlich eine Änderung bezüglich der Domäne. Eine Unterstützung durch vorberechnete Ergebnisse ist nicht sinnvoll.

Aggregatoperationen

Aus der Sicht der Anfrageoptimierung sind Aggregatoperationen besonders gut geeignet, da sie ein MDD auf einen Skalarwert abbilden. Dabei werden alle Zellen zu einem Ergebniswert verdichtet und zusätzlich in einem Systemkatalog gespeichert. Da alle Zellen bei der Berechnung durchlaufen werden, sind diese Operationen sehr rechenintensiv. Zu beachten ist, dass Aggregatoperationen nur auf einem skalaren Basisdatentyp T angewendet werden können. Komplexe Basisdatentypen können durch Kanalselektion auf einen skalaren Datentyp reduziert werden. Teilweise sind somit Ergebnisse von Aggregatoperationen für mehrere Kanäle im Systemkatalog zu speichern. Folgende Aggregatoperationen sind definiert:

- Minimum: *min_cells*()
- Maximum: *max_cells*()

- Summe: *add_cells*()
- Durchschnitt: *avg_cells*()
- Anzahl der Zellen: *count_cells*()
- Existenzquantor: *some_cells*()
- Allquantor: *all_cells*()

Wurde einmal das Minimum, das Maximum, die Summe oder der Durchschnittswert eines MDD α berechnet und im Systemkatalog abgespeichert, so können diese Ergebniswerte solange für zukünftige Anfragen wieder verwendet werden, bis das betroffene Objekt verändert wird. In diesem Fall werden die betroffenen, vorberechneten Ergebnisse aus dem Systemkatalog entfernt. Bei einem erneuten Zugriff erfolgt eine neue Berechnung und die Ergebnisse werden gespeichert. Um die Ergebnisse von nicht mehr angefragten Objekten nicht dauerhaft im Systemkatalog vorhalten zu müssen, kann eine zeitbasierte Bereinigung implementiert werden. So kann in Abhängigkeit des letzten Zugriffes auf diese Ergebnisse entschieden werden, ob sie weiterhin gespeichert bleiben, oder gelöscht werden. Die Komplexität zur erstmaligen Bestimmung von *max_cells*(α) beträgt $O(|cell(\alpha)|) = O(\prod extent(sdom(\alpha)[i])$. Es gilt: $i \in \{1, \ldots, d\}$. Jede Zelle des MDD α muss mindestens einmal verglichen werden. Wird dieses Ergebnis durch einen Systemkatalog vorgehalten, so sinkt der Aufwand beim nächsten Zugriff auf eine konstante Komplexität O(1). Die zusätzliche Speicherung der Ergebniswerte für *avg_cells*() im Systemkatalog ist nicht unbedingt erforderlich. Der Durchschnittswert kann ohne viel Aufwand aus den vorberechneten Werten aus *add_cells*(α) und der Zellenanzahl berechnet werden. Die Ermittlung der Anzahl der Zellen ist über die Domäne des Objektes α möglich ($\prod extent(sdom(\alpha)[i]; i \in \{1, \ldots, d\}$).

Da eine typische Klasse von Anfragen bei RasDaMan nicht nur komplette MDD betreffen, sondern geometrische Operationen wie Trimming oder Section beinhalten, ist es sinnvoll, vorberechnete Ergebnisse zusätzlich in Kachel-Granularität vorzuhalten. Liegt die Domäne der Bereichsanfrage direkt auf den Kachelgrenzen, besteht kein Problem, das Minimum, das Maximum oder die Summe des angefragten Bereiches aus den gespeicherten Kachelergebnissen zu bestimmen. Die Komplexität zur Bestimmung von *add_cells*(*sdom*(q)) beträgt $O(N_{I\tau})$. Für die Bestimmung der Gesamtsumme sind alle vorberechneten Summenwerte der $N_{I\tau}$ Kacheln zu addieren, die durch eine Anfrage q betroffen sind. Die Abbildung 3.49 zeigt den Zeitbedarf (ms) der Operationsberechnung und den E/A-Aufwand der Aggregatoperation *add_cells*() in Abhängigkeit von der Kachelanzahl. Dabei wird die herkömmliche Verfahrensweise von RasDaMan (links) verglichen mit der Verwendung von vorberechneten Ergebnissen (rechts).

Bei den in Abbildung 3.49 durchgeführten Messungen, wird die Domäne der Bereichsanfrage in jedem Schritt um die Domäne einer Kachel [0:499, 0:529] bis zu einer Domäne von [0:2499, 0:3179] erweitert. Die Grenzen der Bereichsanfrage fallen jeweils auf Kachelgrenzen. Bei der herkömmlichen Vorgehensweise (linke Messung) ist zu erkennen, dass bei steigender Kachelanzahl der Aufwand für die Berechnung *sum_cells*() gegenüber dem E/A-Aufwand immer stärker die Gesamtzeitdauer der Anfrage bestimmt. Beim direkten Vergleich

der herkömmlichen (links) und optimierten (rechts) Vorgehensweise fällt sofort auf, dass bei der Verwendung von vorberechneten Ergebnissen keine Balken für die Operationsberechnung zu erkennen sind. Der Aufwand für die Berechnung der Operation *sum_cells*() ist sehr gering (µs) und daher nicht darstellbar. Bei einem Anfragebereich der 30 Kacheln umfasst, sind 29 Additionen der vorberechneten Werte durchzuführen. Dem gegenüber stehen bei der herkömmlichen Vorgehensweise über 7,9 10^6 Additionen von Zellwerten.

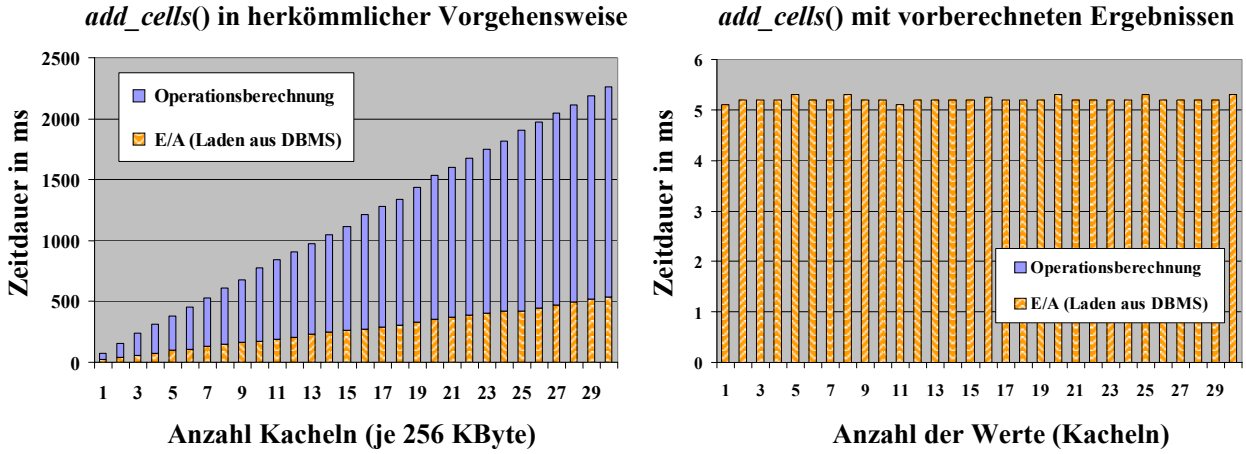


Abbildung 3.49: Aggregatoperation *add_cells*() in herkömmlicher (links) und optimierter (rechts) Vorgehensweise.

Auch der E/A-Aufwand sinkt bei der optimierten Vorgehensweise erheblich. Statt Kacheln mit einem Datenvolumen von je 256 KByte zu laden, sind je nach verwendetem Datentyp Attribute von 4 bis 8 Byte zu lesen. Beim E/A-Verhalten der rechten Messung ist zu erkennen, dass sich die Zeitdauer zum Laden von einem Wert (einer Kachel) gegenüber dem Laden von 30 Werten (30 Kacheln) nicht erhöht. Das liegt daran, dass in diesem Beispiel alle benötigen Einträge auf einer Datenbankseite (8 KByte) gespeichert vorliegen. Nach einmaliger Positionierung (5,2 ms) des Lesekopfes der Festplatte (RAID-System, IBM DeskStar IC35L080AVV, 80 GByte, Ultra ATA-100) wird die benötigte Datenbankseite geladen. Unabhängig, ob ein Eintrag oder 30 Einträge benötigt werden. Werden für eine Anfragebearbeitung mehrer Datenbankseiten benötigt, so steigt selbstverständlich die E/A-Zeitdauer.

Umfasst der Anfragebereich exakt eine Kachel, ist durch die Verwendung von voraggregierten Werten, bei der Anfragebearbeitung eine Speed-Up von 6,2 gegenüber der herkömmlichen Verfahrensweise zu erreichen. Mit wachsender Domäne der Anfrage steigt der Speed-Up weiter an. Enthält der Anfragebereich, wie in unserem Beispiel 30 Kacheln, ergibt sich bereits ein Speed-Up von 162,9. Das zeigt deutlich das Potential einer optimierten Verfahrensweise bei der Datenanalyse. Die hier erreichten Performancesteigerungen wurden bei einer Datenhaltung auf Festplatten erreicht. Bei einer Datenhaltung auf TS-Medien, entfallen zusätzlich zeitaufwendige TS-Zugriffe. Somit ist speziell für HEAVEN eine weitere Performancesteigerung zu erreichen.

Fällt die Grenze des Anfragebereiches innerhalb einzelner Kacheln, so ist die Bestimmung des Gesamtergebnisses nicht mehr so einfach. Bei der Summenbildung *add_cells*() können für die vollständig überdeckten Kacheln die vorberechneten Ergebnisse aus dem Systemkatalog verwendet werden. Bei Grenzkacheln ist es erforderlich, die Kacheln zu laden, um die Summe der teilweise betroffenen Bereiche zu berechen. Durch Addition der einzelnen Teilsummen wird die Gesamtsumme berechnet. Da der Durchschnittswert *avg_cells*() über den Umweg der Summenbildung bestimmt wird, ist auch hier das Laden von Grenzkacheln erforderlich. Bei der Bestimmung der Extremwerte *max_cells*() und *min_cells*(), ist das vollständige Laden der Grenzkacheln nicht unbedingt erforderlich. Soll zum Beispiel das Minimum eines Bereiches bestimmt werden, so wird zunächst das vorläufige Minimum der vom Anfragebereich vollständig überdeckten Kacheln aus dem Systemkatalog ermittelt. Danach werden die vorbestimmten Ergebnisse der teilweise betroffenen Kacheln mit dem vorläufigen Minimalwert verglichen. Liegt der Minimalwert aller Grenzkacheln über dem vorläufigen Minimalwert, so konnte der Minimalwert ohne weiteres Laden von Kacheln bestimmt werden. Gibt es allerdings Grenzkacheln, deren Minimalwert unter dem vorläufigen Minimalwert liegen, so sind diese Kacheln zu laden, da diese das Minimum enthalten können. Bei der Bestimmung des Maximums ist eine analoge Vorgehensweise möglich.

Die Ergebnisse von *count_cells*() und den Quantoren *some_cells*() und *all_cells*() beziehen sich immer auf Prädikate. Die Auswertung dieser Operationen können durch die Nutzung der vorgehaltenen Extremwerte *min_cells*() und *max_cells*() beschleunigt werden. Als Beispiel soll ein MDD α mittels des Operators *some_cells*() auf das Vorhandensein von Zellwerten überprüft werden, die einen Temperaturwert von 23° Celsius übersteigen. Weist der im Systemkatalog gespeicherte Maximalwert des MDD α einen höheren Wert als 23° Celsius auf, kann der Existenzquantor sofort zu „wahr“ evaluieren. Es ist also möglich, einen vorzeitigen Abbruch (Early Termination) zu erreichen, ohne das MDD α zu laden und ohne eine berechnungsintensive zellweise Überprüfung. Um ebenfalls eine Unterstützung von Bereichsanfragen zu erhalten, ist die Granularität der im Systemkatalog gespeicherten Ergebnisse von einem kompletten MDD auf Kacheln zu reduzieren. Liegt die Domäne der Bereichsanfrage direkt auf den Kachelgrenzen, ist es möglich, durch alleiniges Heranziehen der vorberechneten Extremwerte, eine Evaluierung der Quantoren zu „wahr“ oder „falsch“ zu erreichen. Fallen die Domänengrenzen der Anfrage innerhalb einzelner Kacheln, so ist eine Sonderbehandlung für diese Grenzkacheln notwendig. Zur Veranschaulichung dient wieder das Temperaturbeispiel: *some_cells*($\tau > 23$). Liegt der vorberechnete Maximalwert einer Grenzkachel unter 23° Celsius, bzw. ist genau 23° Celsius, so trägt diese Kachel nicht zu einem vorzeitigen Abbruch bei. Übersteigt der vorberechneten Maximalwert einer Kachel den Wert 23° Celsius, muss die entsprechende Kachel geladen werden, um zu überprüfen, ob dieser Wert tatsächlich in dem angefragten Bereich auftritt. Eine ähnliche Vorgehensweise ist bei dem Allquantor *all_cells*(), durch Einbeziehung des vorberechneten Minimalwertes möglich. Wie bereits in Kapitel 3.7.1 beschrieben, kann bei den Existenz-, bzw. Allquantoren, nicht sofort nach erstmaligem Auftreten eines Wahr-Wertes, bzw. Falsch-Wertes der vorzeitige Abbruch erfolgen. Das liegt an dem bei RasDaMan realisierten MDD-basierten Iteratorkonzept zwischen den Operatoren. Aus diesem Grund muss zunächst für den Anfragebereich, der Vergleichsopera-

tor (Induzierte Operation) ein mit Booleanwerten besetztes Ergebnis-MDD α' generieren, bevor dieses durch einen Quantor evaluiert werden kann. Durch die beschriebene Möglichkeit, können allerdings Ladeoperationen und zellweise Vergleiche eingespart werden. Vor allem die Reduzierung von Ladeoperationen, wirkt sich sehr positiv bei einer Datenhaltung auf tertiären Speichermedien aus.

Induzierte Operationen
Induzierte Operationen sind arithmetische und logische Funktionen, die auf jeden Zellwert eines Arrays (MDD) angewendet werden. Da alle Zellen durchlaufen werden, sind induzierte Operationen sehr rechenintensiv. Es lassen sich unäre und binäre induzierte Operationen unterscheiden. Eine unäre induzierte Operation bildet ein MDD α auf ein Ergebnis-MDD α' ab. Als Beispiel kann die Wurzelbildung sqrt(α) jeder Zelle aufgeführt werden. Binäre induzierte Operationen bilden einerseits zwei MDD α und β auf ein Ergebnis-MDD ab (Beispiel: α + β). Oder es wird ein MDD und einen Skalarwert auf ein Ergebnis-MDD abgebildet. Hierunter fällt der zellweise Vergleich (=, !=, <, ≤, >, ≥) mit einer Konstanten. Das Ergebnis-MDD ist aus $\mathbb{B}$.

Im Gegensatz zu Aggregatoperationen, ist das Ergebnis einer induzierten Operation nicht ein einzelner Skalarwert sondern wiederum ein MDD. Aufgrund des hohen Speicheraufwandes, ist es meist nicht von Vorteil, ein berechnetes Ergebnis-MDD für zukünftige Anfragen zu speichern. Wird ein solches Ergebnis-MDD allerdings sehr häufig für weitere Anfragen benötigt, ist dieser Aufwand gerechtfertigt. Aus diesem Grund wurde die Anfragesprache RasQL durch die Möglichkeit einer geschachtelten Einfügeanweisung erweitert:

```
INSERT INTO  <Kollektion>
VALUES       SELECT    <Operation>
             FROM      <Kollektion>
             [WHERE    <Selektionsbedingung>]
```

Die realisierte geschachtelte Einfügeanweisung ermöglicht das Speichern von beliebigen Anfrageergebnissen. Sie können bei zukünftigen Anfragen wieder verwendet werden. Allerdings ist das Abspeichern von Anfrageergebnissen direkt vom Anwender anzustoßen.

Vergleichsbasierte induzierte Operatoren (=, !=, <, ≤, >, ≥), können meist sehr gut durch vorberechnete Extremwerte aus Aggregatoperationen unterstützt werden. Die Verwendung von vorberechneten Extremwerten für den Vergleichsoperator > wurde bereits bei der Beschreibung des Existenzquantors beschrieben. Für die Vergleichsoperatoren <, ≤ und ≥ kann eine analoge Vorgehensweise verwendet werden. Eine Überprüfung auf Gleichheit ist nur bedingt möglich. Es können ebenfalls die die Extremwerte herangezogen werden. Falls das Maximum für einen Bereich gleich dem Minimum ist, bedeutet das, dass alle Zellen den gleichen Wert aufweisen. Zur Überprüfung auf Ungleichheit (!=) können keine vorberechneten Werte verwendet werden.

Abschließend erfolgt eine Abschätzung des Speicherbedarfs bei der Verwendung von vorberechneten Ergebnissen in der Anfragebearbeitung: Bei einem MDD mit 2,4 GByte Datenvolumen und einer typischen Kachelgröße von 256 KByte, müssen Extremwerte (Min- und Maxwert) und die Summe für 9.830 Kacheln abgespeichert werden. Der zusätzliche Speicherbedarf liegt zwischen 115 KByte bei Float-Werten und 230 KByte bei Double-Werten und hält sich somit in Grenzen.

Erreichtes Optimierungspotential

Die Speicherung der vorberechneten Ergebnisse von Operationen ist eine sehr gute Möglichkeit, die Performance der Ausführung von Anfragen zu steigern. Neben dem geringen zusätzlichen Speicheraufwand, sind sehr gute Retrievalzeiten zu realisieren. Das liegt daran, dass sehr häufig bei der Auswahl von Objekten (Selektionsbedingung), auf vorberechnete Ergebnisse zurückgegriffen werden kann. Auch das Endergebnis einer Anfrage beruht in vielen Fällen auf einer Aggregatoperation, wie zum Beispiel der Bestimmung der Maximalwindgeschwindigkeit einer Klimasimulation. Die Verwendung von vorberechneten Ergebnissen bringt Vorteile in zweierlei Hinsicht: Einerseits können Rechenkapazitäten eingespart werden. Nach erstmaliger Berechung der Ergebnisse der Aggregatoperatoren *max_cells*(), *min_cells*() und *add_cells*(), können die Ergebnisse kachelbasiert im Systemkatalog abgelegt werden. Bei der Wiederverwendung dieser Ergebnisse, sind diese zeitintensiven Berechnungen nicht mehr auszuführen. Andererseits wird der E/A-Aufwand reduziert. Anstelle Kacheln aus der Datenbank, bzw. von TS-Medien zu laden, wird auf die vorberechneten Ergebnisse aus dem Systemkatalog zurückgegriffen. Das wiederum bedeutet, dass kostspielige TS-Zugriffe minimiert werden. Zu erwähnen ist noch, dass keines der in Kapitel 2.4 untersuchten Produkte über eine solche Art der Optimierung verfügt.

No way of thinking or doing, however ancient, can be trusted without proof.
Henry David Thoreau (1817-1862)

Kapitel 4 Performance Evaluierung

In diesem Kapitel wird das im Rahmen dieser Dissertation entwickelte System HEAVEN auf der Basis einer Performance Evaluierung bewertet. Dabei wird zunächst in Kapitel 4.1 die verwendete Testumgebung vorgestellt. Weiterhin folgt in Kapitel 4.2 eine kurze Beschreibung der verwendeten Testdaten. Nach dieser Einführung wird in Kapitel 4.3 die Performance des Datenexports auf tertiäre Speichermedien untersucht. Das Kapitel 4.4 evaluiert über alle Ebenen der Speicherhierarchie die Performance beim Datenretrieval.

4.1 Testumgebung

Im Bereich der Performance Evaluierung von Systemen ist die verwendete Testumgebung von entscheidender Bedeutung. Ohne Information über die genutzte Hardware und die Systemarchitektur sind die Qualität und die Richtigkeit der erreichten Messergebnisse nur schwer nachvollziehbar, bzw. belegbar. Aus diesem Grund wird in diesem Kapitel die verwendete Testumgebung vorgestellt. Zur Performance Evaluierung wurden zwei unterschiedliche Systemarchitekturen verwendet.
Bei der Variante A (Abbildung 4.1) befindet sich das DBMS RasDaMan, das konventionelle DBMS Oracle und der Cache-Speicher SHC des HSM-Systems auf dem gleichen Rechner (sunwibas21). Das verwendete TS-System DLT-LXLS ist über eine SCSI-Schnittstelle an den Rechner angebunden. Neben den jeweiligen Schnittstellen, bzw. Übertragungsprotokollen ist die entsprechende Objektgranularität (τ oder σ) der Datenübertragung angegeben.
Bei der Variante B (Abbildung 4.2) wird das HSM-System über ein NFS (Network File System [CPS95]) an das DBMS RasDaMan angebunden. Hierbei handelt sich um ein 100 MBit Ethernet-Netzwerk. Das DBMS RasDaMan ist in dieser Konstellation zusammen mit dem konventionellen DBMS Oracle 8.1.7 auf einem gemeinsamen Rechner (sunwibas21) installiert. Der Cache-Speicher SHC des HSM-Systems befindet sich auf einem separaten Rechner (sunwibas1) und ist als RAID mit dem Level 0 organisiert. An diesen Rechner ist das TS-System DLT-LXLS der Firma Overland Data über einen SCSI-Schnittstelle angebunden. Es sind ebenfalls die jeweiligen Schnittstellen, bzw. Übertragungsprotokolle und die Objektgranularität der Datenübertragung (τ oder σ) angegeben.

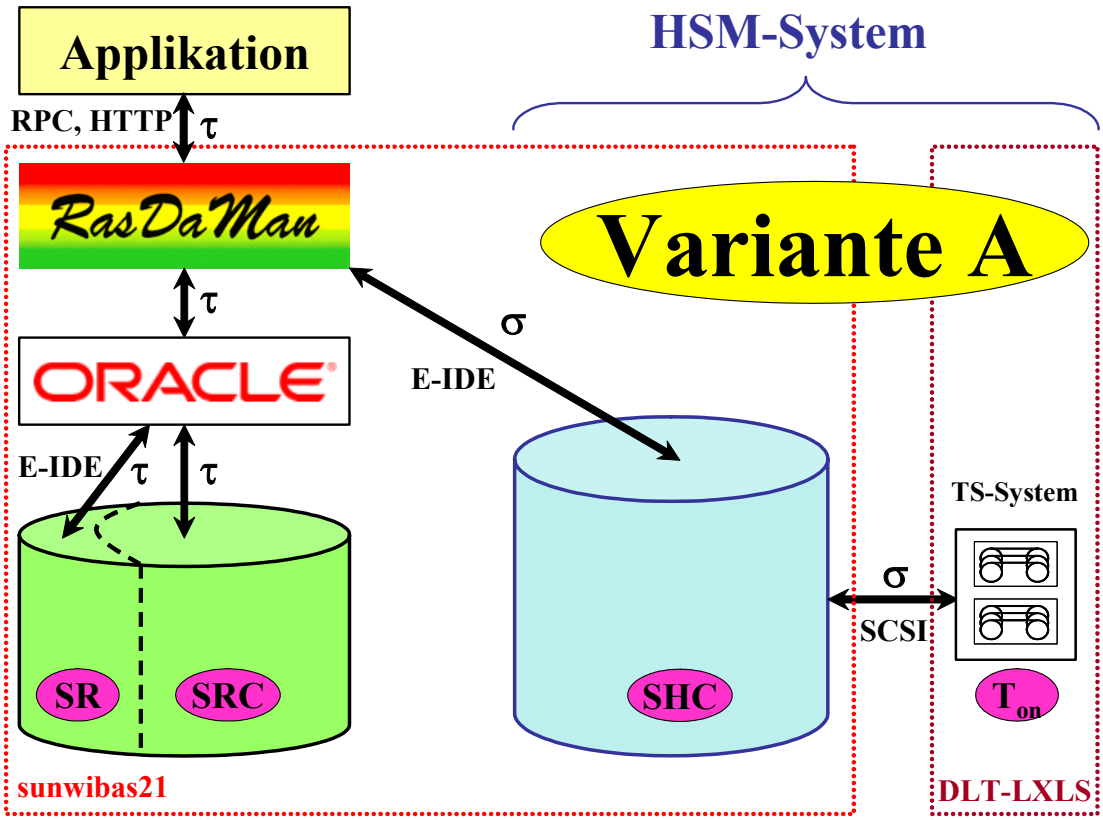


Abbildung 4.1: Testumgebung A: Anbindung eines HSM-Systems mit lokalem SHC.

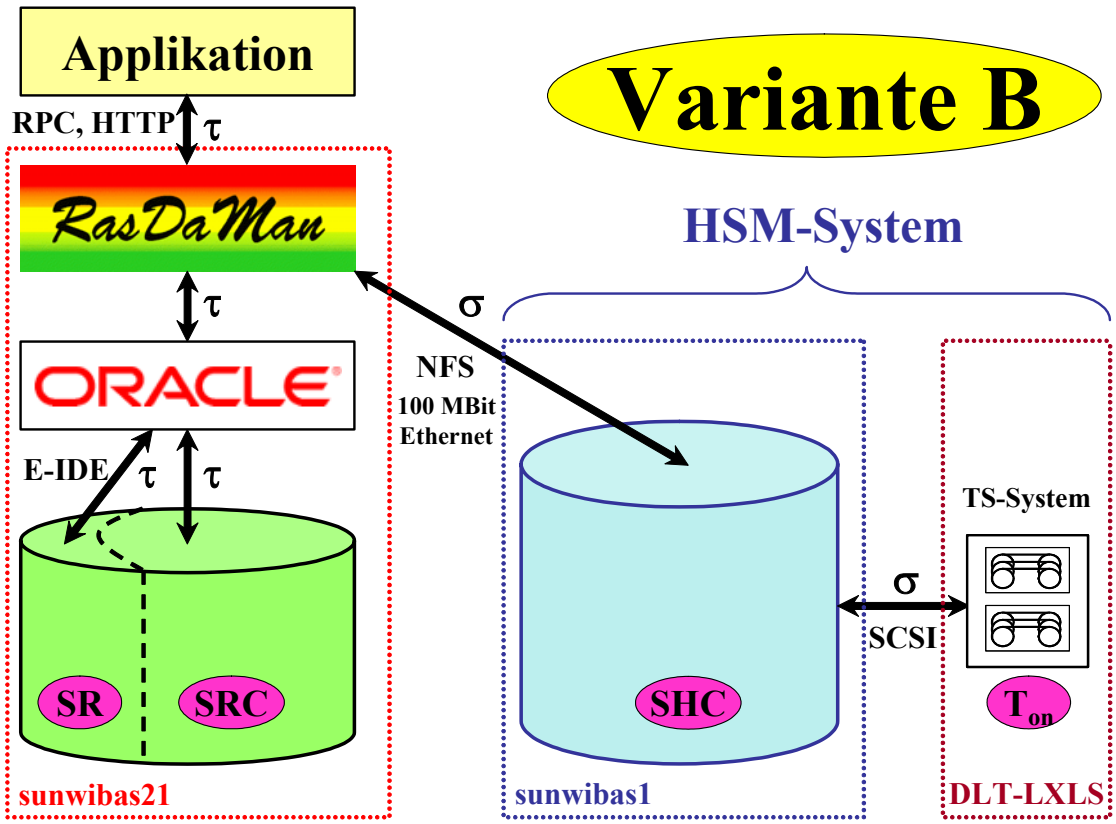


Abbildung 4.2: Testumgebung B: Anbindung eines HSM-Systems über Netzwerk.

Nach erfolgter Beschreibung der beiden verwendeten Systemarchitekturen enthält Tabelle 4.1 eine genauere Spezifikation der verwendeten Rechner sunwibas1 und sunwibas21.

Merkmale	**sunwibas1**	**sunwibas21**
Bezeichnung	Sun Enterprise 250	Sun Blade 150
Hersteller	Sun Microsystems	Sun Microsystems
Betriebssystem	Solaris Version 9	Solaris Version 9
Prozessor	2 * UltraSparc II mit 400 MHz	UltraSparc II mit 650 MHz
Hauptspeicher	1,5 GByte	1,8 GByte
SWAP-Speicher	15 GByte	9,2 GByte
Festplatte	91 GByte (10 * 9,1 GByte Fujitsu Enterprise[49])	80 GByte IBM DeskStar[50] 40 GByte Western Digital Caviar[51]
Blockgröße	8 KByte	8 KByte
Schnittstelle	SCSI	E-IDE
RAID	Level 0	-

Tabelle 4.1: Spezifikation der verwendeten Rechnersysteme sunwibas1 und sunwibas21.

Wie bereits in Kapitel 3.1 beschrieben, können beliebige konventionelle HSM-Systeme bei HEAVEN verwendet werden. Beispiele sind: Storage Archive Manager (SAM) der Firma Sun Microsystems, Tivoli Storage Manager (TSM) der Firma IBM, FileServ der Firma Adic Incorporation, DiskXtender der Firma Legato Systems und StorHouse Storage Manager der Firma FileTek [Sun02, IBM03a, Adic03, CB00, Lega03b]. In Kapitel 3.2 wurde beispielhaft der Tivoli Storage Manager (TSM) der Firma IBM an HEAVEN angebunden. Dieses System wird vom Leibniz-Rechenzentrum (LRZ), dem Rechenzentrum der Münchner Hochschulen, vor allem für die Zwecke Backup und Archivierung eingesetzt. Die HSM-Funktionalität wurde erstmalig im Rahmen dieser Dissertation verwendet. Dem LRZ danke ich für die Möglichkeit der Verwendung des TSM zur Durchführung umfangreicher Tests.

Für die hier vorgestellte Performance Evaluierung des Systems HEAVEN wurde allerdings nicht auf das TSM des LRZ zurückgegriffen. Da sich das TSM im ständigen operativen Betrieb befindet sind exakte Messungen nur schwer durchführbar. Messwerte werden durch konkurrierende Zugriffe verfälscht. Aus diesem Grund wurde im Rahmen der Entwicklung von HEAVEN, auch eine Möglichkeit geschaffen, Bandlaufwerke direkt zu verwenden. Dies wird durch die in Kapitel 3.3 beschriebenen Eigenentwicklung des Tape-Controller-Toolkits (TCT) realisiert. TCT bildet durch die Realisierung eines Cache-Bereiches SHC und eines

[49] Festplatte 9,1 GByte Fujitsu Enterprise MAG3091LC, Ultra-SCSI, 10.000 rpm, 2 MByte Cache, 5 ms Zugriffszeit, 80 MByte/s Transferrate.
[50] Festplatte 80 GByte IBM DeskStar IC35L080AVV, E-IDE (*Enhanced Integrated Drive Electronics*), Ultra-DMA-100, 7.200 rpm, 2 MByte Cache, 8,5 ms Zugriffszeit, 42 MByte/s Transferrate.
[51] Festplatte 40 GByte Western Digital Caviar WD400BB, E-IDE, UltraDMA-100, 7.200 rpm, 8,9 ms Zugriffszeit, 40 MByte/s Transferrate.

TS-Systems die Funktionalität eines HSM-Systems nach. Ein wesentlicher Vorteil von TCT, gegenüber einem im operativen Betrieb befindlichen HSM-System, ist der exklusive Zugriff auf die TS-Medien. Damit werden Messergebnisse nachvollziehbar und reproduzierbar. Sie müssen nicht aufgrund konkurrierender Nutzung gemeinsamer Ressourcen mehrmals wiederholt werden. Konventionelle HSM-Systeme bieten zum Beispiel auch nicht die Möglichkeit, detaillierte Messungen der Ladezeit, der Positionierzeit und der Medienwechselzeit durchzuführen. Es kann nur die gesamte Zeit eines Retrievalvorgangs ermittelt werden. Auch die Dauer der Migration einer Datei auf ein TS-Medium kann nicht bestimmt werden, da ein HSM-System keine Statusmeldung nach dem Schreiben der Datei auf ein TS-Medium liefert. Mit TCT können diese detaillierten Messungen ohne Probleme durchgeführt werden.

Entsprechend den beiden Varianten der verwendeten Systemarchitektur A (Abbildung 4.1) und B (Abbildung 4.2), befindet sich der Cache-Speicher SHC des TCT auf einem über NFS (Network File System) angebundenen Rechner oder auf einer lokalen Festplatte. Das TS-System wird jeweils über eine SCSI-Schnittstelle angebunden. Als TS-System wurde für die Messungen ein Fünffach-Wechsler (1 Laufwerk, 5 Magnetbänder) mit der Bezeichnung DLT-LXLS der Firma Overland Data mit einem integrierten Quantum DLT4000 Laufwerk verwendet. Das Datenblatt des integrierten DLT4000 Laufwerkes, mit den für die Messungen interessanten Spezifikationen, ist in Tabelle 4.2 aufgeführt. Zum Vergleich sind ebenfalls Bandlaufwerke der neuesten Generation enthalten.

Merkmale	**DLT4000**	**Super-DLT 600**	**LTO-Ultrium 4600**
Mittlere Übertragungsrate	1,5 MByte/s	36 MByte/s	30 MByte/s
Spitzen Übertragungsrate	10 MByte/sec	160 MByte/s	-
Kapazität (unkomprimiert)	20 GByte (DLT IV)	300 GByte	200 GByte
Mittlere Zugriffszeit[52]	68 s	79 s	52 s
Mittlere Medienladezeit	37 s	12 s	20 s
Kompressionsverfahren	DLZ[53]	DLZ[53]	ALDC[54]
Interface	SCSI-2	Ultra-160 SCSI 2GBit Fibre Channel	Ultra-3 SCSI

Tabelle 4.2: Datenblatt des DLT4000 Laufwerkes der Firma Quantum, im Vergleich zur aktuellen Band-Technologie.

Das verwendete Bandlaufwerk DLT4000 weist signifikante Unterschiede in der mittleren Übertragungsrate und dem Datenvolumen (Kapazität) auf. Die neueste Generation der Band-

[52] Zeit, die benötigt wird, um ein Magnetband von Beginn bis zur Mitte zu positionieren.
[53] DLZ (*Digital Lempel Ziv*): Algorithmus aus technischer Sicht realisiert als kombinierte Verarbeitung durch Hardware (Zeichenketten suche) und Firmware (Sortierung und Huffman-Kodierung).
[54] ALDC (*Adaptive Lossless Data Compression Algorithm*): Algorithmus basiert auf der DLZ-1 Strategie und ist unter ECMA-222 standardisiert [Ecma95].

laufwerke erreichen Übertragungsraten bis zu 36 MByte/s (unkomprimierte Daten) und native Speichervolumen von 300 GByte. Die mittleren Zugriffszeiten unterscheiden sich hingegen nicht wesentlich. Für die hier durchgeführte Performance Evaluierung hat die verwendete, veraltete Bandlauftechnologie DLT4000 keine Nachteile. Es können alle realisierten Konzepte überprüft werden. Wird eine modernere Bandlaufgeneration verwendet, werden allerdings durch die höhere Übertragungsrate (36 MByte/s anstelle 1,5 MByte/s) bessere Ergebnisse erzielt. Bei den relevanten Messungen wird auf diesen Sachverhalt hingewiesen. Nach erfolgter Beschreibung der Testumgebung werden die Testdaten kurz beschrieben.

4.2 Testdaten

Während der Entwicklungsphase von HEAVEN wurden zur Evaluierung der Performance vor allem zweidimensionale Satellitendaten, drei- und vierdimensionale Klimasimulationen herangezogen. Die verwendeten Datensätze stammen aus den Datenarchiven der Partner aus dem ESTEDI-Projekt. Es handelt sich dabei also nicht um selbst generierte Testdaten, optimiert für gute Messergebnisse, sondern um operative Datensätze aus dem direkten Arbeitsumfeld der Wissenschaftler aus dem Bereich der Hochleistungsrechenzentren.

Für die hier durchgeführte Performance Evaluierung wurde auf vierdimensionale Testdaten einer Klimasimulation des Deutschen Klimarechenzentrums Hamburg zurückgegriffen. Die Simulation zeigt die Entwicklung der Windverhältnisse über einen Zeitraum von 240 Jahren, von Januar 1860 bis Dezember 2099. Die Werte bis in das Jahr 1990 sind gemessene Windgeschwindigkeiten. Ab dem Jahr 1991 entstammen die Daten einer Simulation des Treibhauseffektes entsprechend dem IS92a Szenarium des IPCC'92 (Intergovernmental Panel on Climate Change) mit einem jährlichen Anstieg der Konzentration des Treibhausgases CO_2 um 1 %. Um den Einfluss der so genannten Industriegase besser einfließen zu lassen, wurde das IS92a Szenarium an das Kopenhagen Protokoll IPCC'96 angepasst. Die einzelnen Zellwerte entsprechen der atmosphärischen Windgeschwindigkeit in Kilometer pro Stunde, gespeichert als Gleitkommazahl einfacher Genauigkeit (Datentyp: float). Die vier Dimensionen der Testdaten sind wie folgt besetzt:

1. Dimension: Höhenstufen, bezüglich des atmosphärischen Druckes diskretisiert. Beginnend bei 300 hPa (Hektopascal) bis 1.000 hPa, in 50 hPa-Schritten. (Ausdehnung: 15 Werte).
2. Dimension: Zeitachse in Monaten von Januar 1860 bis Dezember 2099 (Ausdehnung: 2880 Werte).
3. Dimension: diskretisierte geografische Breite (Ausdehnung: 64 Werte).
4. Dimension: diskretisierte geografische Länge (Ausdehnung: 128 Werte).

Aus technischer Sicht entsprechen die einzelnen Zellwerte einem Vektor, der aus einer Windrichtung und einer Windgeschwindigkeit besteht. Zur Speicherung dieses Vektors werden die entsprechenden Werte der Vektorlängen in orthogonaler Richtung gespeichert. Die beiden Komponenten der Windvektoren parallel zum Äquator (geografische Länge) in West-Ost Richtung, bzw. parallel zum Meridian (geografische Breite) in Süd-Nord Richtung, wurden

als zwei MDD von je 1,32 GByte Größe gespeichert. Jedes MDD weist eine Domäne von [0:14, 0:2879, 0:63, 0:127] auf und wurde in einer eigenen Kollektion gespeichert, da viele Analysen auf nur einem Vektor basieren. Die vom Deutschen Klimarechenzentrum gewählte ausgerichtete Kachelung der Ausdehnung [15, 8, 32, 32], mit einem Datenvolumen Δ_τ von 480 KByte, wurde übernommen. Abbildung 4.3 (oben) zeigt die Entwicklung der Windgeschwindigkeit parallel zum Äquator für den Datensatz März 1860, März 1971 und März 2099. Die Zellwerte wurden auf ein Farbspektrum von blau für niedrige Windgeschwindigkeiten bis grün für hohe Windgeschwindigkeiten abgebildet. Sowohl die Visualisierung als dreidimensionale Würfel als auch die Analyse der Extrem- und Durchschnittswerte zeigt eine deutliche Zunahme der Windgeschwindigkeiten im Jahr 2099.

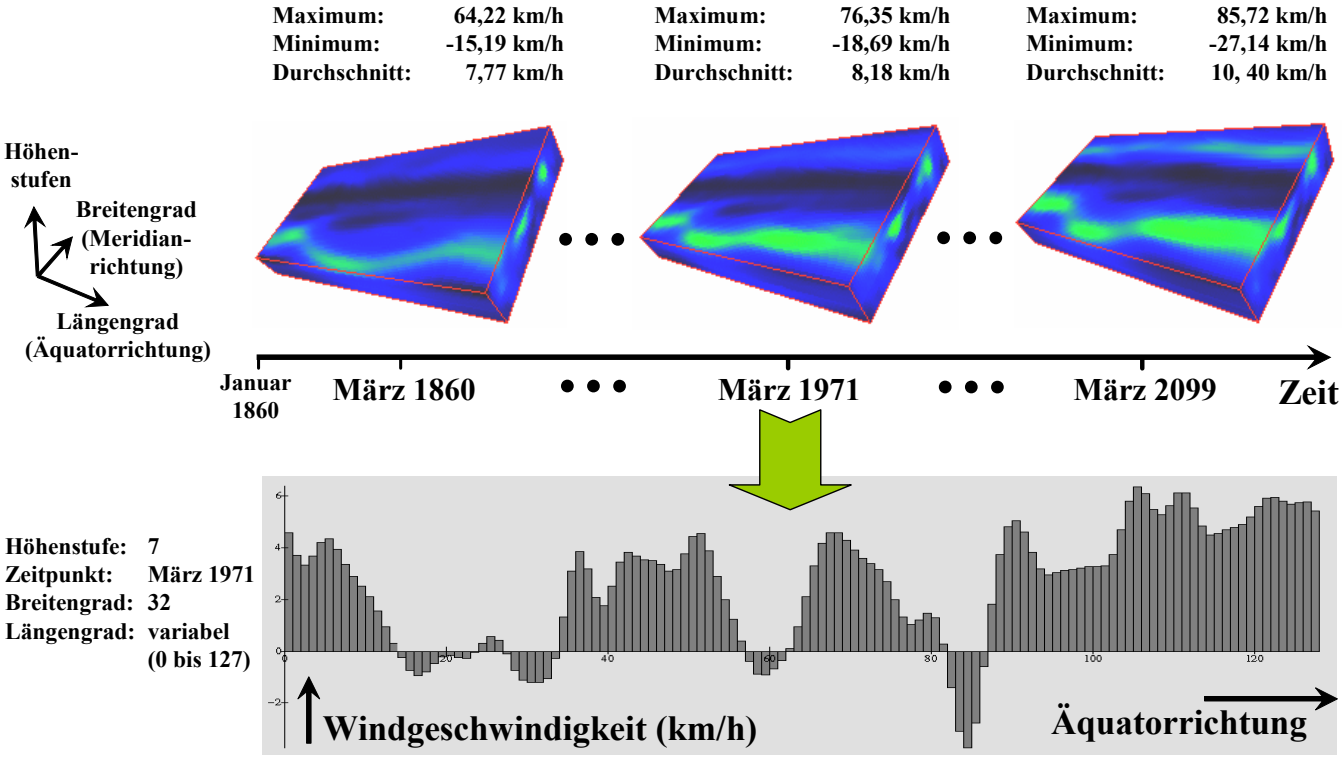


Abbildung 4.3: 4D-Datensatz dkrz4d_vÄ, der Windgeschwindigkeit in Äquatorrichtung.

Das in Abbildung 4.3 (unten) dargestellte Balkendiagramm zeigt den Verlauf der Windgeschwindigkeiten parallel zum Äquator (geografische Länge) für einen ausgewählten Zeitpunkt, Höhenstufe und geografische Breite. Aufgetragene positive Werte der Windgeschwindigkeit entsprechen einem Windverlauf in West-Ost-Richtung. Negative Werte entsprechen einer entgegen gesetzten Windrichtung. Die Ermittlung der absoluten Windgeschwindigkeit $v_{absolut}$ kann durch eine Berechnung nach dem Satz des Pythagoras aus den Vektoren der Windgeschwindigkeit $v_Ä$ parallel zum Äquator (Datensatz: dkrz4d_vÄ), bzw. der Windgeschwindigkeit v_M parallel zum Meridian (Datensatz: dkrz4d_vM) erfolgen:

$$v_{absolut} = \sqrt{v_Ä^2 + v_M^2}$$

4.3 Datenexport

Wird der Exportvorgang von HEAVEN analysiert, so ist, wie bereits in Kapitel 3.6 beschrieben, ein positiver Effekt zu bemerken. Durch die Verwendung eines HSM-Systems (hier TCT), wird das kostenintensive Migrieren von Daten auf tertiäre Speichermedien vollständig von RasDaMan entkoppelt. Erreicht wird das durch den HSM-Cache (SHC), da die zu exportierenden MDD aus der Sicht von RasDaMan, nur in diesen Sekundärspeicher geschrieben werden müssen. Das anschließende, zeitaufwändige Migrieren der MDD (in Granularität von Super-Kacheln) auf TS-Medien, erfolgt automatisch durch das HSM-System (hier TCT), ohne Interaktion von RasDaMan. In dieser Zeit kann RasDaMan neue Aufträge bearbeiten, was zu einer sehr guten Performance führt.

4.3.1 RasDaMan Exportvorgang

Zunächst wird der Exportvorgang untersucht, der das DBMS RasDaMan betrifft. Die anfallenden Kosten $C_{Rexport}$ für das DBMS RasDaMan setzen sich wie folgt zusammen:

$$C_{Rexport} = C_{MC} + C_{eSTAR} + C_{buildST} + C_{moveST}$$

C_{MC} → Aufwand zum Anpassen der Metadaten und der Cacheverwaltung.
C_{eSTAR} → Ermittlung der Super-Kacheln durch den eSTAR-Algorithmus.
$C_{buildST}$ → Laden der relevanten Kacheln (BLOBs) τ aus der Datenbank und mit Funktion *buildST*() die Super-Kacheln σ generieren.
C_{moveST} → Transfer der generierten Super-Kacheln σ mit *moveST*() in den Cache-Speicher SHC des HSM-Systems entsprechend der Testumgebung A oder B (Kapitel 4.1).

Exportvorgang entsprechend der Testumgebung A

Für die Performance Evaluierung des Exportvorganges wird der im vorherigen Kapitel beschriebene Datensatz mit einem Datenvolumen von 1,32 GByte verwendet. Abbildung 4.4 zeigt den Zeitbedarf in Sekunden für die kombinierten Verwaltungskosten C_{MC} und C_{eSTAR}, für die Generierung der Super-Kacheln aus dem DBMS ($C_{buildST}$), für den Transfer der Super-Kacheln in den lokalen Cache-Bereich SHC des TCT (C_{moveST}) und für die Gesamtkosten ($C_{Rexport}$). Als Grundlage dient die Testumgebung A (Abbildung 4.1).

Beim Exportvorgang des Datensatzes werden zunächst durch den eSTAR-Algorithmus 28 Super-Kacheln mit einem Datenvolumen Δ_σ von je 48,7 MByte ermittelt. Eine Super-Kachel enthält somit 104 Kacheln (Δ_τ = 480 KByte). Der Zeitbedarf für die Ermittlung der Super-Kacheln (C_{eSTAR}) und das Anpassen der Metadaten in der Datenbank (C_{MC}) wurden in Abbildung 4.4 unter Verwaltungskosten zusammengefasst. Die Generierung der Super-Kacheln ($C_{buildST}$) umfasst das Laden der relevanten Kacheln (BLOBs) aus der Datenbank und den Zusammenbau der Super-Kacheln. Dabei wird eine Datenrate von etwas mehr als 12 MByte/s erreicht. Bei der Abarbeitung einer Anfrage erreicht die ursprüngliche Version von RasDaMan beim Lesen von Kacheln (BLOBs) aus der Datenbank nur eine Datenrate von ca. 8 MByte/s. Das liegt an der alternierenden Vorgehensweise beim Laden (Kapitel 3.7.1). Die

Performance wird beim Exportvorgang durch die Kombination der Kachel-Ladeaufträge verbessert. Dadurch können mehrere Kacheln (BLOBs) sequentiell von der Festplatte gelesen werden. Betrachtet man den Transfer (C_{moveST}) der einzelnen Super-Kacheln σ in den Cache-Bereich SHC, so wird eine Datenrate von ca. 27,2 MByte/s erreicht. Entsprechend der Testumgebung Variante B befindet sich der SHC auf der lokalen Festplatte. Die Gesamtkosten für den Export von 1,32 GByte nach Variante A betragen 164 Sekunden.

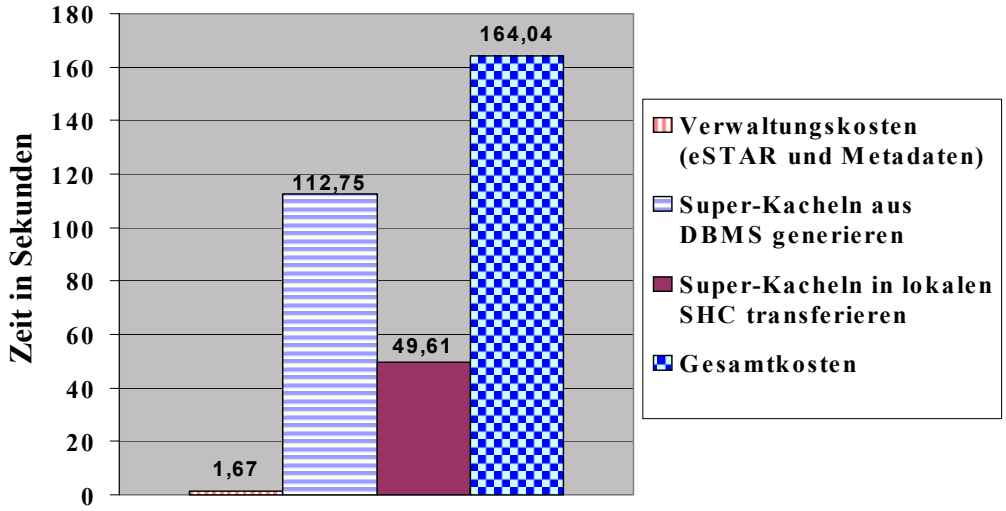


Abbildung 4.4: Export von 1,32 GByte (28 Super-Kacheln) in den lokalen SHC-Bereich.

Abbildung 4.5 zeigt für die einzelnen Super-Kacheln die Kosten $C_{buildST}$ für die Generierung der Super-Kacheln aus der Datenbank und die Kosten C_{moveST} für den Transfer der Super-Kacheln zum lokalen Cache-Bereich SHC des TCT. Wie zu erwarten war weisen die Kosten $C_{buildST}$ und C_{moveST} für die einzelnen Super-Kacheln einen ähnlichen Verlauf auf. Der Zeitbedarf des Exportvorganges steigt linear mit der Anzahl der Super-Kacheln.

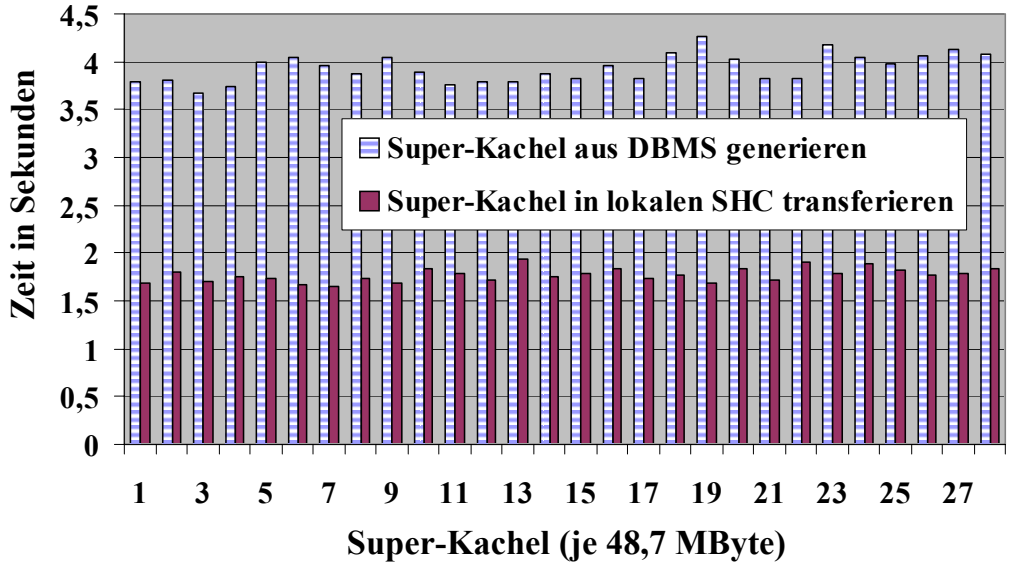


Abbildung 4.5: Datenexport von 28 Super-Kacheln in den lokalen SHC-Bereich.

Exportvorgang entsprechend der Testumgebung B

Da nicht immer vorausgesetzt werden kann, dass sich der Cache-Speicher SHC des HSM-System auf dem gleichen Rechner wie RasDaMan befindet (Variante B) wird eine Anbindung von TCT über ein NFS (Network File System) untersucht. Das entspricht der Testumgebung Variante B (Abbildung 4.2). Bei einer NFS-Anbindung ist aufgrund der notwenigen Übertragung über das Netzwerk eine höhere Laufzeit festzustellen. In der Messung erfolgte die Anbindung über ein 100 MBit Ethernet Netzwerk. Abbildung 4.6 zeigt den Exportvorgang eines 1,32 GByte Datensatzes ohne (links), bzw. mit (rechts) Datenpufferung beim Transfer der Super-Kacheln über das Netzwerk in den SHC-Bereich.

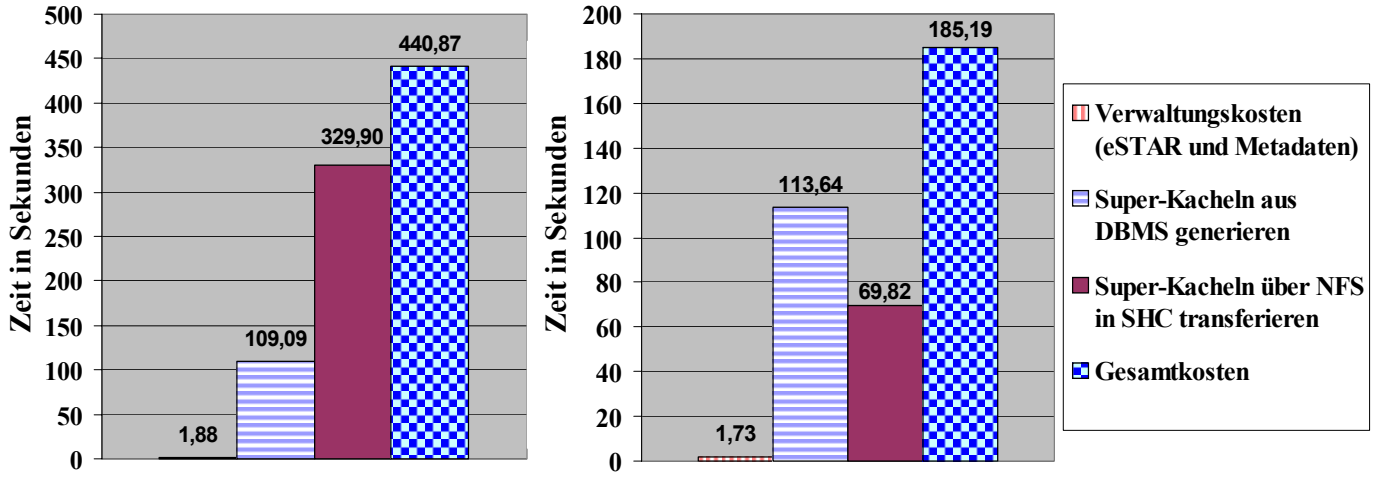


Abbildung 4.6: Datenexport von 1,32 GByte (28 Super-Kacheln) über NFS in SHC-Bereich, ohne (links), bzw. mit (rechts) Datenpufferung.

Sowohl für die Verwaltungskosten (C_{MC} und C_{eSTAR}), als auch für die Generierung der Super-Kacheln aus der Datenbank ($C_{buildST}$) haben sich die Bedingungen nicht verändert. Aus diesem Grund zeigen die beiden Messungen aus Abbildung 4.6 einen ähnlichen Zeitbedarf. Ebenso im Vergleich zu den Messergebnissen der Testumgebung Variante A (Abbildung 4.4) sind bei den ersten beiden Balken kaum Unterschiede zu erkennen. Durch die Übertragung der Super-Kacheln über das Netzwerk ist erwartungsgemäß für C_{moveST} eine höhere Laufzeit gegenüber Variante A festzustellen. Das führt zu höheren Gesamtkosten $C_{Rexport}$ (rechter Balken). Durch den Einsatz einer Datenpufferung während des Schreibvorganges der Datenobjekte (Super-Kacheln) über das Netzwerk in den Speicherbereich SHC wird ein erheblicher Performancegewinn erreicht. Der Zeitaufwand für das Übertragen der Super-Kacheln in den HSM-Cache-Bereich SHC sinkt um einen Faktor 4 bis 5 (Abbildung 4.6). Der Datendurchsatz steigt von 4,09 MByte/s auf 19,56 MByte/s. Durch den verwendeten Datenpuffer ist die Schreiboperation für RasDaMan nicht mehr blockierend und es wird eine gewisse Parallelisierung ermöglicht. Das bedeutet, RasDaMan muss nicht mehr warten, bis eine komplette Super-Kachel über das Netzwerk in den Speicherbereich SHC geschrieben wurde. Vielmehr wird aufgrund der Datenpufferung diese Aufgabe teilweise vom Betriebssystem übernommen. Entsprechend der Größe des verwendeten Datenpuffers ist die Übertragung zum Teil von RasDaMan entkoppelt. Während der restliche, gepufferte Teil einer Super-Kachel noch über das Netzwerk in

den Speicherbereich SHC geschrieben wird, werden parallel dazu bereits von RasDaMan Kacheln (BLOBs) aus der Datenbank geladen, um eine weitere Super-Kachel zu generieren. Das wirkt sich positiv auf die Gesamtlaufzeit ($C_{RasDaMan\,export}$) des Exportvorganges aus. Im ungepufferten Fall werden für 28 Super-Kacheln mit einem Gesamtvolumen von 1,32 GByte ca. 440 Sekunden benötigt. Mit Datenpufferung sinkt der Zeitaufwand auf ca. 185 Sekunden.

Mit speziellen, im HPC-Bereich eingesetzten Technologien, verringert sich die Zugriffslücke zwischen einem Zugriff auf eine lokale Festplatte und einem Netzwerkzugriff. InfiniBand (Infinite Bandwidth) stellt zum Beispiel eine E/A-Architektur zur Verfügung, mit der Übertragungsraten von 500 MByte/s bis zu 6 GByte/s erreicht werden [Mell02, Jni01]. Eine durchgeführte Messung mit einem InfiniBand-Cluster, hat eine Übertragungsrate von ca. 700 MByte/s bestätigt. Bei dem für die Messungen verwendeten 100 MBit Ethernet Netzwerk ist theoretisch eine Transferrate von 12,5 MByte/s möglich. Allerdings werden in der Praxis maximale Übertragungsraten von 10 MByte/s erreicht. Durch verbesserte Technologien wird der bestehende Flaschenhals durch eine Netzwerkübertragung signifikant verringert.

4.3.2 Entkoppelter TCT Exportvorgang

Die Verwendung eines HSM-Systems entkoppelt das kostenintensive Migrieren von Daten auf tertiäre Speichermedien vollständig von RasDaMan. Als HSM-System wird die Eigenentwicklung TCT (Tape Controller Toolkit) eingesetzt. Wurden die Super-Kacheln σ von RasDaMan in den Cache-Bereich SHC des HSM-Systems geschrieben, kann RasDaMan eine neue Aufgabe übernehmen. Das anschließende, zeitaufwändige Migrieren von Super-Kacheln auf TS-Medien, erfolgt automatisch durch das HSM-System, ohne Interaktion von RasDaMan. Die anfallenden Kosten $C_{TCT\,export}$ für das HSM-System setzen sich wie folgt zusammen:

$$C_{TCT\,export} = C_{Verw} + C_{Anlauf} + C_{Pos} + C_{write\,\sigma}$$

C_{Verw} → Verwaltungskosten des HSM-Systems. Z.B. Suche nach geeignetem TS-Medium $v \in V$ für den exportierenden Datensatzes und Anpassung der Metadaten.

C_{Anlauf} → Anlaufkosten beziehen das Warten auf freie Ressourcen und die Kosten für anfallende Operationen zum Medienwechsel[55] $o_{sv} = \{o_{gv}, o_{lv}, o_{rv}, o_{ev}, o_{dv}\}$ mit ein.

C_{Pos} → Positionierung des Schreibkopfes an den Anfang des freien Speicherbereiches.

$C_{write\,\sigma}$ → Schreibvorgang eines Datensatzes σ auf das TS-Medium $v \in V$.

Die Verwaltungskosten C_{Verw} zum Auffinden eines geeigneten TS-Mediums für den anstehenden Datenexport, bzw. das Anpassen der Metadaten, bewegen sich im Bereich von mehreren Millisekunden. Etwaige Anlaufkosten C_{Anlauf} hängen sehr stark von der Frequentierung des HSM-Systems ab. Sie können sich im Bereich von 40 Sekunden bis etwa 400 Sekunden bewegen. Bei der durchgeführten Messung gehen wir davon aus, dass sich das relevante TS-Medium bereits in einem Laufwerk befindet. Abbildung 4.7 zeigt den von RasDaMan ent-

[55] Medienwechseloperation *Swap-Volume* o_{sv} setzt sich zusammen aus den Operationen *Get-Volume* o_{gv}, *Load-Volume* o_{lv}, *Rewind-Volume* o_{rv}, *Eject-Volume* o_{ev}, und *Deposit-Volume* o_{dv} (Definition 3.16 und Definition 3.17).

koppelten Exportvorgang des HSM-Systems TCT. Die dargestellten Messergebnisse gelten gleichermaßen für die beiden Testumgebungen A und B. Mittels des HSM-Systems TCT werden 28 Super-Kacheln mit einem Gesamtvolumen von 1,32 GByte unter Verwendung des Fünffach-Wechslers DLT-LXLS der Firma Overland Data auf ein DLT4000 Magnetband geschrieben. Nach einer initialen Positionierung (C_{Pos}) von etwa 135 Sekunden werden die einzelnen Super-Kacheln sequentiell auf das Magnetband geschrieben ($C_{write\sigma}$). Die Dauer der initialen Positionierung ist abhängig von der Position des freien Speicherbereiches auf dem Magnetband. In diesem Fall befindet sich der freie Speicherbereich im letzten Viertel des Magnetbandes. Ein DLT4000 Magnetband weist eine mittlere Positionierungszeit von 68 Sekunden auf. Vor jeder weiteren Schreiboperation findet eine weitere Positionierungsoperation mit sehr kurzer Dauer (ms) statt, um zum nächsten freien Datenblock auf dem Magnetband zu positionieren. In Abbildung 4.7 sind diese zusätzlichen Operationen zum Positionieren aufgrund der sehr kurzen Zeitdauer nicht zu erkennen. Die Gesamtdauer des Exportvorganges von 28 Super-Kacheln mit einem Gesamtvolumen von 1,32 GByte mit dem HSM-System TCT beträgt ca. 900 Sekunden. Abzüglich der initialen Positionierungsoperation benötigt das Scheiben 766 Sekunden. Das entspricht einer Schreibrate von 1,76 MByte/s. Wird ein TS-System der neuesten Generation eingesetzt, können Schreibraten von über 36 MByte/s erreicht werden. Die mittlere Zugriffszeit wird allerdings nicht verbessert.

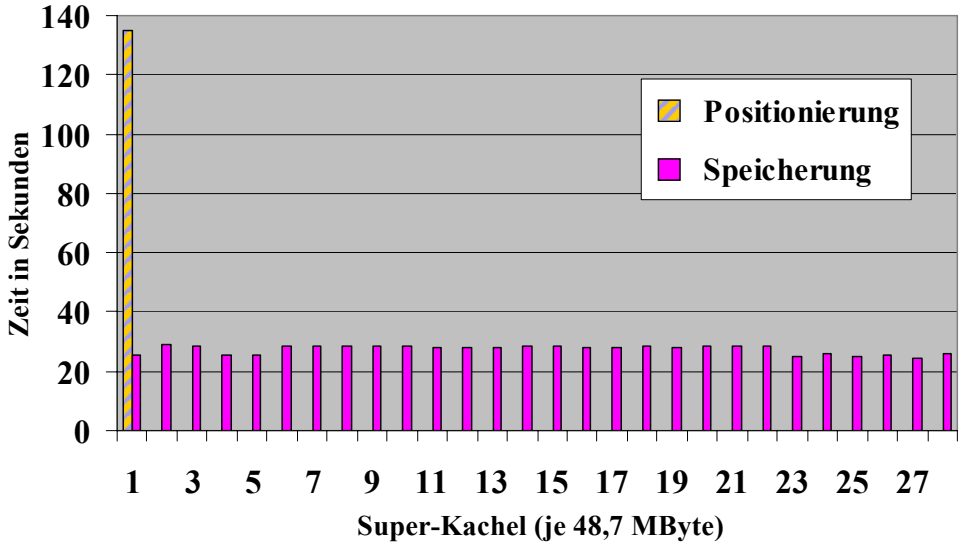


Abbildung 4.7: Datenexport von 28 Super-Kacheln mit TCT auf ein DLT4000 Magnetband.

4.4 Datenretrieval

Aus der Sicht des Anwenders, erfolgt das Retrieval von multidimensionalen Daten innerhalb HEAVEN, ebenso wie in der ursprünglichen Version von RasDaMan, ohne Anbindung von tertiären Speichersystemen (Kapitel 3.7). Es lassen sich alle in RasDaMan zur Verfügung stehenden Operationen ohne Einschränkung ausführen. Für den Anwender ist der Retrievalvorgang transparent. Als einziger Unterschied gilt die teilweise höhere Antwortzeit bei Anfragen, wenn sich die gewünschten Daten auf TS-Medien befinden. Das liegt daran, dass im

Unterschied zum Exportieren von Daten auf TS-Medien, die Aktionen von RasDaMan und dem HSM-System zumindest teilweise gekoppelt sind. Bei einem Zugriff *access*(τ) wird zunächst mit *locate*(τ), die in der Speicherhierarchie am höchsten gelegene Speicherebene (SR, SRC, SHC, T_{on}, T_{near}, T_{off}) ermittelt, welche das Objekt τ enthält. Mit der Funktion *replicate*(τ, source ∈ {SR, SRC, SHC, T_{on}, T_{near}, T_{off}}, P), wird der entsprechende Datensatz von der ermittelten Speicherebene in den Primärspeicher P geladen und die gewünschten Operationen ausgeführt. Befindet sich das für eine Anfrage (Query) q relevante Datenobjekt auf einem TS-Medium setzen sich die Kosten für die Anfragebearbeitung C_q wie folgt zusammen:

$$C_q = C_{TCTaccess} + C_{Raccess}$$

$C_{TCTaccess}$ → Ladevorgang der Datenobjekte σ von den Speicherbereichen T_{on}, T_{near}, bzw. T_{off} in den Cache-Bereich SHC des HSM-Systems TCT (Kapitel 4.4.1).

$C_{Raccess}$ → Bearbeitung der Anfrage q durch RasDaMan, inklusive Ladevorgang der Datenobjekte σ vom Speicherbereich SHC in den Primärspeicher P (Kapitel 4.4.2).

Im Rahmen der Performance Evaluierung werden zunächst die anfallenden Kosten $C_{TCTaccess}$ beim Lesen der Datenobjekte von einem TS-Medium untersucht. Im Anschluss folgt in Kapitel 4.4.2 eine Evaluierung der aufgeschlüsselten Kosten $C_{Raccess}$.

4.4.1 Datenretrieval durch das TS-System

Die im HSM-System TCT anfallenden Kosten $C_{TCTaccess}$ beim Laden der Datenobjekte von einem TS-Medium setzten sich wie folgt zusammen:

$$C_{TCTaccess} = C_{Verw} + C_{Anlauf} + C_{Pos} + C_{readσ}$$

C_{Verw} → Verwaltungskosten des HSM-Systems. Suche des für die Anfrage q relevanten TS-Mediums v(σ) ∈ V.

C_{Anlauf} → Anlaufkosten beziehen das Warten auf freie Ressourcen und die Kosten für anfallenden Operationen zum Medienwechsel[56] $o_{sv} = \{o_{gv}, o_{lv}, o_{rv}, o_{ev}, o_{dv}\}$ mit ein.

C_{Pos} → Positionierung des Schreibkopfes an den Anfang des zu lesenden Datensatzes σ.

$C_{readσ}$ → Lesevorgang eines Datensatzes σ von einem TS-Medium v(σ) ∈ V.

Die Verwaltungskosten C_{Verw} des HSM-Systems beim Suchen des für die Anfrage q relevanten TS-Mediums v(σ) ∈ V bewegen sich im Bereich einiger Millisekunden. Im Vergleich zu den weiteren Kosten können sie vernachlässigt werden. Wie bereits beim Datenexport (Kapitel 4.3.2) beschrieben, hängen die Anlaufkosten C_{Anlauf} (40 - 400 Sekunden) sehr stark von der Frequentierung des HSM-Systems ab. Bei den durchgeführten Messungen gehen wir davon aus, dass sich das relevante TS-Medium bereits in einem Laufwerk befindet. Die Zugriffskosten reduzieren sich bei den Messungen somit auf Operationen zum Positionieren (C_{Pos}) und zum Lesen ($C_{readσ}$) der Datensätze. Anhand des unterschiedlichen Positionierungsverhaltens

[56] Medienwechseloperation *Swap-Volume* o_{sv} setzt sich zusammen aus den Operationen *Get-Volume* o_{gv}, *Load-Volume* o_{lv}, *Rewind-Volume* o_{rv}, *Eject-Volume* o_{ev}, und *Deposit-Volume* o_{dv} (Definition 3.16 und Definition 3.17).

eines TS-Systems wird die Relevanz der in Kapitel 3 vorgestellten Optimierungen schrittweise bewiesen. Es werden die in HEAVEN realisierten Optimierungen, wie das Super-Kachel-Konzept, das Intra-Super-Tile-Clustering (Kapitel 3.5 und 3.6), das Inter-Super-Tile-Clustering (Kapitel 3.6) und das Intra-Query-Scheduling (Kapitel 3.7) am Beispiel von Bereichsanfragen auf einzelne Datenobjekte und Objektmengen durch Messungen evaluiert.

4.4.1.1 Bereichsanfragen auf Einzelobjekte

Zur Evaluierung von Bereichsanfragen auf einzelne Datenobjekte, wird durch eine geometrische Operation die Windgeschwindigkeit $v_{Ä}$ parallel zum Äquator (Datensatz: dkrz4d_vÄ) im Zeitraum von Dezember 1999 bis Dezember 2099 (100 Jahre) angefragt. Die entsprechende RasQL-Anfrage lautet:

```
SELECT vÄ[*:*, 1680:2879, *:*, *:*] FROM dkrz4d_vÄ as vÄ
```

Ausgehend vom vierdimensionalen Testdatensatz dkrz4d_vÄ mit einem Gesamtvolumen von 1,32 GByte wird die Zeitdomäne von 240 Jahren (2.880 Monate) auf 100 Jahre (1.200 Monate) eingeschränkt. Diese Bereichsanfrage liegt direkt auf Kachelgrenzen und umfasst 1.200 Kacheln (Δ_τ = 480 KByte), mit einem Gesamtdatenvolumen von 562,5 MByte. Ein Verschnitt υ_τ tritt somit nicht auf. Die hier vorgestellten Messergebnisse gelten sowohl für Testumgebung Variante A, als auch für Variante B.

Auswirkung der Zugriffsgranularität beim Datenretrieval

Zunächst wird untersucht, wie sich die Granularität der auf TS-Medien gespeicherten Datenobjekte auf das Zugriffsverhalten beim Datenretrieval auswirkt. Im Gegensatz zur HEAVEN, wird bei dem in Kapitel 2.4.3 vorgestellten DBMS Postgres mit TS-Anbindung, keine der jeweiligen Speicherhierarchie angepasste Datengranularität gewählt. Postgres ermöglicht durch das Chunking-Konzept (entspricht regulärer Kachelung) große Objekte in gleichförmige Teilobjekte zu zerlegen und als einzelne LOBs (Large Objekts) in der Datenbank, bzw. auf einem TS-Medium, abzuspeichern. Die Teilobjekte werden entsprechend der Reihenfolge entlang der ersten Dimension auf das Magnetband geschrieben. Es muss klar sein, dass ein vom Anwender gewähltes Chunking nicht für alle Anfragenmuster gleich gut geeignet ist. Eine Verschlechterung der Performance tritt ein, wenn eine Bereichsanfrage Teildatenobjekte anfordert, die nicht der beim Exportieren gewählten Ordnung entsprechen. Das wird vor allem durch die notwendigen Positionierungsoperationen auf dem Magnetband hervorgerufen. Abbildung 4.8 zeigt beispielhaft den zeitlichen Verlauf der Operationen zum Positionieren und zum Lesen der einzelnen Teildatenobjekte. Die Teildatenobjekte wurden in Kachel-Granularität (Δ_τ = 480 KByte) auf das Magnetband geschrieben. Die Anfrageordnung der oben beschriebenen Bereichsanfrage entspricht nicht der Reihenfolge der Kacheln auf dem Magnetband. Zur Beantwortung der Bereichsanfrage sind 1.200 Kacheln vom TS-Medium zu laden. Bei der Betrachtung der Messung in Abbildung 4.8 fällt sofort auf, dass die Positionierungsoperationen stark überwiegen. Aufgrund des geringen Datenvolumens (je 480 KByte) der gespeicherten Datenobjekte sind die Leseoperationen kaum zu erkennen. Der Ladevorgang mit dem Fünffach-Wechsler DLT-LXLS der Firma Overland Data mit einem integrier-

ten Quantum DLT4000 Laufwerk benötigt für 1.200 Kacheln 102,33 Minuten. Für das angefragte Datenvolumen von 562,5 MByte ist dieser Wert inakzeptabel.

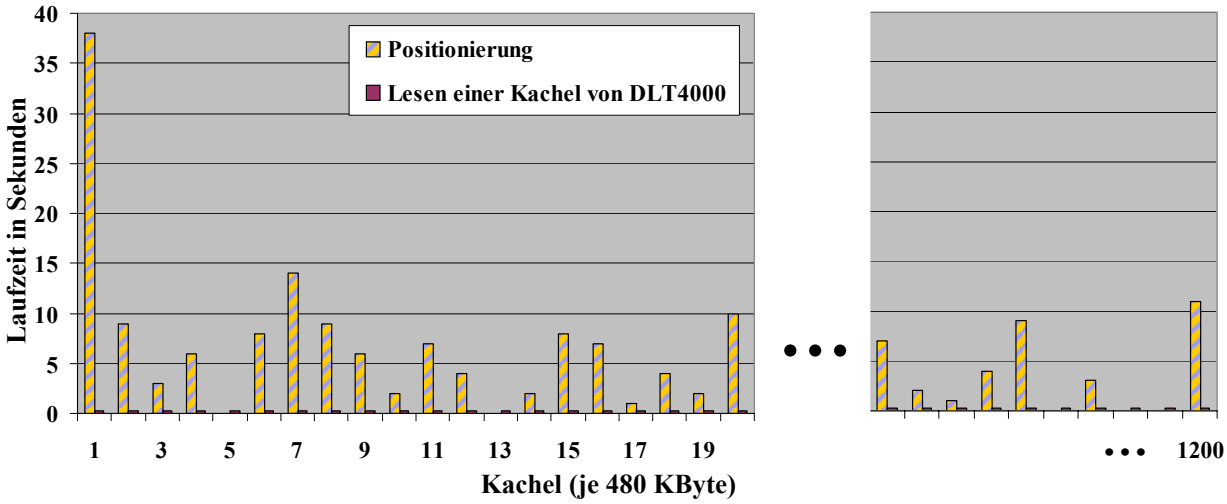


Abbildung 4.8: Bearbeitung einer Bereichsanfrage in Kachel-Granularität ohne Optimierung.

Bei dem im Rahmen dieser Dissertation entwickelten System HEAVEN wird die Dominanz der Positionierungsoperationen gegenüber den Leseoparationen durch eine, dem jeweiligen Speicherort angepasste Datengranularität (Kachel τ und Super-Kachel σ), verringert (Kapitel 3.5). Durch die höhere Datengranularität und der damit verbundenen geringeren Anzahl an Teildatenobjekten sinkt die maximal mögliche Anzahl an Positionierungsoperationen bei einer Anfrage enorm. Als optimale Objektgranularität für das DLT4000 Laufwerk wurde von HEAVEN ein Datenvolumen von 48,7 MByte ermittelt (Kapitel 3.5.4). Eine Super-Kachel σ enthält somit 104 Kacheln τ mit je 480 KByte. Entscheidend für eine gute Performance bei einem späteren Datenretrieval ist eine Kombination geeigneter Kachel zu Super-Kacheln. Entgegen dieser Vorgabe zeigt Abbildung 4.9 die Laufzeit der Operationen zum Positionieren und Laden der Super-Kacheln von einem TS-Medium bei einer wahllosen Kombination von Kacheln zu Super-Kacheln. Aufgrund der zufälligen Verteilung der Kacheln auf die Super-Kacheln sind zur Beantwortung der Bereichsanfrage 28 Super-Kacheln zu laden.

Obwohl gegenüber der kachelbasierten Speicherung ein größeres Datenvolumen, mit einem damit verbundenen Verschnitt υ_σ von 58,3 %, vom Magnetband zu laden ist, sinkt die Gesamtlaufzeit der Ladeoperation von 102,33 Minuten auf 35,98 Minuten. Das wird durch die Reduzierung der Anzahl der maximal möglichen Positionierungsoperationen erreicht. Die Anzahl sinkt von 1.200 bei der Speicherung in Kachel-Granularität auf 28 bei Super-Kachel-Granularität. Auch konventionelle Systeme, wie das in Kapitel 2.4.1 untersuchte System StorHouse/RM der Firma FileTek, bieten die Möglichkeit die Speichergranularität durch die Bildung von Dateifamilien zu erhöhen. Allerdings können diese nicht hinsichtlich zukünftiger multidimensionaler Anfragemuster optimiert werden. Bei der Entscheidung zur Bildung von Dateifamilien werden ausschließlich Kriterien wie zum Beispiel das Entstehungsdatum herangezogen.

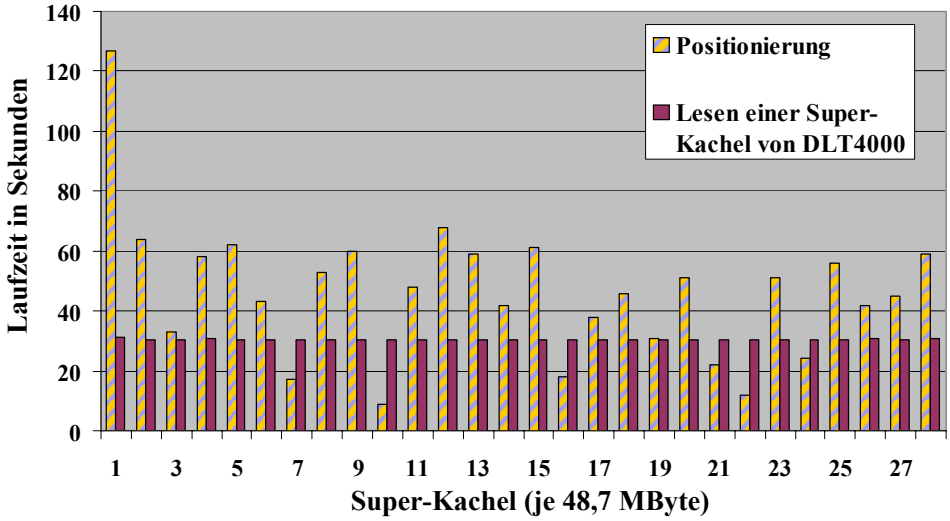


Abbildung 4.9: Bearbeitung einer Bereichsanfrage ohne Optimierung.

Entscheidend für eine Verbesserung der Retrievalperformance ist eine Kombination geeigneter Kachel zu Super-Kacheln hinsichtlich zukünftiger Anfragemuster. Im Idealfall werden für eine Anfrage alle in einer Super-Kachel enthaltenen Kacheln benötigt. Das reduziert gegenüber der wahllosen Kombination von Kacheln zu Super-Kacheln meist die Anzahl der für eine Anfrage zu ladenden Super-Kacheln. Bei einer optimierten Vorgehensweise bei der Kombination von Kacheln zu Super-Kacheln kann die Anzahl der zu ladenden Super-Kacheln von 28 auf 12 Stück (1.200 * 480 KByte ≤ 12 * 48,7 MByte) reduziert werden. Bei HEAVEN wird mittels Intra-Super-Tile-Clustering eine Optimierung diesbezüglich durchgeführt.

Datenretrieval mit Intra-Super-Tile-Clustering

Beim Exportieren der Datensätze auf TS-Medien realisiert der erweiterte Super-Kachel-Algorithmus (eSTAR) das Intra-Super-Tile-Clustering (Kapitel 3.5). Durch dieses Vorgehen werden Kacheln, die im multidimensionalen Raum benachbart sind, zu Super-Kacheln zusammengefasst und auf Magnetband exportiert. Dadurch wird die Wahrscheinlichkeit gesteigert, dass die für eine Bereichsanfrage relevanten Kacheln in nur wenigen Super-Kacheln enthalten sind. Die Anzahl der zu ladenden Datenobjekte wird durch das Intra-Super-Tile-Clustering reduziert. Gleichzeit entfallen kostenintensive Positionierungsoperationen. Zur Beantwortung der oben gestellten Bereichsanfrage, die 1.200 Kacheln (562,5 MByte) umfasst, sind 12 Super-Kacheln (585 MByte) von einem TS-Medium zu laden. Der Verschnitt υ_σ beträgt 3,84 % (es gilt $\upsilon_\sigma = \upsilon_{\sigma\setminus\tau}$, da $\upsilon_\tau = 0$). In Abbildung 4.10 ist der zeitliche Aufwand für das Positionieren und das Laden der erforderlichen 12 Super-Kacheln zu erkennen. Gegenüber der Messung aus Abbildung 4.9 konnte die zu ladende Objektanzahl von 28 Super-Kacheln auf 12 Super-Kacheln reduziert werden. Die Gesamtzeit zum Laden der für die Bereichsanfrage relevanten Super-Kacheln verkürzt sich gegenüber der nicht optimierten Super-Kachel-Variante von 35,98 Minuten auf 15,79 Minuten. Allerdings fällt auf, dass vor dem Lesen jeder Super-Kachel eine zeitintensive Positionierungsoperation erforderlich ist. Dieses Defizit wird durch die Kombination des Inter-Super-Tile-Clustering mit dem Intra-Query-Scheduling beseitigt.

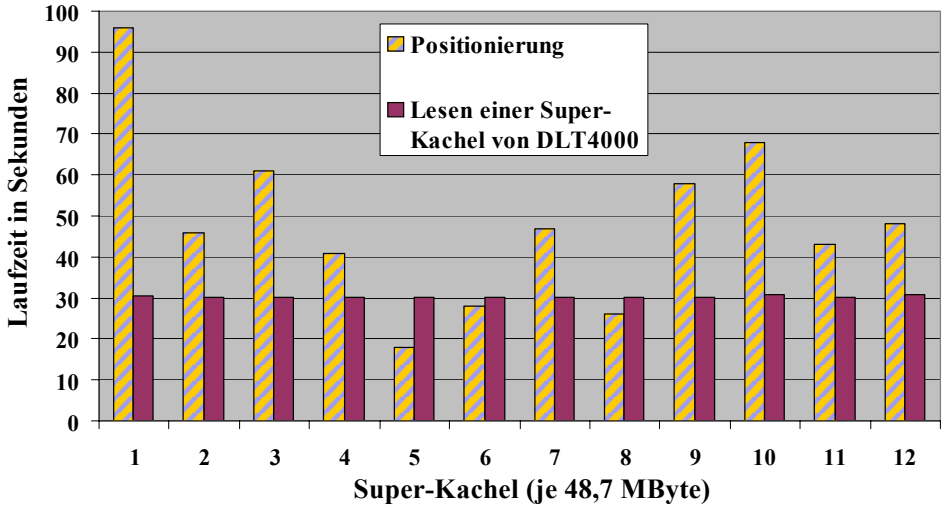


Abbildung 4.10: Bearbeitung einer Bereichsanfrage mit Intra-Super-Tile-Clustering.

Datenretrieval mit Intra-Query-Scheduling

Das realisierte Intra-Query-Scheduling (Kapitel 3.7) passt die Anforderungsreihenfolge für die auf TS-Medien gespeicherten Datenobjekte (Super-Kacheln) entsprechend der beim Exportieren gewählten Linearisierungsordnung (Inter-Super-Tile-Clustering) an. Die in HEAVEN implementierte Umsetzung minimiert die Retrievalkosten erheblich. Abbildung 4.11 zeigt den gegenüber Abbildung 4.10 reduzierten zeitlichen Aufwand für die gleiche Bereichsanfrage. Abgesehen von einer initialen Positionierungsoperation vor dem Lesevorgang der ersten Super-Kachel konnten alle weiteren Operationen durch eine geschickte Sortierung der Anfragewarteschlange eingespart werden. Die Zeitdauer des Ladevorganges der Super-Kacheln vom Magnetband konnte gegenüber der Messung aus Abbildung 4.10 mehr als halbiert werden und beträgt noch 7,21 Minuten.

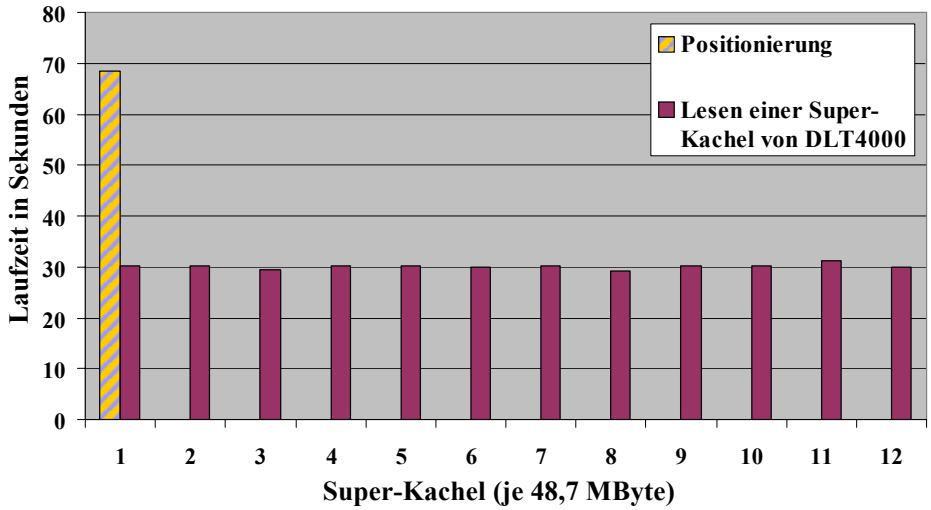


Abbildung 4.11: Bearbeitung einer Bereichsanfrage mit Intra- und Inter-Super-Tile-Clustering in Kombination mit Intra-Query-Scheduling (HEAVEN).

Abbildung 4.12 vergleicht die Gesamtlaufzeit des Ladevorganges der Bereichsanfrage für die einzelnen Optimierungsstufen. Das verdeutlicht eindrucksvoll die Relevanz der bei HEAVEN umgesetzten Optimierungen bezüglich eines effizienten Datenretrievals von tertiären Speichermedien.

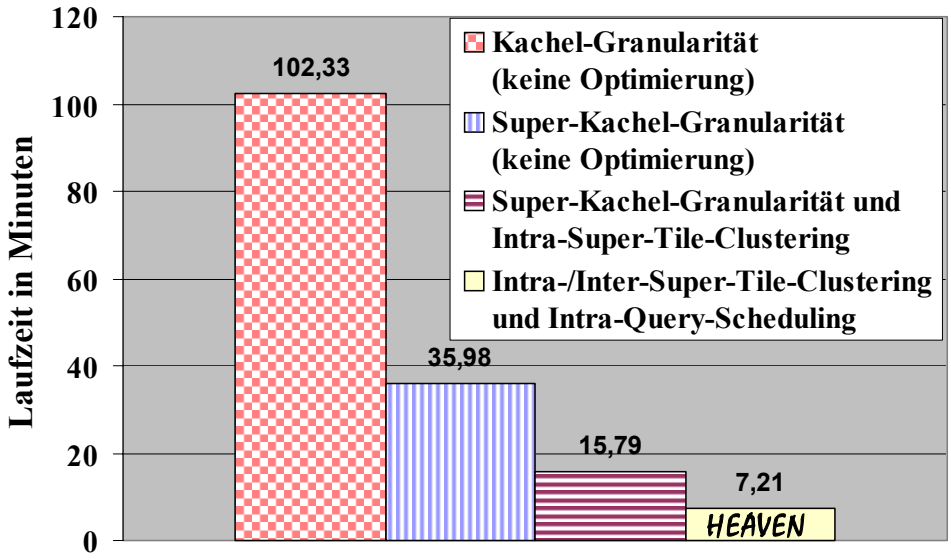


Abbildung 4.12: Vergleich der Ladevorgänge der einzelnen Optimierungsstufen.

4.4.1.2 Bereichsanfragen auf Objektmengen

Wie bereits in Kapitel 3.7.3 behandelt, ist bei einer Datenhaltung auf TS-Medien, ein besonderes Augenmerk auf die Optimierung von Bereichsanfragen auf Objektmengen zu legen. Zur Untersuchung der bei HEAVEN realisierten Optimierung berechnen wir die durchschnittliche, absolute Windgeschwindigkeit über 100 Jahre im Zeitraum von Dezember 1999 bis Dezember 2099. Die Ermittlung der absoluten Windgeschwindigkeit erfolgt durch die Berechnung nach dem Satz des Pythagoras aus den Vektoren der Windgeschwindigkeit $v_Ä$ parallel zum Äquator (Datensatz dkrz4d_vÄ), bzw. der Windgeschwindigkeit v_M parallel zum Meridian (Datensatz: dkrz4d_vM). Die entsprechende RasQL-Anfrage wird wie folgt formuliert:

```
SELECT   avg_cells(
         sqrt( ( vÄ[*:*, 1680:2879, *:*, *:*] * vÄ[*:*, 1680:2879, *:*, *:*] ) +
               ( vM[*:*, 1680:2879, *:*, *:*] * vM[*:*, 1680:2879, *:*, *:*] )))
FROM dkrz4d_vÄ as vÄ, dkrz4d_vM as vM
```

Ausgehend von den Originaldatensätzen wird die Zeitdomäne von 240 Jahren (2.880 Monate) auf 100 Jahre (1.200 Monate) eingeschränkt. Die Bereichsanfrage liegt direkt auf Kachelgrenzen und umfasst für jeden der beiden Datensätze 1.200 Kacheln (je 480 KByte). Das angefragte Gesamtdatenvolumen beträgt 1,1 GByte. Eine Super-Kachel σ enthält 104 Kacheln τ und hat ein Datenvolumen Δ_σ von 48,7 MByte. Beim Exportieren der Super-Kacheln wurde zur Optimierung das Intra- und Inter-Super-Tile-Clustering realisiert.

Abbildung 4.13 zeigt den Zeitbedarf für die Positionierung (C_{Pos}) und das Laden ($C_{read\sigma}$) der relevanten Super-Kacheln von einem TS-Medium. Die beiden Datensätze befinden sich dabei auf dem gleichen Magnetband. Die Anfragebearbeitung erfolgt in diesem Fall durch die RasDaMan typischen alternierenden Vorgehensweise beim Laden von Datenobjekten (Abbildung 3.33, Seite 145). Das bedeutet, dass zunächst die erste Super-Kachel des MDD vÄ (Datensatz: dkrz4d_vÄ) geladen wird, gefolgt von der ersten Super-Kachel des MDD vM (Datensatz: dkrz4d_vM). Anschließend wird die zweite Super-Kachel des MDD vÄ und des MDD vM, usw. benötigt. Bei einem Magnetband sind durch diese Bearbeitungsweise sehr viele kostenintensive Operationen zur Positionierung in Vorwärts- und Rückwärtsrichtung notwendig. Die beiden MDD wurden auf dem Medium nicht direkt aufeinander folgend gespeichert.

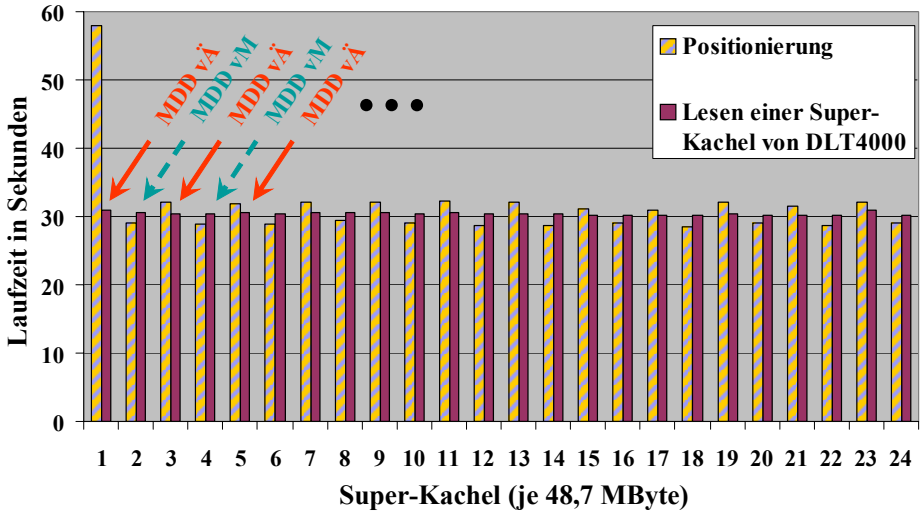


Abbildung 4.13: Anfrage auf Objektmenge bei alternierender Vorgehensweise (RasDaMan).

Der ähnliche Zeitbedarf (Abbildung 4.13) für die einzelnen Operationen zur Positionierung wird durch die gleichen Abstände der Super-Kacheln der beiden MDD auf dem TS-Medium hervorgerufen. Ausgenommen hiervon ist die Initialpositionierung. Das Positionieren in Rückwärtsrichtung auf dem Magnetband ist zeitlich aufwändiger, da beim Positionierungsvorgang zunächst das zu ladende Datenobjekt überspult wird. Danach findet eine Richtungsumkehr statt und die endgültige Positionierung zum Anfang des Datensatzes erfolgt in langsamerer Geschwindigkeit. Bei einem Spulvorgang in Vorwärtsrichtung ist weder das Überspulen des Datensatzes noch eine Richtungsumkehr erforderlich. In der Abbildung 4.13 ist der Vorgang einer Rückwärtspositionierung bei den Super-Kacheln mit ungerader Nummer durch den höheren Zeitbedarf zu erkennen. Ausgenommen hiervon ist die initiale Positionierungsoperation vor der ersten Super-Kachel. Der gesamte Ladevorgang der für die Bereichsanfrage relevanten 24 Superkacheln von T_{on} in den Speicherbereich SHC benötigt 24,8 Minuten.

Optimierte Vorgehensweise bei Objektmengen

Durch das in HEAVEN realisierte Intra-Query-Scheduling erfolgt eine Umsortierung der Anfragereihenfolge und der alternierende Ladevorgang wird bei involvierten TS-Zugriffen durch ein Demand-Prefetching ersetzt. Abbildung 4.14 zeigt den zeitlichen Aufwand der in dieser

Weise veränderte Anfragebearbeitung. Nach der initialen Positionierungsoperation werden zunächst alle für die Anfrage relevanten Super-Kacheln des MDD vÄ (Datensatz: dkrz4d_vÄ) geladen. Anschließend erfolgt eine Positionierung zum Ersten zu ladenden Datensatz des MDD vM (Datensatz: dkrz4d_vM). Gefolgt von den Ladeoperationen der relevanten Super-Kacheln. Das bei der alternierenden Vorgehensweise notwenige Springen zwischen den beiden Datensätzen auf dem Magnetband mit den dadurch notwendigen Positionierungsoperationen entfällt. Der Gesamtvorgang zum Laden der 24 Super-Kacheln von T_{on} in den Speicherbereich SHC erfordert 13,3 Minuten. Gegenüber der alternierenden Vorgehensweise mit einem Zeitaufwand von 24,8 Minuten wird ein Speed-Up von 1,86 erreicht.

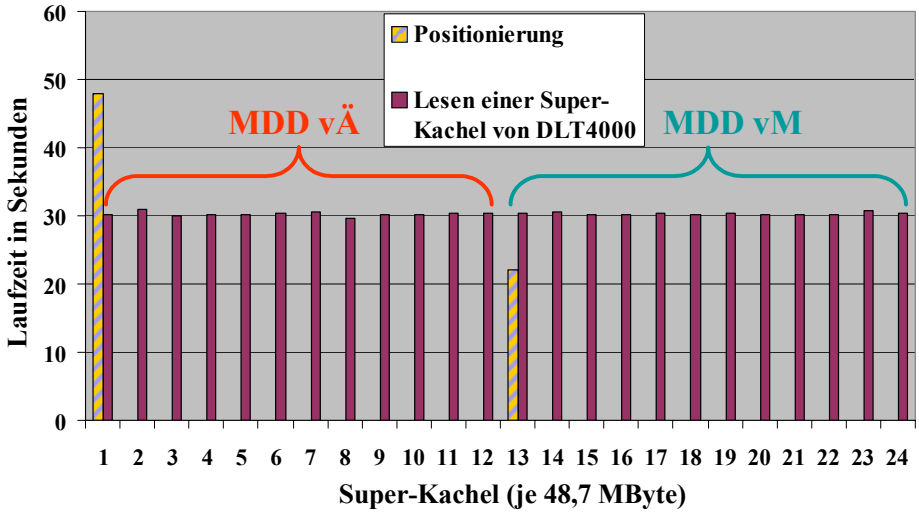


Abbildung 4.14: Anfrage auf Objektmenge mit Demand-Prefetching in Kombination mit Intra-Query-Scheduling (*HEAVEN*).

Die Kombination der bei *HEAVEN* realisierten Optimierungen bewirkt eine herausragende Steigerung der Performance bei der Anfragebearbeitung.

4.4.2 Datenretrieval durch RasDaMan

Nach erfolgter Performance Evaluierung der Ladeoperationen des HSM-Systems TCT wird das Datenretrieval innerhalb RasDaMan untersucht. Die anfallenden Kosten $C_{Raccess}$ setzen sich zusammen aus:

$$C_{Raccess} = C_{load\sigma} + C_{build\tau} + C_{\tau SRC} + C_{MC} + C_{read\tau} + C_{op} + C_{trans}$$

$C_{load\sigma}$ → Ladevorgang der für eine Anfrage q relevanten Super-Kacheln σ aus dem Speicherbereich SHC des HSM-Systems (TCT).

$C_{build\tau}$ → Rückgewinnung der Kacheln (BLOBs) τ aus den Super-Kacheln σ.

$C_{\tau SRC}$ → Speicherung der reproduzierten Kacheln τ in den Speicherbereich SRC.

C_{MC} → Aufwand zum Anpassen der Metadaten und der Cacheverwaltung.

$C_{read\tau}$ → Laden der für eine Anfrage q relevanten Kacheln τ aus dem Speicherbereich SRC, bzw. SR in den Primärspeicher P.

C_{op} → Ausführen der entsprechenden RasDaMan Operationen (Kapitel 2.5.5 und 2.5.6).
C_{trans} → Transfer des Ergebnisses der Anfrage q zur Applikation.

Gegenüber der originalen RasDaMan Version fallen bei einem Datenretrieval vom TS-Medium zusätzlich die Kosten für das Laden der relevanten Super-Kacheln σ aus dem Speicherbereich SHC des HSM-Systems ($C_{load\sigma}$), die Rückgewinnung der Kacheln τ aus den Super-Kacheln σ ($C_{build\tau}$), die Speicherung der reproduzierten Kacheln τ in den Speicherbereich SRC ($C_{\tau SRC}$) und der Aufwand zur Anpassung der Metadaten und der Cacheverwaltung (C_{MC}) an. Zur Evaluierung der Retrievalperformance wird durch eine geometrische Operation die Windgeschwindigkeit $v_Ä$ parallel zum Äquator (Datensatz: dkrz4d_vÄ) im Zeitraum von Dezember 1999 bis Dezember 2099 (100 Jahre) angefragt. Die entsprechende RasQL-Anfrage lautet:

SELECT vÄ[*:*, 1680:2879, *:*, *:*] FROM dkrz4d_vÄ as vÄ

Die Zeitdomäne des vierdimensionalen Testdatensatzes dkrz4d_vÄ mit einem Gesamtvolumen von 1,32 GByte wird auf 100 Jahre (1.200 Monate) eingeschränkt. Diese Bereichsanfrage liegt direkt auf den Kachelgrenzen und umfasst 1.200 Kacheln (je 480 KByte), mit einem Gesamtdatenvolumen von 562,5 MByte. Ein Verschnitt υ_τ tritt somit nicht auf.

4.4.2.1 Testumgebung Variante A

Der Zeitbedarf für die Kostenstellen $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} mit lokalem Cache-Speicherbereich SHC (Testumgebung Variante A) sind in Abbildung 4.15 dargestellt. Die kombinierten Kosten $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} entsprechen der Funktion *replicate*(σ, SHC, SRC).

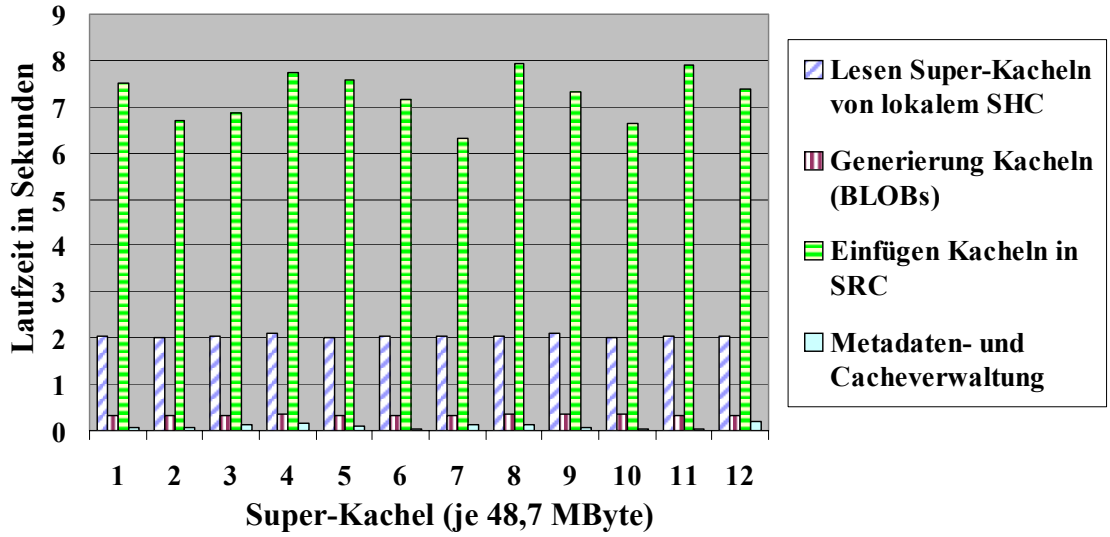


Abbildung 4.15: Zeitbedarf für die Kostenstellen $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} mit lokalem Cache-Speicherbereich SHC (Testumgebung A).

Der Lesevorgang ($C_{load\sigma}$) der Super-Kacheln von einem lokalem Cache-Speicherbereich SHC erreicht eine Datenrate von 23,64 MByte/s. Die Rückgewinnung ($C_{build\tau}$) der Kacheln τ aus den einzelnen Super-Kacheln erfolgt im Hauptspeicher. Je Super-Kachel werden 104 Kacheln (BLOBs) generiert. Das Einfügen ($C_{\tau SRC}$) der Kacheln in den Cache-Speicherbereich von RasDaMan SRC erfolgt mit einer Datenrate von 6,67 MByte/s. Insgesamt werden 1.248 Kacheln mit einem Datenvolumen von 585 MByte eingefügt. Die Bereichsanfrage weist ein Datenvolumen von 562,5 MByte auf und umfasst 1.200 Kacheln. Das entspricht einem Verschnitt υ_σ von 3,84 % (es gilt $\upsilon_\sigma = \upsilon_{\sigma\setminus\tau}$, da $\upsilon_\tau = 0$). Der Aufwand für die Metadaten- und Cacheverwaltung C_{MC} liegt je Super-Kachel im zweistelligen Millisekundenbereich. Der Gesamtzeitaufwand beträgt 116,67 Sekunden.

4.4.2.2 Testumgebung Variante B

Wird der Cache-Speicherbereich SHC entsprechend der Testumgebung Variante B über NFS an RasDaMan angebunden, so erhöht sich, wie in Abbildung 4.16 ersichtlich, ausschließlich der Zeitbedarf für das Lesen der Super-Kacheln vom SHC ($C_{load\sigma}$). Die erreichte Datenrate sinkt auf 6,57 MByte/s. Dadurch erhöht sich der Gesamtzeitaufwand der Testumgebung B gegenüber der Testumgebung A von 116,67 Sekunden um 62,79 Sekunden auf 179,46 Sekunden.

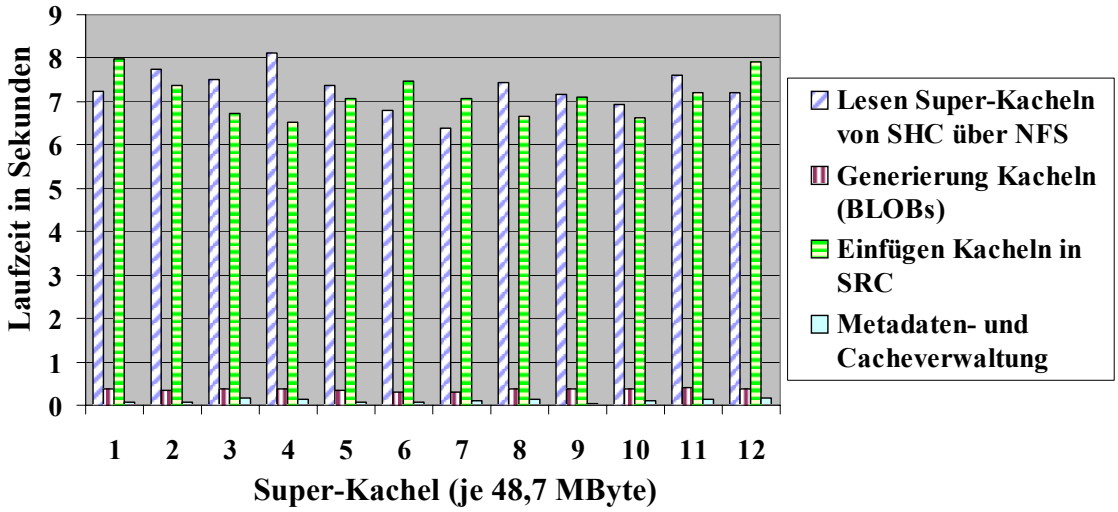


Abbildung 4.16: Zeitbedarf für die Kostenstellen $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} bei SHC Anbindung über NFS (Testumgebung B).

Die Kosten $C_{read\tau}$ für das Laden der für eine Anfrage q relevanten Kacheln τ aus dem Speicherbereich SRC, bzw. SR in den Primärspeicher P, C_{op} für die Ausführung der Operationen und C_{trans} für den Transfer der Ergebnisse zur Applikation entsprechen der Vorgehensweise der ursprünglichen RasDaMan Version ohne TS-Anbindung. Bei der Bearbeitung einer Anfrage erreicht die ursprüngliche Version von RasDaMan beim Lesen ($C_{read\tau}$) von Kacheln (BLOBs) aus der Datenbank durch die alternierenden Vorgehensweise beim Laden (Kapitel 3.7.1) eine Datenrate von ca. 8 MByte/s. Die Kosten C_{op} für Bearbeitung einer Operation

bewegen sich von wenigen Millisekunden für eine geometrische Operation, bis hin zu mehreren Stunden für komplexe Operationen. Nähere Information über mögliche Operationen sind in Kapitel 2.5.5 oder bei [Rasd02, Rits99, Baum99, Baum04] zu finden. Der zeitliche Aufwand für den Transfer eines Ergebnisses zu einer angebundenen Applikation (C_{trans}) hängt stark vom zu übertragenden Datenvolumen, dem verwendeten Protokoll RPC (Remote Procedure Call) oder HTTP (Hypertext Transfer Protocol) und der Netzwerkanbindung ab. Im Folgenden wird das Datenretrieval über alle Speicherhierarchien näher betrachtet.

4.4.2.3 Datenretrieval über alle Speicherhierarchien

Um das Zugriffsverhaltens auf die unterschiedlichen Speicherbereiche SRC, SHC und T_{on} zu untersuchen, wird der Speicherbereich SR als Referenzmarke herangezogen. Abbildung 4.17 vergleicht die Gesamtlaufzeit (C_q) folgender Bereichsanfrage bei unterschiedlich involvierten Speicherebenen:

SELECT vÄ[*:*, 1680:2879, *:*, *:*] FROM dkrz4d_vÄ as vÄ

Die Zeitdomäne des vierdimensionalen Testdatensatzes dkrz4d_vÄ mit einem Gesamtvolumen von 1,32 GByte wird auf 100 Jahre (1.200 Monate) eingeschränkt. Der angefragte Bereich des Datensatzes weist ein Volumen von 562,5 MByte auf.

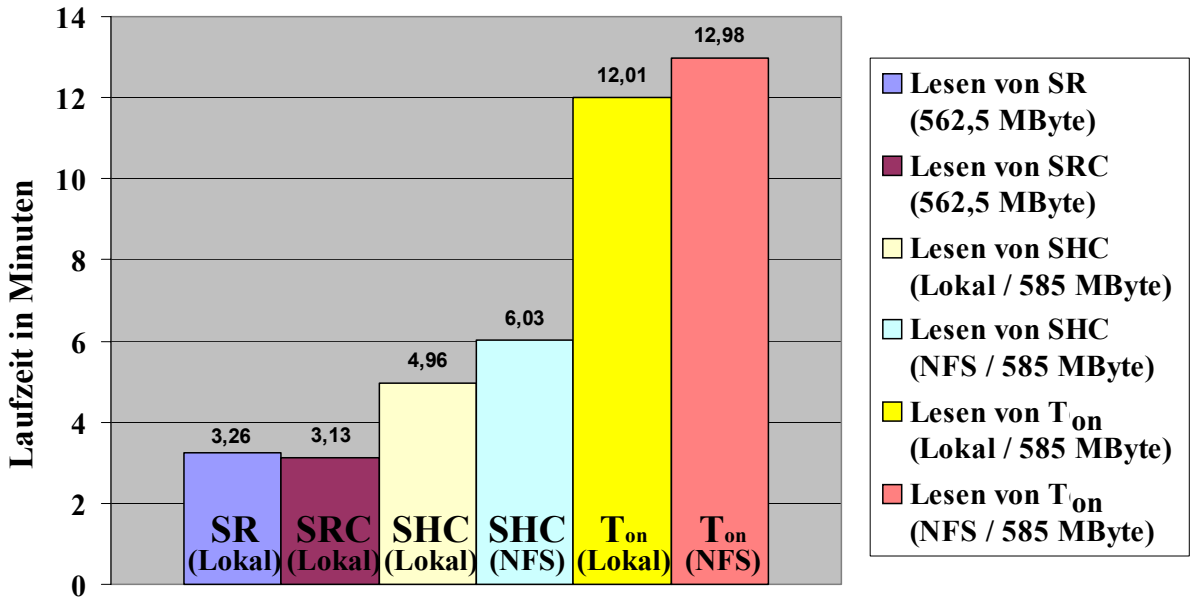


Abbildung 4.17: Gesamtlaufzeiten C_q der Durchführung einer Bereichsanfrage bei unterschiedlich involvierten Speicherebenen.

Lesen von SR (562,5 MByte)

Zunächst wird die oben genannte Anfrage mit der herkömmlichen RasDaMan Version ohne TS-Anbindung bearbeitet. Da es sich bei der gestellten Bereichsanfrage um eine geometrische Operation handelt, bei der ausschließlich die Domäne eingeschränkt wird, liegen die Kosten

C_{op} im Bereich von ca. 20 Millisekunden. Für die Anfrage werden 1.200 Kacheln mit einem Datenvolumen von insgesamt 562,5 MByte angefragt (Δ_τ = 480 KByte). Das Laden ($C_{read\tau}$) der Daten aus dem Sekundärspeicher des konventionellen DBMS von RasDaMan (Speicherbereich SR) in den Primärspeicher P dauert ca. 1,16 Minuten. Bei einem Datenvolumen von 562,5 MByte wird eine Datenrate von etwa 8,1 MByte/s erreicht. Der Transfer (C_{trans}) des durch die geometrische Operation eingeschränkten Datenbereiches (562,5 MByte) zur Applikation benötigt 2,1 Minuten. Das ergibt für die RPC-Anbindung eine Datenrate von 4,46 MByte/s. Insgesamt ist mit einer originalen RasDaMan Version für die Durchführung der Bereichsanfrage eine Zeit von 3,26 Minuten erforderlich.

Lesen von SRC (562,5 MByte)
Wie zu erwarten, zeigt das Laden von Datenobjekten aus dem TS-Cache-Bereich des konventionellen DBMS von RasDaMan (Speicherbereich SRC) eine ähnliche Performance wie das Laden aus dem Bereich SR (Abbildung 4.17). Das liegt an der Tatsache, dass die beiden Bereiche nur aus logischer Sichtweise zwei disjunkte Teilmengen darstellen. Eine Unterscheidung erfolgt durch zusätzliche Metadaten. Auf physikalischer Ebene liegen die Daten in der gleichen Tabelle des konventionellen DBMS. Die Verteilung des Zeitbedarfes auf die drei Kostenstellen $C_{read\tau}$, C_{op} und C_{trans} sind ähnlich angesiedelt, wie beim Lesen vom Speicherbereich SR. Die Gesamtlaufzeit der Anfrage beträgt 3,13 Minuten.

Lesen von SHC (585 MByte)
Beim Laden von Datenobjekten aus dem Cache-Speicher des HSM-Systems (SHC) sind zwei Möglichkeiten zu unterscheiden. Einerseits kann sich dieser Sekundärspeicherbereich auf dem lokalen Rechner (Testumgebung Variante A) befinden oder über NFS (Testumgebung Variante B) angebunden sein. Abbildung 4.17 zeigt beide Varianten. Bei Zugriff auf den Speicherbereich SHC steigt aufgrund der größeren Zugriffsgranularität (σ) das zu ladende Speichervolumen in diesen Bereichen auf 585 MByte. Das entspricht zwölf Super-Kacheln mit einer Kapazität Δ_σ von je 48,7 MByte. Der Verschnitt υ_σ beträgt somit 3,84 % (es gilt $\upsilon_\sigma = \upsilon_{\sigma\setminus\tau}$, da $\upsilon_\tau = 0$). Der zeitliche Mehraufwand beim Laden von Datenobjekten aus dem Speicherbereich SHC, gegenüber SRC mit den Kosten $C_{read\tau}$, C_{op} und C_{trans}, entsteht durch die zusätzlich anfallenden Kosten $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} beim Transfer der Super-Kacheln vom Speicherbereich SHC in den Bereich SRC. Abbildung 4.15 zeigt für einen lokalen SHC (Testumgebung Variante A) eine detaillierte zeitliche Aufschlüsselung der Kosten $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC}. Entsprechend der Abbildung 4.17 beträgt die Gesamtlaufzeit der Bereichsanfrage 4,96 Minuten. Bei einer NFS-Anbindung des Speicherbereiches SHC (Testumgebung Variante B) ist aufgrund der notwendigen Netzwerkübertragung ein weiterer Anstieg der Laufzeit festzustellen. Die Gesamtlaufzeit steigt auf 6,03 Minuten. Der zeitliche Bedarf der einzelnen Kostenstellen $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} ist in Abbildung 4.16 dargestellt. Die Zeitdifferenz in der Gesamtlaufzeit bei der durchgeführten Bereichsanfrage, unter Verwendung eines lokalen, bzw. NFS gebundenen SHC, beträgt 1,07 Minuten. Das entspricht in etwa der gleichen Zeitdifferenz (1,05 Minuten), die beim Vergleich der Gesamtlaufzeiten der Messungen aus Abbildung 4.15 und Abbildung 4.16 auftreten. Da bei der Messung aus Abbildung 4.17 alle entsprechenden Kostenstellen ($C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$, C_{MC}, $C_{read\tau}$, C_{op} und C_{trans}) mit einfließen

macht sich der Unterschied in der Performance gegenüber den Messungen aus Abbildung 4.15 und Abbildung 4.16 mit den dort aufgeschlüsselten Kostenstellen $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} nicht so stark bemerkbar, obwohl die Zeitdifferenzen in etwa gleich sind.

Lesen von T_{on} (585 MByte)

Befindet sich keines der angeforderten Datenobjekte in einem der Cache-Bereiche SRC oder SHC, so müssen die Super-Kacheln aufwändig von einem TS-Medium T_{on} geladen werden. Diesen Sachverhalt zeigen die beiden rechten Balken der Abbildung 4.17. Gemäß der Art der Anbindung des Cache-Bereiches SHC (Lokal oder über NFS) an RasDaMan werden, wie erwartet, mit 12,01 Minuten und 12,98 Minuten unterschiedliche Antwortzeiten erzielt. Das entspricht in etwa der gleichen Zeitdifferenz, die beim Vergleich der Gesamtlaufzeitenzeiten der Messungen aus Abbildung 4.15 und Abbildung 4.16 auftreten. Da bei der Messung aus Abbildung 4.17 alle Kostenstellen (C_{Verw}, C_{Pos}, $C_{read\sigma}$, $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$, C_{MC}, $C_{read\tau}$, C_{op} und C_{trans}) mit einfließen, macht sich der Unterschied in der Performance gegenüber den Messungen aus Abbildung 4.15 und Abbildung 4.16 mit den Kostenstellen $C_{load\sigma}$, $C_{build\tau}$, $C_{\tau SRC}$ und C_{MC} nicht so stark bemerkbar, obwohl die Zeitdifferenzen in etwa gleich sind.
Das Datenretrieval von einem TS-Medium wurde entsprechend der bei *HEAVEN* realisierten optimierten Anfragebearbeitung mit Intra- und Inter-Super-Tile-Clustering in Kombination mit Intra-Query-Scheduling durchgeführt. Den Aufwand ($C_{TCT\mathit{access}} = C_{Verw} + C_{Anlauf} + C_{Pos} + C_{read\sigma}$) zum Laden der relevanten Super-Kacheln von einem TS-Medium und zum Speichern im Cache-Bereich SHC zeigt Abbildung 4.11. Bei der durchgeführten Messung wurden keine Anlaufkosten berücksichtigt. Das TS-Medium (T_{on}) mit den relevanten Super-Kacheln befand sich bereits in der Lesestation. Zu den Kosten $C_{TCT\mathit{access}}$, die beim Laden der Super-Kacheln von T_{on} in den Speicher-Bereich SHC anfallen, addieren sich die Kosten $C_{R\mathit{access}} = C_{load\sigma} + C_{build\tau} + C_{\tau SRC} + C_{MC} + C_{read\tau} + C_{op} + C_{trans}$. Der zeitliche Aufwand dieser Kosten gleicht den entsprechenden Kosten der bereits beschriebenen Speicherbereiche SR, SRC, bzw. SHC.

Der durchgeführte Vergleich der Retrievalperformance bei unterschiedlichen Speicherebenen, spricht klar für die in Kapitel 3.9 beschriebenen Realisierung einer Caching-Hierarchie: Bestehend aus den Cache-Bereichen SRC und SHC. Werden Datenobjekte im Speicherbereich SRC vorgehalten, so verbessert sich bei der Testumgebung B die Retrievalperformance um ca. 75,9 % (Speed-Up: 4,15) gegenüber dem Ladevorgang von tertiären Speichermedien. Wird Testumgebung A vorausgesetzt, so wird eine Steigerung der Performance von 74 % (Speed-Up: 3,84) ermöglicht. Der Performancegewinn bei einem Caching im Speicherbereich SHC, beträgt in Abhängigkeit von den Übertragungskosten (lokale Festplatte oder NFS) gegenüber dem Speicherbereich T_{on} zwischen 53,5 % (Speed-Up: 2,15) und 58,7 % (Speed-Up: 2,42). Die Laufzeitbeschleunigung beim Speicherbereich SRC gegenüber SHC, erreicht einen Wert zwischen 36,7 % (Speed-Up: 1,58) und 48,2 % (Speed-Up: 1,93).

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them.
Sir. William Henry Bragg (1862 - 1942)

Kapitel 5 Diskussion

Dieses Kapitel grenzt das entwickelte System HEAVEN (Kapitel 3) gegenüber den Systemen SDM, MDBS, APRIL, StorHouse/RM, CERA und Postgres (Kapitel 2.4 und [Rein03b]) ab und diskutiert die gewonnenen Ergebnisse. Erkenntnisse aus dem ESTEDI-Projekt fließen in den Vergleich der Systeme mit ein. Als entscheidendes Bewertungskriterium gilt eine effiziente Verwaltung und Speicherung von multidimensionalen Array-Daten. Es werden essentielle Defizite und Probleme der in Kapitel 2.4 bzw. der bei [Rein03b] untersuchten Systeme gegenüber HEAVEN aufgezeigt.

Zunächst werden wichtige Eigenschaften hinsichtlich einer effizienten Verwaltung von multidimensionalen Array-Daten vorgestellt. Diese Anforderungen wurden aus den im Kapitel 1.1 aufgeführten Problem beim Retrieval und bei der Speicherung großer Datenvolumen entwickelt. Zur Bewertung und zum Vergleich der Systeme gegenüber HEAVEN werden folgende Kriterien herangezogen:

- Können **Metadaten** durch ein DBMS effizient gespeichert und verwaltet werden und wird dadurch ein schnelles Auffinden der angefragten Daten möglich?
- Werden multidimensionale, diskretisierte Array-Daten **(MDD) im DBMS** mit einer automatisierten, direkten Kopplung zu TS-Systemen gespeichert, um die Vorteile der Datenhaltung durch ein DBMS zu nutzen?
- Besteht die Möglichkeit, große Objekte in gleichförmige Teilbereiche zu unterteilen (**Chunking**)? Das ist die Voraussetzung für eine effiziente Bearbeitung von Bereichsanfragen.
- Ist es möglich, diese Unterteilung (Chunking) in einer allgemeineren Form zu realisieren (**Tiling**), um durch eine exaktere Datenmodellierung den Zugriffsmustern der Anwender besser zu entsprechen?
- Ist die **Objektgranularität** des Datenzugriffes, entsprechend den Besonderheiten der verwendeten Speicherorte, wie Festplatte, bzw. TS-System, angepasst?
- Können Anwender Teilbereiche der Datenobjekte anfragen (**Bereichsanfrage**) und vor allem, werden auch nur diese Teilobjekte übertragen?

- Verfügt das System über eine spezielle, multidimensionale Anfragesprache (Multidimensional Query Language, **MQL**[57]), um Anfragen im DBMS zu bearbeiten oder wird das DBMS nur als Speicher- und Transaktionsmanager verwendet?
- Werden die Daten entsprechend einer multidimensionalen **Clustering**-Strategie auf den TS-Medien abgelegt um Zugriffskosten zu minimieren?
- Erfolgt ein durch **Query-Scheduling** optimierter Zugriff auf Daten, die sich auf einem TS-System befinden?
- Findet ein **Caching** der Daten mit optimierter Verdrängungsstrategie für TS-Systeme statt?
- Wird bei der Anfragebearbeitung auf **vorberechnete Werte** (Ergebnisse) berechnungsintensiver Operatoren zurückgegriffen, um eine Performancesteigerung zu erzielen?
- Kann die zu übertragende Datenmenge durch eine Transfer-**Kompression** zwischen TS-System und DBMS-Server, bzw. DBMS-Server und Client verringert werden?
- Ist es möglich, die Anfragebearbeitung durch Nutzung von Verarbeitungs- und E/A-**Parallelität** zu beschleunigen?
- Wie gut ist das untersuchte System hinsichtlich der **Skalierbarkeit**?
- Ist eine gute und einfache **Erweiterbarkeit** des Systems gewährleistet?

In der Tabelle 5.1 werden die untersuchten Systeme hinsichtlich der genannten Kriterien gegenübergestellt und bewertet. Dabei steht das Symbol ✓ für die Unterstützung der jeweiligen Eigenschaften. Das Symbol ~ bedeutet, die Eigenschaften werden teilweise oder nur rudimentär erfüllt. Neben dem in dieser Arbeit entwickeltem System HEAVEN und den in Kapitel 2.4 beschriebenen und bewerteten Systemen StorHouse/RM, CERA und Postgres wurden bei [Rein03b] darüber hinaus die Systeme SDM, MDMS und APRIL auf ihre Tauglichkeit untersucht. Die Systeme wurden entsprechend ihrer Eignung für die Speicherung und das Retrieval multidimensionaler Array-Daten (wenig, bis sehr gut geeignet) sortiert.

	Metadaten	MDD im DBMS	Chunking	Tiling	Objektgranularität	Subset Retrieval	MQL	Clustering	Query-Scheduling	Caching	Vorberechnete Werte	Kompression	Parallelität	Skalierbarkeit	Erweiterbarkeit
SDM	✓												~	~	~
MDMS & APRIL	✓		✓			✓		~	~	~			~	~	~
StorHouse/RM	✓	✓							~					~	
CERA	✓	✓												~	
Postgres	✓	✓	✓			✓		~	✓	✓					
HEAVEN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Tabelle 5.1: Vergleich der untersuchten Systeme gegenüber HEAVEN.

[57] MQL ist keine standardisierte Anfragesprache, wie SQL. Eine Ausprägung einer multidimensionalen Anfragesprache ist die für RasDaMan entwickelte Anfragesprache RasQL (Kapitel 2.5).

Systeme SDM, MDMS und APRIL

Die Systeme SDM (Scientific Data Manager), MDMS (Meta-Data Management System) und APRIL (A Parallel Run-Time Library for Tape-Resident Data) realisieren keine durchgehende Verschmelzung eines DBMS mit einem TS-System. Sie verwenden ein relationales DBMS ausschließlich zur Verwaltung der Metadaten. Die einzelnen Daten (z.B. MDD) werden nach wie vor als Dateien auf die TS-Medien gespeichert. Nur das Auffinden der Daten wird vereinfacht. Es werden keine bekannten Vorteile der Datenverwaltung durch DBMS, wie das Transaktionskonzept oder die Zugriffskontrolle usw. unterstützt. Durch die fehlende multidimensionale Anfragesprache (MQL) können die Systeme für Array-Daten ausschließlich zur Speicherverwaltung eingesetzt werden.

Das System SDM (Tabelle 5.1) verfügt weder über die Möglichkeit, große Datenobjekte zu unterteilen, um Bereichsanfragen zu ermöglichen, noch über eine dem jeweiligen Speicherort angepasste Objektgranularität. Weiterhin wurden keine Konzepte zur Optimierung, wie Clustering, Query-Scheduling oder Caching mit eingebracht. Durch die Verwendung der *Message Passing Interface*-Schnittstelle MPI-IO wird E/A-Parallelität bei der Anbindung des Dateisystems realisiert. Eine Verarbeitungsparallelität ist nicht möglich. Durch das verwendete HSM-System HPSS (High Performance Storage System) wird eine gewisse Skalierbarkeit in Bezug auf das Datenvolumen erreicht. SDM ist hinsichtlich neuer Benutzer und neuer Applikationen nicht einfach zu erweitern.

Die Systeme MDMS und APRIL (Tabelle 5.1) werden meist kombiniert eingesetzt. Dabei realisiert MDMS die Verwaltung und Speicherung der Metadaten und APRIL sorgt für die Anbindung einer Applikation an ein HSM-System. Der wesentliche Vorteil von MDMS und APRIL gegenüber SDM ist die Möglichkeit, große multidimensionale Daten in gleichförmige Teilobjekte (in jeder Dimension) aufzuspalten und als separate Dateien auf Tertiärspeicher abzulegen (Chunking). Dadurch können Bereichsanfragen durchgeführt werden. Allerdings wurde keine dem jeweiligen Speicherort angepasste Objektgranularität verwirklicht. Die implementierten Konzepte, wie Clustering, Query-Scheduling und Caching, sind nur rudimentär ausgeprägt. Ebenso wie bei dem SDM-System wird E/A-Parallelität und eine gewisse Skalierbarkeit in Bezug auf das Datenvolumen erreicht. Allerdings ist MDMS und APRIL hinsichtlich neuer Benutzer und neuer Applikationen nicht einfach erweiterbar.

Eine weiterführende, umfassende Beschreibung und Bewertung der Systeme SDM, MDMS und APRIL ist bei [Rein03b] zu finden.

Systeme StorHouse/RM, CERA und Postgres

Im Gegensatz zu den Systemen SRM, MDMS und APRIL werden bei den Systemen StorHouse/RM (StorHouse Relational Manager), CERA (Climate and Environmental Data Retrieval and Archiving System) und Postgres nicht nur die Metadaten, sondern auch die eigentlichen Daten (z.B. MDD) innerhalb eines relationalen DBMS gespeichert und verwaltet. Durch eine direkte Kopplung mit HSM-Systemen kann die Datenbasis auf tertiäre Speichermedien ausgelagert werden. Da keines der Systeme über eine multidimensionale Anfra-

gesprache (MQL) verfügt können diese Produkte für Array-Daten nur zur Speicherverwaltung eingesetzt werden.

StorHouse/RM (Tabelle 5.1 und Kapitel 2.4) bietet die Möglichkeit, multidimensionale Array-Daten (MDD) als LOBs in Tabellen zu speichern und zu verwalten. Da das System StorHouse/RM für relationale Daten entwickelt wurde, gibt es allerdings keine weitere Unterstützung für Array-Daten. So ist es nicht möglich, Teilobjekte zu speichern, eine dem jeweiligen Speicherort angepasste Datengranularität zu wählen oder Bereichsanfragen effizient zu unterstützen. Ein Hauptanwendungsgebiet für StorHouse/RM sind Data Warehouse Systeme. Da normalerweise nur historische Daten und nicht operative Daten auf TS-Medien ausgelagert werden, fehlen Optimierungen in Bezug auf ein effizientes Datenretrieval, wie zum Beispiel Clustering, Caching oder parallele Anfragebearbeitung. Lediglich Query-Scheduling wurde rudimentär realisiert. Die Anbindung an ein HSM-System lässt auf eine gewisse Skalierbarkeit hinsichtlich des Datenvolumens schließen. Aufgrund der Inkompatibilität von StorHouse/RM zu HSM-Systemen anderer Hersteller ist eine Erweiterung schwer zu erreichen.

Das System CERA (Tabelle 5.1 und Kapitel 2.4) wurde im Bereich der Klimaforschung für die Verwaltung von MDD entwickelt. Durch das modellierte Datenbankschema für Metadaten ist eine semantikorientierte Möglichkeit des Datenretrievals geschaffen worden. Ein gravierender Nachteil des CERA-Systems ist die hohe Granularität des Datentransfers zwischen DBMS und TS-System, da nur komplette Tablespaces aus-, bzw. eingelagert werden können. In einer Tablespace werden normalerweise mehrere Simulationsergebnisse, jeweils als BLOBs, gespeichert. Eine dem jeweiligen Speicherort angepasste Objektgranularität besteht nicht. Auch Bereichsanfragen auf Teilbereiche von MDD werden nicht unterstützt. Weiterhin wurden in CERA keine Mechanismen, wie zum Beispiel Clustering, Query-Scheduling, Caching oder parallele Anfragebearbeitung, zur Optimierung des Datenretrievals integriert. Durch die Verwendung eines HSM-Systems wird eine gewisse Skalierbarkeit hinsichtlich des Datenvolumens erreicht. Eine Erweiterung ist nur mit viel Aufwand zu realisieren.

Das DMBS Postgres (Tabelle 5.1 und Kapitel 2.4) verfügt über gute Voraussetzungen hinsichtlich der Speicherung und Verwaltung multidimensionaler Daten (MDD). Es können sowohl Metadaten, als auch MDD gespeichert werden. Einen großen Vorteil bietet das Konzept des Chunking von Objekten. Durch die Speicherung von Teilobjekten können Bereichsanfragen durchgeführt werden. Die Objektgranularität wurde nicht dem jeweiligen Speicherort angepasst. Im Bereich der Optimierung wurde ein Clustering von Objekten ansatzweise verwirklicht. Weiterhin berücksichtigt ein Scheduler bei der Bestimmung der Ausführungsreihenfolge von Anfragen den Inhalt des Caches bei der Entscheidung, welches Datenobjekt als nächstes vom TS-System geladen werden soll. Möglichkeiten zur parallelen Anfragebearbeitung wurden nicht bedacht. Da keine automatisierten Medienbibliotheken angebunden wurden ist keine akzeptable Skalierbarkeit hinsichtlich des Datenvolumens möglich. Auch die Erweiterbarkeit lässt Wünsche offen, da für jedes TS-System ein eigenständiger Gerätemanager implementiert werden muss.

Eine ausführliche Beschreibung und Bewertung der Systeme StorHouse/RM, CERA und Postgres hinsichtlich der Verwaltung und des Retrievals von multidimensionalen Array-Daten enthält Kapitel 2.4 und [Rein03b].

System HEAVEN

Das System HEAVEN geht einen anderen Weg: Es fusioniert das speziell für multidimensionale Array-Daten (MDD) entwickelte DBMS RasDaMan mit Tertiärspeichersystemen. Bei der Entwicklung von HEAVEN wurden von Anfang an die Bedürfnisse der Hochleistungsrechenzentren berücksichtigt. So konnten zum Beispiel Erkenntnisse aus der Praxis, durch die im ESTEDI-Projekt gegründete User Interest Group (UIG), im Produkt berücksichtigt werden. Dadurch können Optimierungen speziell für multidimensionale Array-Daten bei der Anbindung von TS-Systemen mit eingebracht werden. Der Systemkatalog von RasDaMan musste dabei um Metadaten angereichert werden, die für die TS-Anbindung relevant sind (Kapitel 3.6). Bei HEAVEN werden die verwalteten MDD direkt im DBMS RasDaMan mit automatisierter, direkter Kopplung zu TS-Systemen gespeichert. Die für die Festplatte optimierte Objektgranularität der vorhandenen Kachelungsstrategien (Tiling mit dem enthaltenen Chunking) wurde für die Speicherung auf tertiäre Speichermedien um das Super-Kachel-Konzept erweitert (Kapitel 3.5). Dabei wird auf eine dem Speicherort angepasste Objektgranularität geachtet. Eine Abbildung von Bereichsanfragen (Subset-Retrieval) auf diese neue Struktur wurde vorgenommen (Kapitel 3.7). Diese ist für den Anwender transparent. Die multidimensionale Anfragesprache von RasDaMan ist um einen neuen Operator (Object-Framing) erweitert worden, der ermöglicht, das zu ladende Datenvolumen erheblich zu reduzieren (Kapitel 3.10). Um unnötige Positionierungsoperationen auf TS-Medien zu vermeiden, konnten Konzepte entwickelt werden, die Daten entsprechend ihrer räumlichen Nähe (Clustering) auf TS-Medien speichern (Kapitel 3.6). Aus dem gleichen Grund wurde auch ein Query-Scheduler für RasDaMan implementiert (Kapitel 3.7). Der Aufbau einer für TS-Systeme optimierten Caching-Hierarchie, reduziert kostenintensive TS-Zugriffe (Kapitel 3.9). Kapitel 3.11 behandelt die Vorteile der Verwendung vorberechneter Ergebnisse bei der Anfragebearbeitung. Bei der Realisierung der Kopplung von RasDaMan mit TS-Systemen ist darauf geachtet worden, dass die in RasDaMan vorhandene Funktionalität, wie die Daten- und Transferkompression, in die Entwicklung mit einbezogen wurden. Eine detaillierte Beschreibung bietet [Dehm02]. Weiterhin sind die Konzepte der parallelen Anfrageausführung für die Anbindung tertiärer Speichermedien genutzt worden (Kapitel 3.7.3, bzw. [Hahn05]). Durch die Kopplung relationaler DBMS und unterschiedlichster HSM-Systeme ist HEAVEN in Bezug auf das zu speichernde Datenvolumen sehr gut skalierbar (Kapitel 3). Nicht zuletzt durch die Möglichkeit, gleichzeitig mehrere HSM-Systeme an RasDaMan zu koppeln. Eine modulare und gekapselte Implementierung der entwickelten Komponenten, garantiert eine gute Erweiterbarkeit von HEAVEN.

Das entwickelte System HEAVEN bietet in vielen Bereichen einen bedeutenden Mehrwert gegenüber den herkömmlichen Systemen wie SDM, MDMS, APRIL, StorHouse/RM, CERA und Postgres. Das wird nicht zuletzt in der Tabelle 5.1 eindrucksvoll herausgestellt. Das folgende Kapitel fasst die umfangreichen Ergebnisse dieser Arbeit zusammen.

The Road goes ever on and on down from the door where it began. Now far ahead the Road has gone, and I must follow, if I can, pursuing it with eager feet, until it joins some larger way where many paths and errands meet. And whither then? I cannot say.
J.R.R. Tolkien (1892-1973)

Kapitel 6 Zusammenfassung

Viele Phänomene der Natur mit ihren Messwerten, werden als Array-Daten mit entsprechender Dimensionalität modelliert: Dazu zählen Klimasimulationen, von Satelliten übertragene Daten der Atmosphäre, Berechnungen der Strömungsdynamik, Simulationen chemischer Prozesse oder Simulationen in der Kosmologie und der Genetik. Auch umfassende wissenschaftliche Experimente und Simulationen an Großrechnern, generieren multidimensionale Rasterdaten. Als eine gemeinsame Charakteristik gilt das große Datenvolumen, das gespeichert werden muss. Der Datenumfang kann mehrere hundert Terabyte (10^{12} Byte), bis hin zu mehreren Petabyte (10^{15} Byte) erreichen. Bei CERN (Conseil Europeen pour la Recherche Nucleaire), einer europäischen Organisation für Nuklear-Forschung in der Schweiz, werden zum Beispiel jedes Jahr etwa 3 Petabyte neue Daten gespeichert. Ab dem Jahr 2007 beträgt der jährliche Datenzuwachs etwa 10 Petabyte [HS03]. Eine Datenhaltung auf Sekundärspeicher (Festplatten) ist aufgrund des sehr großen Datenvolumens nicht möglich. Typischerweise werden die Daten als Dateien in automatisierten Tertiärspeichersystemen (TS-Systemen), so genannten hierarchischen Speichermanagement Systemen (HSM-Systemen), mit bis zu tausenden von Magnetbändern, optischen oder magnetooptischen Medien gespeichert. Obwohl die mittlere Zugriffszeit dieser Art von Speichertechnologien, verglichen mit Festplatten, relativ langsam ist, sind Tertiärspeicher bei riesigen Datenvolumina der gegenwärtige und auch zukünftige Entwicklungsstand. Das liegt vor allem an dem viel günstigeren €/GByte-Preis gegenüber Festplatten.

Probleme bereitete bisher nicht das Speichern dieser großen Datenvolumina, sondern der effiziente, selektive Zugriff auf die Daten und die eigentliche Datenanalyse. Die Probleme beim Datenretrieval lagen vor allem in der unzureichenden Datenorganisation. Gewonnene Rohdaten wurden meist direkt in der Ordnung des Generierungsprozesses als Datei auf Tertiärspeichermedien abgelegt. Diese „naive“ Speicherung in der Generierungsordnung ist nicht für einen direkten, anwendungsorientierten Datenzugriff geeignet. Zur Beantwortung wissenschaftlicher Fragestellungen mussten zunächst verarbeitbare Daten mit hohem Aufwand aus den im Tertiärspeicher abgelegten Rohdaten-Dateien extrahiert werden. Bearbeitungszeiten von einigen Wochen, bis hin zu Monaten, waren dabei nicht ungewöhnlich [Laut02]. Das lag auch daran, dass Wissenschaftler mit Dateien, Verzeichnissen und unterschiedlichen, oft

komplexen Datenformaten umgehen und ihre Daten meist selbst verwalten mussten. Ein weiter Kritikpunkt war die hohe Zugriffsgranularität auf Datenobjekte, da nicht auf Teilbereiche von Datenobjekte zugegriffen werden konnte. Dabei benötigten Wissenschaftler nur etwa ein bis zehn Prozent der angeforderten Daten für die weitere Verarbeitung [GKDT02, LT01]. Auch fehlte eine zentrale Möglichkeit zur multidimensionalen Analyse der gespeicherten Rasterdaten.

Die hier aufgeführten Probleme gehören mit dem in dieser Dissertation entwickeltem System HEAVEN der Vergangenheit an. HEAVEN realisiert eine aktive ***h**ierarchische Sp**e**icher- und **A**rchi**v**ierungsumgebung für multidim**e**nsionale Array Date**n**bank Management Systeme*. „Aktiv“ bedeutet in diesem Kontext, dass Anfragen über alle Ebenen der Speicherhierarchie automatisch beantwortet werden und keine weitere Nutzerinteraktion notwendig ist. Mit HEAVEN wird somit der Übergang von einer herkömmlichen, dateisystembasierten Datenhaltung auf eine datenbankbasierte Datenhaltung vollzogen. Beispiele der damit verbundenen Vorteile sind flexible und einheitliche Zugriffsmöglichkeiten, Zugriffskontrolle und die Vermeidung von Redundanz und Inkonsistenz. Die Analyse der Array-Daten erfolgt zentral durch das DBMS mittels einer integrierten, multidimensionalen Anfragesprache. Die bisher notwendigen Werkzeuge, bzw. Arbeitsschritte eines Anwenders, werden reduziert. Wissenschaftler müssen nicht mehr mit Dateien, Verzeichnissen, Dateinamen und Dateiformaten umgehen. Sie können Daten auf einer adäquaten semantischen Ebene anfragen. Einziges Problem, das gegen den Einsatz von Datenbanksystemen spricht, ist das enorme Datenvolumen von mehreren hundert Terabyte bis hin zu mehreren Petabyte. Heute sind Gigabyte-Datenbanken üblich, Terabyte-Datenbanken die aktuelle Herausforderung und Petabyte-Datenbanken ein wichtiges Forschungsgebiet.

Zentrales Thema dieser Arbeit war daher die Realisierung einer effizienten, direkten und automatisierten Anbindung von TS-Systemen an ein multidimensionales Array Datenbanksystem. HEAVEN realisiert ein hochgradig erweiterbares und skalierbares Datenmanagement, das durch intelligente Automatisierung der Datenauslagerung, Datenspeicherung und des Datenretrievals erreicht wird. So wurde eine hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array Datenbank Management Systeme mit nahezu unlimitiertem Speichervolumen geschaffen. Nutzern und Applikationen wird eine transparente Sicht auf die gespeicherten Daten gegeben, unabhängig vom tatsächlichen, physikalischen Speicherort. Die automatisierte und direkte Kopplung eines multidimensionalen Array-Datenbanksystems mit TS-Systemen wurde erstmals mit HEAVEN realisiert. Bei der Entwicklung von HEAVEN sind von Anfang an die Bedürfnisse der Hochleistungsrechenzentren berücksichtigt worden. So konnten Erkenntnisse aus der Praxis durch die im interdisziplinären Projekt ESTEDI gegründete User Interest Group (UIG) in das Produkt mit einfließen. Erklärtes Ziel war daher, für Hochleistungsrechenzentren, eine flexible und performante Infrastruktur für sehr große Datenvolumen zu entwickeln. Dabei sollte vor allem der bestehende Flaschenhals im Bereich des Retrieval und der Evaluierung dieser Rasterdaten (Array-Daten) beseitigt werden. Es wurde nicht nur konzeptuelle Arbeit geleistet, sondern unter konsequenter Einbeziehung der Nutzer auf die Praxistauglichkeit der entwickelten Systeme geachtet. Auch die Ergebnisse

zahlreicher Diskussionen auf renommierten, internationalen Konferenzen gingen in die Arbeit mit ein.

HEAVEN ist für performante Zugriffe auf sehr große Datenmengen beliebiger Dimensionalität ausgelegt. Durch die Verwendung einer multidimensionalen Indexstruktur (R^+-Baum) erlaubt HEAVEN eine effiziente Navigation auf sehr großen Datenmengen. Dabei besteht die Möglichkeit, selektiv auf bestimmte Teilbereiche multidimensionaler Objekte zuzugreifen, die auf TS-Medien gespeichert wurden. Das hat zur Folge, dass die Dauer der Anfrage mit der Größe der Bereichsanfrage skaliert und nicht mit der Objektgröße, wie es im HPC-Bereich bisher üblich war. Der „Nutzungsgrad", der von Wissenschaftlern angeforderten Daten, steigt dadurch um ein Vielfaches. Bisher verwendeten Wissenschaftler nur etwa 10% der angeforderten Daten [GKDT02]. Auch Datenzugriffe, die einen Schnitt durch eine Menge von Datenobjekten bedeuten (z.B. bei Zeitreihen), werden effizient bearbeitet. Das entlastet Netzwerke und ermöglicht neue Anwendungsgebiete, wie zum Beispiel die Datensimulation über Internet, da nur eine minimale Ergebnis- bzw. Datenmenge übertragen wird. Eine geeignete, nach bestimmten Anfragemustern entsprechende Modellierung, der in der Datenbank und somit auf TS-Medien gespeicherten Daten, gewährleisten ein effizientes Datenretrieval. Eine zeitaufwändige Rohdatenextraktion und eine primäre Datenverarbeitung vor einer Analyse sind nicht mehr notwendig.

Mit HEAVEN ist eine einfache Möglichkeit zur Speicherung und Archivierung von multidimensionalen Rasterdaten gegeben. Als Schlagwort dient hier auch Information Lifecycle Management. Das entwickelte System ist gleichermaßen hinsichtlich operativer Daten und historischer Daten optimiert. Aspekte der Performance spielen naturgemäß eine wesentliche Rolle im Bereich des Retrievals riesiger Datenmengen. Deshalb wurde nicht nur eine einfache Anbindung eines TS-Systems an RasDaMan (Raster Data Management) realisiert. Vielmehr wurden neben den in HEAVEN implementierten Basisfunktionen Retrieval, Update, Delete und nutzeraktiviertes Prefetching von multidimensionalen Array-Daten weit reichende Optimierungen umgesetzt. Diese Optimierungen betreffen hauptsächlich ein performantes und angepasstes Datenretrieval über alle Speicherhierarchien hinweg. Sind bei einer Anfrage Zugriffe auf TS-Medien involviert, so kann nicht auf die traditionelle Art und Weise verfahren werden, da das zu einer katastrophalen Performance der Anfragebearbeitung führt. Es müssen vielmehr neue Konzepte eingesetzt werden, um das zu verhindern. Optimierungen beim Datenretrieval von TS-Medien sind aufgrund der vorhandenen großen Zugriffslücke (ca. 10^5) in Bezug auf die Performance zwischen Sekundär- und Tertiärspeicher besonders sinnvoll. Die umgesetzten Konzepte verfolgen vor allem die Ziele:

1. Vermeidung, bzw. Reduzierung von Zugriffen auf dem TS-Medium
2. Minimierung von Medienwechseloperationen
3. Minimierung von Operationen zum Positionieren

Die aufgeführten Ziele sind entsprechend des zu erreichenden zeitlichen Optimierungspotentials sortiert. Bei der Umsetzung dieser Ziele helfen die in dieser Arbeit entwickelten Konzep-

te und Strategien. Die in Klammern angegebenen Nummern spiegeln die zu erreichenden Ziele wider:

- Super-Kachel-Konzept (Ziel 1 und 3)
- Intra- und Inter-Super-Tile-Clustering (Ziel 2 und 3)
- Query-Scheduling (Ziel 2 und 3)
- Aufbau einer Caching-Hierarchie (Ziel 1)
- Erweiterte Analysemöglichkeit Object-Framing (Ziel 1)
- Verwendung vorberechneter Operationsergebnisse (Ziel 1)

Im Folgenden werden die Besonderheiten der einzelnen Optimierungsstrategien aufgeführt.

Super-Kachel-Konzept
Das entwickelte Super-Kachel-Konzept, ermöglicht durch eine dem jeweiligen Speicherort angepasste Datengranularität und dem realisierten Intra- und Inter-Super-Tile-Clustering, eine Reduzierung der TS-Zugriffe und eine Minimierung kostspieliger Positionierungsoperationen. Die speziell für das Datenretrieval von TS-Medien eingeführte Datengranularität heißt Super-Kachel. Es wird dabei die große Anzahl von Zugriffen auf kleine Objekte (Kacheln) ersetzt durch eine geringe Anzahl von Zugriffen auf größenoptimierte Objekte (Super-Kachel). Dadurch wird die Dominanz der Positionierungszeit gegenüber der Übertragungszeit abgeschwächt. Entfallen Zugriffe auf kleine Datenobjekte, reduzieren sich selbstverständlich auch die kostenintensiven Operationen zur Positionierung dieser Objekte. Die optimale Größe eines angepassten Datenvolumens einer Super-Kachel wird, abhängig von medien- und objektspezifischen Eigenschaften, automatisch ohne Nutzerinteraktion ermittelt.

Intra- und Inter-Super-Tile-Clustering
Die Art und Weise, wie das Exportieren multidimensionaler Daten auf TS-Medien vollzogen wird, hat starke Auswirkungen auf ein künftiges performantes Datenretrieval. Durch eine geschickt gewählte Exportreihenfolge der Datenobjekte und eine durchdachte Speicherung der Daten auf den vorhandenen TS-Medien, werden beim Datenretrieval Medienwechsel und Positionierungsoperationen minimiert. Entscheidend hierfür ist das multidimensionale Clustering. Es sorgt für die Aufrechterhaltung der räumlichen Nachbarschaft von Zellen innerhalb des multidimensionalen Datenraumes. So kann vermieden werden, dass einzelne Kacheln auf Magnetband „zufällig“ verteilt vorliegen. Um die Verluste bei der Linearisierung einer mehrdimensionalen Repräsentation der MDD auf die eindimensionale Abbildung auf einem TS-Medium möglichst gering zu halten, erfolgt bei HEAVEN der Exportvorgang in zwei Schritten: Zunächst werden durch das eSTAR-Verfahren einzelne Kacheln zu Super-Kacheln kombiniert, um eine größere Zugriffsgranularität zu erhalten. Bei diesem Übergang bleibt durch das realisierte Intra-Super-Tile-Clustering die räumliche Nachbarschaft und die bei der Datenmodellierung gewünschten Vorzugsdimensionen erhalten. Um ein effizientes Datenretrieval zu erreichen, ist nicht nur die Kombination von räumlich benachbarten Kacheln zu Super-Kacheln (Intra-Super-Tile-Clustering) wichtig, sondern auch das Erhalten der räumlichen Nachbarschaft zwischen den Super-Kacheln auf dem TS-Medien (Inter-Super-Tile-Clustering). Aus diesem Grund erfolgt in einem weiteren Schritt die Bestimmung der Lineari-

sierungsordnung, die beim Schreiben der Super-Kacheln auf TS-Medium verwendet wird. Bei HEAVEN wird zur Ermittlung dieser Ordnung der R^+-Baum herangezogen. Das hat den Vorteil, dass weder beim Exportieren, noch beim späteren Retrieval, zusätzliche Berechnungen notwendig sind, um eine spezielle Linearisierungsordnung zu erhalten.

Query-Scheduling
Sind bei einer Anfrage Zugriffe auf TS-Medien involviert, kann nicht entsprechend der traditionellen Art und Weise von RasDaMan verfahren werden. Das führt aufgrund zufälliger Zugriffe auf TS-Medien zu einer sehr schlechten Performance durch die Vielzahl der benötigen Positionierungsoperationen. Teilweise sind weiterhin zeitaufwändige Medienwechsel notwendig (mehrere Minuten). Um eine Performancesteigerung bei TS-Zugriffen zu erreichen, wurden Maßnahmen ergriffen, um unnötige Positionierungs- und Medienwechseloperationen einzusparen. Es wurden Techniken, wie das Intra-Query-Scheduling, das Inter-Query-Scheduling und das Query-Interleaving umgesetzt. Dabei werden durch eine geschickte Vorsortierung der Anfragewarteschlange zufällige Ladevorgänge von TS-Medien reduziert und möglichst durch kombinierte Operationen ersetzt. Das realisierte Intra-Query-Scheduling passt die Anforderungsreihenfolge innerhalb einer Anfrage für die auf TS-Medien gespeicherten Datenobjekte (Super-Kacheln) entsprechend der beim Exportieren gewählten Linearisierungsordnung an. Das Inter-Query-Scheduling wird bei HEAVEN nach dem FCFS-Prinzip, im Zusammenspiel mit der Verwendung paralleler RasDaMan-Server durchgeführt. Dadurch ist ein Mehrbenutzerbetrieb bei HEAVEN kein primäres Problem, was auch die Erfahrungen aus dem Projekt ESTEDI beweisen. Bei dem in dieser Arbeit entwickeltem HSM-System TCT (Tape-Controller-Toolkit) wurde zur Realisierung des Query-Interleaving das SCAN-Verfahren umgesetzt. Alle bei HEAVEN umgesetzten Konzepte im Bereich des Query-Scheduling bewirken eine enorme Steigerung der Performance bei der Anfragebearbeitung.

Caching-Hierarchie
Das Zwischenspeichern von Datenobjekten auf einer schnelleren Speicherebene, hat vor allem das Ziel, zeitaufwändige Zugriffe auf TS-Medien zu vermeiden. Aus diesem Grund wurde in HEAVEN eine mehrschichtige Caching-Hierarchie, bestehend aus dem Primärspeicher P, dem Sekundärspeicher-Cache-Bereich SRC und dem HSM-Cache-Bereich SHC realisiert. Durch die mehrstufige Caching-Hierarchie, wird die Wahrscheinlichkeit gesteigert, dass sich angefragte Datenelemente in einem dieser Cache-Bereiche aufhalten. Weiterhin wurde bei HEAVEN mit dem H-LRU-Verfahren eine speziell für angebundene TS-Systeme optimierte Verdrängungsstrategie entwickelt. Hier steht vor allem die Minimierung der mittleren Antwortzeit für Zugriffe auf Datenobjekte, die sich auf TS-Medien befinden, an erster Stelle. Es wird beim Caching eine entstehende Antwortzeiterhöhung durch das Verdrängen von Datenobjekten zusätzlich berücksichtig. Objekte, deren Fehlen zu hohen Antwortzeiten führt, werden eher im Cache gehalten, als Objekte, deren Fehlen zu geringeren Antwortzeiten führt. Weiterhin werden keine Datenobjekte aus dem Cache-Bereich SRC entfernt, die von einer aktuell ausgeführten Anfrage eines der parallel arbeitenden RasDaMan-Server benötigt werden. Außerdem wird vermieden, Datenobjekte aus dem Cache-Bereich zu verdrängen, die durch einen nutzeraktivierten Prefetching-Auftrag vorgeladen wurden. Gerade die Kombina-

tion dieser Entscheidungskriterien bringt Vorteile in Bezug auf die Minimierung der Zugriffszeit, gegenüber herkömmlichen Verdrängungsstrategien.

Object-Framing
Die Erweiterung der multidimensionalen Anfragesprache um neue Anfragekonstrukte, wie zum Beispiel das Object-Framing, ermöglicht neben neuen Analysemöglichkeiten auch die Reduzierung des zu ladenden Datenvolumens. Durch Object-Framing wird ein neuer Weg in der Datenanalyse beschritten. Eine Vielzahl von Analysemöglichkeiten waren bisher bei RasDaMan nicht oder nur sehr umständlich möglich. In der einschlägigen Literatur ist kein vergleichbares Verfahren für die Datenanalyse bekannt. Neben dem Mehrwert in der Datenanalyse ist Object-Framing ein ressourcenschonendes Verfahren, da das zu ladende Datenvolumen für einen spezifizierten Anfragebereich minimiert wird. Es wird eine performante Anfragebearbeitung ermöglicht. Speziell aus der Sicht von HEAVEN kann durch das verminderte Datenvolumen, das für eine Anfrage zu laden ist, die Anzahl der zeitintensiven TS-Zugriffe reduziert werden. Zeitintensive Medienwechsel und Positionierungsoperationen können eingespart werden. Das führt zu einer Steigerung der Performance.

Verwendung vorberechneter Operationsergebnisse
Die Speicherung der vorberechneten Ergebnisse von Operationen, bietet ein sehr großes Potential, die Performance der Ausführung von Anfragen zu steigern. Das liegt daran, dass sehr häufig bei der Auswahl von Objekten (Selektionsbedingung), auf vorberechnete Ergebnisse zurückgegriffen werden kann. Auch das Endergebnis einer Anfrage beruht in vielen Fällen auf einer Aggregatoperation, wie zum Beispiel der Bestimmung der Maximalwindgeschwindigkeit einer Klimasimulation. Die Verwendung von vorberechneten Ergebnissen bringt Vorteile in zweierlei Hinsicht: Einerseits können Rechenkapazitäten eingespart werden. Nach erstmaliger Berechung der Ergebnisse von Aggregatoperatoren, werden die Ergebnisse kachelbasiert im Systemkatalog abgelegt. Bei der Wiederverwendung dieser Ergebnisse, entfällt die erneute zeitintensive Berechnung. Andererseits wird der E/A-Aufwand enorm reduziert. Anstelle Kacheln aus der Datenbank, bzw. von TS-Medien zu laden, wird auf die vorberechneten Ergebnisse aus dem Systemkatalog zurückgegriffen. Das wiederum bedeutet, dass kostspielige TS-Zugriffe minimiert werden.

HEAVEN ist mehr als nur ein „Proof of Concept“. Es ist eine voll funktionsfähige, hierarchische Speicher- und Archivierungsumgebung für multidimensionale Array-Daten. Weiterhin ist eine flexible und einfache Anbindung an bestehende, konventionelle HSM-Systeme möglich. Durch das Zurückgreifen auf konventionelle HSM-Systeme wurde eine Kompatibilität zu unterschiedlichen Bandsystemtechnologien erreicht. Ist eine entsprechende Schreib-/Lesestation in dem angebundenen HSM-System vorhanden, wird diese TS-Technologie von HEAVEN unterstützt. Das garantiert eine Offenheit gegenüber Neuentwicklungen im Bereich der Tertiärspeichertechnologien. Bei der Realisierung von HEAVEN wurde der Balanceakt zwischen performantem Zugriffsverhalten und günstigem Speicherplatz (TS-Medien) erfolgreich umgesetzt. Eine Vielzahl an Veröffentlichungen zu der hier vorgestellten Thematik unterstreichen die Relevanz dieser Arbeit für die Wissenschaft und die Praxis.

Problems can become opportunities when the right people come together.
Robert South (1634–1716)

Notationsverzeichnis

Symbol	**Beispiel**	**Beschreibung**
T	{float; {char; char}}	Basisdatentyp. Menge aller einzelnen Werte, die diesem Typ entsprechen.
ζ	integer, real, boolean	Menge gegebener skalarer (atomarer) Datentypen.
size(T)	10 Byte	Speichervolumen des Basisdatentyps in Byte.
$\nu \in T$	{26,7; {124; 39}}	Konkreter Wert (value) des Basisdatentyps T.
α, β	([100,0,0], {26,7; 31}), ([100,0,1], {23,1; 22}), ([100,0,2], {20,9; 19})	MDD mit Abbildung f: D → T. Jedem Punkt der Domäne D wird Wert mit Basisdatentyp T zugewiesen.
τ	([100,0,0], {26,7; 31}), ([100,0,1], {23,1; 22})	Kachel τ eines MDD α mit der Abbildung f: D → T
σ	([100,0,0], {26,7; 31}), ([100,0,1], {23,1; 22})	Super-Kachel σ eines MDD α mit der Abbildung f: D → T.
ϕ		Super-Kachel-Knoten im R^+-Baum.
δ	0,7	Datendichte eines MDD α. Anteil des mit Werte besetzten Datenraumes gegenüber komplettem Datenraum.
$\underline{l}, \underline{h}, \underline{x} \in \mathbb{Z}^d$	$[3, 200, 320] \in \mathbb{Z}^3$	Vektor, Punkt mit Position im d-dimensionalen Raum.

Symbol	Beispiel	Beschreibung
D	D = [0:299, 0:599]	Multidimensionales Intervall eines Objektes.
d = *dim*(α)	*dim*(α) = 3	Dimensionalität eines Objektes.
C	C ⊂{α|*type*(α) = <D,T>}	Kollektion aus mehreren MDD mit gleicher Domäne D und Basisdatentyp T.
sdom(α) = D	*sdom*(α) = [0:299, 0:599]	Multidimensionales Intervall bzw. Domäne (engl. spatial domain) eines MDD.
type(α) = <D, T>	<[5:9, 0:9], {124; 39}>	Typ M eines MDD α.
low(D), bzw. *low*(*sdom*(α))	[0,0]	Untere Grenzpunkte eines MDD α.
high(D), bzw. *high*(*sdom*(α))	[299,599]	Oberer Grenzpunkte eines MDD α.
extent(D), bzw. *extent*(*sdom*(α))	[300,600]	Ausdehnung eines MDD α mit *extent*(D) = *high*(D) – *low*(D) + 1
intersect(q, α)	$I_\tau = \{\tau_1, \ldots, \tau_n\}$ oder $I_\sigma = \{\sigma_1, \ldots, \sigma_n\}$	Ermittelt Menge I_τ oder I_σ der von einer Anfrage q betroffenen Kacheln oder Super-Kacheln.
h_{SN}	4	Höhe des Unterbaumes eines Super-Knotens im R^+-Baum.
M	80	Maximale Anzahl an Einträgen pro Indexknoten im R^+-Baum.
m	60	Tatsächliche Anzahl an Einträgen pro Indexknoten im R^+-Baum ($m \leq M$).
q	RasQL	Anfrage auf multidimensionale Daten.
Q	$Q = \{q_1, \ldots, q_n\}$	Menge von Anfragen.
I_τ, I_σ	$I_\tau = \{\tau_1, \ldots, \tau_n\}$	Menge der von einer Anfrage q betroffenen Kacheln τ oder Super-Kacheln σ.
$N_{I\tau}$, $N_{I\sigma}$	1644	Anzahl der von einer Anfrage q betroffenen Kacheln ($N_{I\tau} = \|I_\tau\|$), bzw. Super-Kacheln ($N_{I\sigma} = \|I_\sigma\|$)..

Symbol	Beispiel	Beschreibung
υ	0,36	Verschnitt bei Anfragen q_i Prozentualer Anteil des überschüssig gelesenen Datenraumes bei kachelbasierter Speicherung.
Δ	262144 Byte	Daten-, bzw. Speichervolumen einer Kachel oder einer Super-Kachel in Byte.
Ψ	0,8	Prozentualer Anteil der Positionierungszeit t_p gegenüber der Gesamtzeit t_g aus Transfer- und Positionierungszeit.
Γ	40 MByte/s	Transferrate, bzw. Datenübertragungsrate
$\Theta_{\alpha,\sigma,\tau}$	$\tau \in \Theta_\tau$	Objektmenge aller MDD α, Super-Kacheln σ oder Kacheln τ.
$\Theta_{P,S,SRC,SHC,Ton,Tnear,Toff}$	$\tau \in \Theta_{Tnear}$	Speicherinhalte der unterschiedlichen Speicherhierarchien.
$\Theta_{SRCvictims}$	$\sigma \in \Theta_{SRCvictims}$	Menge aller durch einen Verdrängungsalgorithmus ermittelten Kandidaten.
N_Θ	7654398	Anzahl der Objekte aus $\Theta_{\alpha,\sigma,\tau}$, bzw. $\Theta_{P,S,SRC,SHC,Ton,Tnear,Toff}$
$v \in V$	$v(\tau)$	Tertiäres Speichermedium aus der Menge V aller von HEAVEN verwalteten Medien.
V	$V = \{v_1, \ldots, v_n\}$	Menge aller von HEAVEN verwalteten Speichermedien v.
s	$s \in S$	Schreib-/Lesestation eines TS-Systems.
S		Menge aller Schreib-/Lesestationen des an HEAVEN angebundenen TS-Systems.
r	$r \in R$	Roboterarm eines TS-Systems.
R	$R = \{r_1, \ldots, r_n\}$	Menge aller Roboterarme des an HEAVEN angebundenen TS-Systems.
locate(object)	*locate*($\tau \in \Theta_{Tnear}$)	Liefert Speicherort eines Objektes
move(object, source, target)	*move*(τ, Θ_{Ton}, Θ_{SHC})	Verschiebt Objekt von einer Speicherebene zu einer anderen.

Symbol	Beispiel	Beschreibung
replicate(object, source, target)	*replicate*(τ, Θ_{SHC}, Θ_{SRC})	Repliziert Objekt einer Speicherebene auf eine andere.
access(object)	*access*($\tau \in \Theta_{Ton}$)	Realisiert Zugriff auf ein Objekt.
buildST(object)	*buildST*(α)	Kombiniert räumlich benachbarte Kacheln eines Objektes zu Super-Kacheln.
export(object)	*export*(α)	Realisiert das Exportieren eines Objektes.
delete(object, source)	*delete*(τ, Θ_{SHC})	Löscht ein Objekt von einer Speicherebene.
update(object, source)	*update*(τ, Θ_{Ton})	Aktualisiert ein Objekt einer Speicherebene nach einer Änderung.
reimport(object, source)	*reimport*(α, Θ_{Ton})	Re-Importiert ein Objekt von einer HSM-Speicherebene in den Sekundärspeicher von RasDaMan (SR).
prefetch(object, source)	*prefetch*(τ, Θ_{Ton})	Realisiert das Vorladen eines auf TS-Medium gespeicherten Objektes in den RasDaMan-Cache-Bereich (SRC).
P		Primärspeicher oder Hauptspeicher
SR		Sekundärspeicher des von RasDaMan verwendeten konventionellen DBMS.
SRC		Sekundärspeicher des TS-Cache-Bereiches, des von RasDaMan verwendeten konventionellen DBMS.
SHC		Sekundärspeichercache eines HSM-Systems. Wird auch als Onlinespeicherbereich des HSM-Systems bezeichnet.
T_{on}		Online Speicherebene eines HSM-Systems. TS-Medium, das sich in einer Schreib-/Lesestation befindet.
T_{near}		Nearline Speicherebene eines HSM-Systems. TS-Medien, die automatisch durch Roboter zugreifbar sind.
T_{off}		Offline Speicherebene eines HSM-Systems. TS-Medien, die nur durch Nutzerinteraktion zugreifbar sind.

Symbol	Beispiel	Beschreibung
H	0,8	Cache-Trefferrate (Cache-Hit-Ratio)
o_{gv}, bzw. O_{gv}	$o_{gv} \in O_{gv}$	Operation(en) *Get-Volume*: Hole TS-Medium von Medienbibliothek.
o_{lv}, bzw. O_{lv}	$o_{lv} \in O_{lv}$	Operation(en) *Load-Volume*: Lade TS-Medium in eine Schreib-/Lesestation.
o_{pv}, bzw. O_{pv}	$o_{pv} \in O_{pv}$	Operation(en) *Position-Volume*: Positioniere Schreib-/Lesekopf auf Dateianfang.
o_{rv}, bzw. O_{rv}	$o_{rv} \in O_{rv}$	Operation(en) *Rewind-Volume*: Rückspulen eines TS-Mediums.
o_{ev}, bzw. O_{ev}	$o_{ev} \in O_{ev}$	Operation(en) *Eject-Volume*: Auswerfen eines TS-Mediums.
o_{dv}, bzw. O_{dv}	$o_{dv} \in O_{dv}$	Operation(en) *Deposit-Volume*: Befördere TS-Medium von Schreib-/Lesestation zur Medienbibliothek.
o_{sv}, bzw. O_{sv}	$O_q = \{O_{gv}, O_{lv}, O_{rv}, O_{ev}, O_{dv}\}$	Operation(en) *Swap-Volume*: Operationen zum Medienwechsel.
O_q	$O_q = \{O_{sv}, O_{pv}\}$	Die Menge aller für eine Anfrage q erforderlichen TS-Operationen.

Computers are useless. They can only give you answers.
Pablo Picasso (1881 - 1973)

Glossar

ACID	Atomicity, Consistency, Isolation, Durability: Das ACID-Paradigma beschreibt wichtige Eigenschaften von Transaktionen.
AIT	Advanced Intelligent Tape: Magnetband in Kassettenbauform mit Schrägspuraufzeichnung für den professionellen Einsatz.
AIX	Advanced Interactive eXecution: Unix Derivat von IBM Corporation.
ALDC	Adaptive Lossless Data Compression: Standardisierter Komprimierungsalgorithmus, der von LTO-Bandlaufwerken verwendet wird. Basiert auf Lempel-Ziv1.
AOD	Advanced Optical Disc: Optisches Speichermedium der nächsten Generation, mit einer Speicherkapazität von 20 GByte pro Speicherschicht, gegenüber 4,7 GByte einer DVD. Direkte Konkurrenz zur Blu-Ray-Disk.
API	Application Programming Interface.
APRIL	A Parallel Run-Time Library: Laufzeitumgebung zur Speicherung von multidimensionalen Daten auf Tertiärspeicher.
AVHRR	Advanced Very High Resolution Radiation.
BFILE	Binary File: Dateityp von Oracle, um LOBs außerhalb des DBMS direkt im Dateisystem zu speichern.
BLOB	Binary Large Object: Ein Container, um große beliebig strukturierte Binärdaten, wie z.B. Bilder, in einer Datenbank zu speichern.
Blu-Ray-Disk	Optisches Speichermedium der nächsten Generation, mit einer Speicherkapazität von 27 GByte pro Speicherschicht, gegenüber 4,7 GByte einer DVD. Direkte Konkurrenz zur AOD.
BOT	Begin of Tape: Marke auf einem Magnetband, die den Beginn kennzeichnet.
CD	Compact Disk: Optisches Speichermedium mit starker Verbreitung.

CERA	Climate and Environmental Data Retrieval and Archiving System: Entwickelt vom Deutschen Klimarechenzentrum.
CERN	Conseil Europeen pour la Recherche Nucleaire: Europäisches Labor für Nuklear-Forschung mit Sitz in der Schweiz.
Chunking	Unterteilung eines multidimensionalen Objektes in mehrere, disjunkte und in jeder Dimension gleichförmige Teilobjekte. Tiling ist eine Verallgemeinerung von Chunking.
Clustering	Ausnutzung logischer, räumlicher Nähe bei der physikalischen Speicherung von Daten. Überführung einer logischen Ordnung in eine physikalische Ordnung.
CPU	Central Processing Unit: Die zentrale Verarbeitungseinheit eines Rechnersystems.
CPU-bound	Anfrage mit einem überwiegenden Anteil an Berechnungsoperationen, gegenüber den Ein-/Ausgabeoperationen.
DASD	Direct Access Storage Device: Online-Speichergeräte, die direkt zugreifbar sind, wie zum Beispiel eine Festplatte.
DBMS	Datenbank Management System.
DDS	Digital Data Storage: Magnetband in Kassettenbauform mit Schrägspuraufzeichnung für den semiprofessionellen Einsatz.
Demand-Prefechting	Bedarfsgetriebenes Vorladen von Datenobjekten, die im Gegensatz zum Prefetching Vorfeld bekannt sind.
DFD	Deutsches Fernerkundungsdatenzentrum des DLR. Deutschlands größtes Satellitenbildarchiv.
DLT	Digital Linear Tape: Magnetband in Cartridgebauform für den professionellen Einsatz.
DLZ	Digital Lempel Ziv: Komprimieralgorithmus, eingesetzt bei DLT-Bandlaufsystemen.
DRAM	Dynamic Random Access Memory: Arbeitsspeicher eines Rechners, dessen Inhalt in regelmäßigen Abständen aufgefrischt werden muss.
DVD	Digital Versatile Disk: Optisches Speichermedium mit zunehmender Verbreitung.
E/A	Eingang/Ausgang (I/O, Input/Output): Datentransfer zwischen der CPU und einem Peripheriegerät. Jeder Transfer stellt einen Ausgang eines Gerätes und einen Eingang in ein anders Gerät dar.
E-IDE	Enhanced Integrated Device Electronics: Standardschnittstelle für Massenspeicher wie zum Beispiel Festplatten und CD-ROM-Laufwerke.
EOT	End of Tape: Marke auf einem Magnetband, die das Ende kennzeichnet.

EOWEB	Earth Observation WEB: Internet Erdbeobachtungs-Informationsservice des Deutschen Luft- und Raumfahrtzentrums (http://eoweb.dlr.de).
ESQL	Embedded Structured Query Language.
eSTAR	Extended STAR: Im Rahmen dieser Dissertation erweiterter Algorithmus zur effizienten Speicherung von MDD auf TS-Medien.
ESTEDI	European Spatio-Temporal Data Infrastructure for High-Performance Computing: Europäisches Projekt das unter IST-1999-11009 gefördert wurde. Laufzeit: 1. Februar 2000 bis 30. April 2003.
Ethernet	Rahmenbasierte Computer-Vernetzungstechnologie für LAN. Sie definiert Kabeltypen, Signalisierung für die Bitübertragungsschicht, Paketformate und Protokolle für die Medienzugriffskontrolle des OSI Modells.
FCFS	First-Come-First-Served: Verfahren, das bei der Prozessverwaltung (Scheduling) eingesetzt wird. Die Ausführungsreihenfolge entspricht der Reihenfolge des Eintreffens.
FIFO	First-In-First-Out: Verdrängungsstrategie für Arbeitsspeicher- oder Prozessverwaltung.
FORWISS	Bayrisches Forschungszentrum für Wissensbasierte Systeme.
FTP	File Transfer Protocol: Ein Protokoll, das erlaubt, Daten von einem Rechner via TCP/IP auf einen anderen Rechner zu übertragen.
GIS	Geo Information Systems.
GPFS	General Parallel File System: Ein Hochleistungsdateisystem von IBM.
HDD	Harddisk (Festplatte).
HEAVEN	Hierarchical Storage and Archive Environment for multidimensional Array Database Management Systems.
Helical Scan	Schrägspuraufzeichnung. Aufzeichnungsverfahren bei Magnetbandspeicher, wie zum Beispiel bei DDS-Magnetband.
H-LRU	*HEAVEN*-LRU: Erweiterter LRU Verdrängungsalgorithmus mit speziell für TS-Zugriffe optimierten Verbesserungen.
HPC	High Performance Computing.
HPSS	High Performance Storage System: HSM-Software für High-Performance-Netz. Verfügbar auf AIX, Solaris, SGI, IRIX.
HRRN	Highest-Response-Ratio-Next: Verfahren, das bei der Prozessverwaltung (Scheduling) eingesetzt wird. Kompromiss zwischen FCFS und SJF. Die Ausführungsreihenfolge wird durch den Response-Ratio-Wert berechnet.
HSM	Hierarchisches Speicher Management: Verwaltungseinheit zum automatischen Auslagern und Zurückholen von Dateien zu/von Tertiärspeichersystemen.

HTTP	Hypertext Transfer Protocol: Standardisiertes Protokoll für die Übertragung von Hypertext.
HVD	Holographic Versatile Disc: Holographische Speicherscheibe mit zunächst 200 GByte Speicherkapazität in DVD-Größe.
I/O	Input/Output: Siehe E/A.
I/O-bound	Anfrage mit einem überwiegenden Anteil an Ein-/Ausgabeoperationen, gegenüber der eigentlichen Anfrageberechnung (CPU).
IDE	Integrated Device Electronics: siehe E-IDE.
IEEE	Institute of Electronical and Electronics Engineers.
Index-sequentieller Zugriff	Auf größere Speicherblöcke kann wahlfrei zugegriffen werden. Auf einzelne Speicherzellen innerhalb eines Speicherblockes kann nur sequentiell zugegriffen werden.
InfiniBand	InfiniBand: Stellt eine E/A-Architektur zur Verfügung, mit der Übertragungsraten von 500 MByte/s bis zu 6 GByte/s erreicht werden.
Inode	Index Node. In der Dateiverwaltung eines Unix Betriebssystems beschreibt ein Indexknoten genau eine Datei. Neben den Informationen, um alle Blöcke dieser Datei zu adressieren, sind Informationen über Eigentümer, Zugriffsrechte, Typ und Größe der Datei enthalten.
Inter-Query-Scheduling	Ablaufplanung mehrerer Anfragen, gewährleistet eine optimierte Ausführungsreihenfolge, durch Sortierung der Anfragewarteschlange.
Inter-Super-Tile Clustering	Aufrechterhaltung der räumlichen Nachbarschaften zwischen Super-Kacheln, bei der Speicherung auf tertiären Speichermedien.
Intra-Query-Scheduling	Ablaufplanung innerhalb einer Anfrage, gewährleistet eine optimierte Ausführungsreihenfolge, durch Sortierung der Anfragewarteschlange.
Intra-Super-Tile Clustering	Aufrechterhaltung der räumlichen Nachbarschaften zwischen Kacheln, bei der Verschmelzung von Kacheln zu Super-Kacheln.
IPCC	Intergovernmental Panel on Climate Change: 1988 von der World Meteorological Organization (WMO) und dem United Nations Environment Programme (UNEP) zu Abschätzung der Klimaänderung gegründet.
LAN	Local Area Network. Lokales Netzwerk mit eingeschränkter Reichweite.
Lazy Eject	Träges Auswerfen: TS-Medien verbleiben zunächst in der Schreib-/Lesestation, da von einer erhöhten Wahrscheinlichkeit ausgegangen wird, dass in naher Zukunft weitere Aufträge folgen.
LFU	Least-Frequently Used: Verdrängungsstrategie für die Arbeitsspeicherverwaltung.
LOB	Large Object: Ein Container, um große Daten, wie z.B. Bilder, in einer Datenbank zu speichern.

LRU	Least-Recently Used: Verdrängungsstrategie für die Arbeitsspeicherverwaltung.
LRZ	Leibniz-Rechenzentrum: Rechenzentrum für Münchner Hochschulen.
LTO	Linear Tape Open: Magnetband in Cartridgebauform für den professionellen Einsatz.
Main Memory	Hauptspeicher bzw. Arbeitsspeicher eines Computersystems. Auch Primärspeicher genannt.
Mammoth	Magnetband in Kassettenbauform mit Schrägspuraufzeichnung für den professionellen Einsatz.
MDM	Meta-data Management System: Ein System zur Metadatenverwaltung in DBMS für sehr große multidimensionale Daten.
MPI	Message Passing Interface.
MPI-Met	Max-Planck-Institut für Meteorologie.
MQL	Multidimensional Query Language: Anfragesprache für multidimensionale DMBS, allerdings nicht wie SQL standardisiert. Eine Ausprägung einer MQL ist die für RasDaMan entwickelte Anfragesprache RasQL.
MSSRM	IEEE Mass Storage System Reference Model: Die IEEE Storage System Standards Working Group entwickelt dieses Referenzmodell für offene Speichersysteme.
Native Capacity	Native Grundkapazität von Magnetbändern. Entspricht dem Speichervolumen eines Magnetbandes ohne Ausnutzung von Komprimierung.
NFS	Network File System: Ein Protokoll, entwickelt von Sun Microsystems. Erlaubt Rechnern, auf Dateien über das Netzwerk zuzugreifen, so als wären sie lokal gespeichert. NFS ist ein de facto Standard.
NOAA	National Oceanic and Atmonsperic Administration.
Object Framing	Rahmenbasiertes Zuschneiden bei Bereichsanfragen. Anfragen sind nicht mehr länger in der Form auf multidimensionale Hyperquader eingeschränkt. Komplexere Anfragebereiche sind möglich.
ODBC	Open Database Connectivity: Standard, um auf unterschiedliche DBMS zuzugreifen.
OID	Object Identifier: Eindeutiger Bezeichner für Objekte.
OLTP	Online Transaction Processing: Transaktionssystem, das eine hohe Anzahl von Transaktionen pro Sekunde verarbeitet.
Prefetching	Vorladen von Datenobjekten oder Seiten, die in naher Zukunft mit hoher Wahrscheinlichkeit referenziert werden.
QFS	Quick File System: Ein Unix Dateisystem von Sun Microsystems.

Query Interleaving	Anfrageverschränkung: Anfragen werden nicht nacheinander sondern verschränkt ausgeführt um die Retrievalperformance zu steigern.
Query Scheduling	Ablaufplanung von Anfragen, gewährleistet eine optimierte Ausführungsreihenfolge, durch Sortierung der Anfragewarteschlange.
RAID	Redundant Arrays of Independent Disks: Standard zur Beschreibung redundanter Speicherung von Daten im Massenspeicherbereich (Festplatten).
RAIT	Redundant Arrays of Independent Tapes: Beschreibung der redundanten Speicherung von Daten im Tertiärspeicherbereich. Angelehnt an RAID.
RasDaMan	Raster Data Management: Mutidimensionales DBMS für Array-Daten.
RasDa-Visualizer	Programm zur Visualisierung von Speicherstrukturen, eines in RasDaMan gespeicherten MDD, in der Granularität von Kacheln und Super-Kacheln.
RasDL	RasDaMan Data Definition Language: Datendefinitionssprache des Array-DBMS RasDaMan.
RasLib	RasDaMan Library: C++ bzw. JAVA Klassenbibliothek des Array-DBMS RasDaMan.
RasML	RasDaMan Data Manipulation Language: Datenmanipulationssprache des Array-DBMS RasDaMan.
RasQL	RasDaMan Query Language: Multidimensionale Anfragesprache des Array-DBMS RasDaMan.
RDBMS	Relationales Datenbank Management System.
RDRAM	Rambus DRAM: Arbeitsspeicher eines Rechners, dessen Inhalt in regelmäßigen Abständen aufgefrischt werden muss und synchron mit dem Speicherbus arbeitet.
RGB	Rot-Grün-Blau: Additives Farbmodell. Eine Farbe wird durch drei Werte beschrieben: den Rot-, den Grün- und den Blauanteil. Jeder Farbanteil kann zwischen 0 % und 100 % variieren.
RPC	Remote Procedure Call: Eine Programmierschnittstelle, das einem Programm ermöglicht, Services eines Programms einer Remote-Maschine zu nutzen.
R-Tree	Rectangle Tree: Indexierung von räumlich ausgedehnten Objekten. Abwandlungen davon sind der R^*-Tree und der R^+-Tree.
SAM	Storage Archive Manager: Ein von Sun Microsystems angebotenes HSM-System.
SAM-FS	Storage Archive Manager File System: Ein Unix Dateisystem, für das HSM-System SAM HSM-Systems von Sun Microsystems.

SAN	Storage Area Networks: Eigenes Speichernetzwerk, das den Transfer von Massendaten aus dem LAN in das SAN verlagert.
Scheduling	Prozessverwaltung: Es bezeichnet die faire Verwaltung von mehreren Prozessen, die auf einem Computer ausgeführt werden. Planung eines zeitlichen Ablaufes.
SCSI	Small Computer System Interface: Ein Standard für Peripheriebussysteme z.B. zum Anbinden von Festplatten.
S-DLT	Super Digital Linear Tape: Magnetband in Cartridgebauform für den professionellen Einsatz.
SDM	Scientific Data Manager: Ein System zur Metadatenverwaltung in DBMS für sehr große multidimensionale Daten.
SDRAM	Synchronous DRAM: Arbeitsspeicher eines Rechners, dessen Inhalt in regelmäßigen Abständen aufgefrischt werden muss und synchron mit dem Speicherbus arbeiten.
Sequentieller Zugriff	Auf die Speicherzelle kann nur in einer bestimmten Reihenfolge zugegriffen werden.
SFC	Space Filling Curves. Klassische Vertreter raumfüllender Kurven sind die Z-Kurve und die Hilbert-Kurve.
SJF	Shortes-Job-First: Verfahren, das bei der Prozessverwaltung (Scheduling) eingesetzt wird. Der Prozess mit der zu erwartenden kürzesten Laufzeit wird zu erst ausgeführt.
Speed-Up	Beschreibt den Performanzgewinn bezüglich der Ausführungszeit eines Systems mit optimierten Komponenten, gegenüber einem nicht optimierten System.
SQL	Structured Query Language: Anfragesprache für Datenbanksysteme.
SRAM	Static Random Access Memory: Arbeitsspeicher eines Rechners, dessen Inhalt ohne wiederholte Auffrischung erhalten bleibt.
STAR	Super-Tile Algorithm: Im Rahmen dieser Dissertation entwickelter Algorithmus zur effizienten Speicherung von MDD auf TS-Medien.
StorHouse/RM	StorHouse Relational Manager: Realisiert die Anbindung kommerzieller relationaler DBMS an das StorHouse HSM-System der Firma FileTek.
StorHouse/SM	StorHouse Storage Manager: Realisiert das Speichermanagement des StorHouse HSM-Systems der Firma FileTek.
Streaming	Unter Stromverarbeitung versteht man die kontinuierliche Übertragung von Daten. Ziel dabei ist es, die Daten möglichst ohne Zeitverlust übertragen und bearbeiten zu können. Die Alternative ist der Download, bei dem vor der Bearbeitung die komplette Datei übermittelt werden muss.

Striping	Beschreibt die Technik, Daten auf mehrere Medien (Festplatte oder Magnetband) zu verteilen. Dadurch können diese Daten parallel geladen werden.
TCP/IP	Transmission Control Protocol over Internet Protocol: Standardprotokolle für das Internet. TCP behandelt die Transportschicht und IP die Netzwerkschicht, entsprechend des OSI-Schichtenmodells.
TCT	Tape Controller Toolkit: Im Rahmen von HEAVEN entwickeltes Werkzeug, um Magnetbandsysteme direkt mit RasDaMan zu koppeln.
Tiling	Unterteilung (Kachelung) eines multidimensionalen Objektes in mehrere disjunkte Sub-Objekte. Verallgemeinerung von Chunking.
TS	Tertiärspeicher: Massenspeicher, wie zum Beispiel automatisierte Bandlaufwerke.
TSM	Tivoli Space Manager: Ein HSM-System von IBM erhältlich für Solaris und AIX.
UB-Tree	Universal B-Tree. Erweiterung des B-Baumes für multidimensionale Indexierung.
UDF	User Defined Functions: Benutzerdefinierte Funktionen für DBMS.
UFS	Unix File System.
VFS	Virtual File System: Virtuelle Dateisystemschicht bei Unix Betriebssystemen zur einheitlichen Darstellung der Dateisysteme.
VSAC	Volume Storage Allocation and Control: Verwaltungskomponente von StorHouse/SM.
VxFS	Veritas File System: Kommerzielles Journaling Dateisystem der Firma VERITAS für Solaris, HP-UX und Unix Ware.
Wahlfreier Zugriff	Auf jede Speicherzelle kann in beliebiger Reihenfolge zugegriffen werden (auch Random Access genannt).
WORM	Write Once Read Many: Einmal beschreibbares Medium, wie zum Beispiel die CD-R.
wxWindows	Plattformübergreifende (open source) C++-Programmierbibliothek, um grafische Benutzeroberflächen zu erstellen (http://www.wxwindows.org).
XDSM	X/Open Data Storage Management: Verwaltungseinheit eines HSM-Systems. Realisiert in Form einer API die Anbindung an den Kernel des Betriebssystems. 1997 standardisiert durch die Open Group.

Things should be made as simple as possible, but not any simpler.
Albert Einstein (1879 - 1955)

Abbildungsverzeichnis

There is nothing as practical as a good theory.
Kurt Lewin (1890-1947)

Tabellenverzeichnis

The beginning of wisdom is to call things by their right names.
Chinese Proverb

Definitionsverzeichnis

Reading furnishes the mind only with materials of knowledge; it is thinking that makes what we read ours.
John Locke (1632 – 1704)

Literaturverzeichnis

Adic03 Advanced Digital Information Corporation: *FileServ High-Performance Centralized HSM*; ADIC, FileServ Datasheet, 2003. http://www.adic.com/us/collateral/fileserv_datasheet.pdf

AI92 ANSI, ISO/IEC: *Database Language SQL*; Document ANSI X3.135-1992 or document ISO/IEC 9075:1992, 1992.

Akel01 Akelbein J. P.: *Hierarchisches Speichermanagement – Datenautomaten*; iX Magazin für professionelle Informationstechnik, Heise Zeitschriften Verlag, Seite 108 - 112, Hannover, Deutschland, Dezember 2001.

Albe04 Albers J.: *Konzeptioneller Entwurf und Implementierung eines Data Dictionary Systems für das multidimensionale Array DBMS RasDaMan*; Diplomarbeit, Fakultät für Informatik, Technische Universität München, April 2004.

Alte04 Alternate Computerversand GmbH, August 2004. http://www.alternate.de

Baum04 Baumann P.: *Rasterdatenbanken am Beispiel rasdaman*; Datenbank Spektrum, Zeitschrift für Datenbanktechnologie, Ausgabe 10, dpunkt Verlag GmbH, Heidelberg, Deutschland, August 2004.

Baum99 Baumann P.: *A Database Array Algebra for Spatio-Temporal Data and Beyond*; Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems (NGITS), pages 76 - 93, 1999.

Baye97 Bayer R.: *The Universal B-Tree for Multidimensional Indexing: General Concepts*; Proceedings of the International Conference of Worldwide Computing and its Applications (WWCA), Tsukuba, Japan, March 1997.

BDFR98 Baumann P., Dehmel A., Furtado P., Ritsch R., Widmann N.: *The Multidimensional Database System RasDaMan*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), pages 575 - 577, Seattle, Washington, USA, June 1998.

BDFR99 Baumann P., Dehmel A., Furtado P., Ritsch R., Widmann N.: *Spatio-Temporal Retrieval with RasDaMan*: Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), pages 746 - 749, Edinburgh, Scotland, September 1999.

Bela66 Belady L. A.: *A Study of Replacement Algorithms for Virtual Storage Computers*; IBM Systems Journal, Volume 5, Number 2, pages 79 - 101, 1966.

BFRW97 Baumann P., Furtado P., Ritsch R., Widmann N.: *Geo/Environmental and Medical Data Management in the RasDaMan System*; Proceedings of the 23th International Conference on Very Large Data Bases (VLDB), pages 548 - 552, Athens, Greece, September 1997.

BG01 Bauer A., Günzel H. (Hrsg.): *Data-Warehouse Systeme – Architektur, Entwicklung, Anwendung*; Dpunkt Verlag, Heidelberg, Deutschland, 2001.

BM72 Bayer R., McCreight E. M.: *Organization and Maintenance of Large Ordered Indexes*; Acta Informatica, Volume 1, Fasc. 3, pages 173 - 189, 1972.

CB00 Carino F., Burgess J.: *StorHouse/Relational Manager (RM) – Active Storage Hierarchy Database System and Applications*; Proceedings of the 8th NASA Goddard Conference on Mass Storage Systems and Technologies, pages 179 - 186, Maryland, USA, March 2000.

CB01 Carino F. Jr., Burgess J.: *StorHouse.com – The Data Management Service Provider*; Proceedings of the 18th IEEE Symposium on Mass Storage Systems, San Diego, USA, April 2001.

Cher94 Chervenak A. L.: *Tertiary Storage: An Evaluation of New Applications*; Dissertation, University of California at Berkeley, USA, 1994.

CIEJ03 Childers E. R., Imaino W., Eaton J. H., Jaquette G. A., Koeppe P. V., Hellman D. J.: *Six orders of magnitude in linear tape technology: The one-terabyte project*; Research and Development Journal, Volume 47, Number 4, IBM Corporation, California, USA, July 2003.

CKK00 Carino F. Jr., Kaufmann A., Kostamaa P.: *Are you ready for Yottabytes? StorHouse – Federated and Object/Relational Solution*; Proceedings of the 8th NASA Goddard Conference on Mass Storage Systems and Technologies, Maryland, USA, March 2000.

CKKB01 Carino F. Jr., Kostamaa P., Kaufmann A., Burgess J.: *StorHouse Metanoia – New Applications for Database, Storage & Data Warehousing*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), Santa Barbara, California, USA, May 2001.

CO98 Carino F., O'Connell W.: *Plan-Per-Tuple Optimization Solution - Parallel Execution of Expensive User-Defined Functions*; Proceedings of the 24th International Conference on Very Large Data Bases (VLDB), pages 690 - 695, New York, USA, August 1998.

Codd70 Codd E. F.: *A Relational Model for Large Shared Data Banks*; Communications of the ACM, 13 (6), S. 377 – 387, 1970.

Com79 Comer D.: *The Ubiquitous B-Tree*; ACM Computing Surveys, Volume 11, Number 2, June 1979.

CPS95 Callaghan B., Pawlowski B., Staubach P.: *RFC 1813 - NFS Version 3 Protocol Specification*; Sun Microsystems, California, USA, June 1995

CWTH02 Curtis K., Wilson W. L., Tackitt M., Hill A. J., Campbell S.: *High Density, High Performance Data Storage via Volume Holography: The Lucent Technologies Hardware Platform*; Bell Laboratories, Lucent Technologies, Murry Hill, New Jersey, USA, 2002.

CZR01 Chao T., Zhou H., Reyes G.: *Compact Holographic Data Storage System*; Proceedings of the 18th IEEE Symposium on Mass Storage Systems, pages 237 - 247, San Diego, USA, April 2001.

Dätw98 Dätwyler M.: *Optische Speichermedien und Datenarchivierung*; Eidgenössische Materialprüfungs- und Forschungsanstalt (EMPA), Bericht Nr. 239, St. Gallen, Schweiz, 1998.

Degr01 DeGraaf C.: *Tape Drive Technology Comparison*; Spectra Logic Corporation, Colorado, USA, November 2001.

Dehm02 Dehmel A.: *A Compression Engine for Multidimensional Array Database Systems*; Dissertation, Fakultät für Informatik, Technische Universität München, Deutschland, 2003.

DKLP94 DeWitt D., Kabra N., Luo J., Patel J., Yu J.: *Client-Server Paradise*; Proceedings of the International Conference on Very Large Data Bases (VLDB), pages 558 - 569, Santiago, Chile, 1994.

Ecma95 Standard ECMA-222: *Adaptive Lossless Data Compression Algorithm*; ECMA, Switzerland, June 1995.

EH84 Effelsberger W., Härder T.: *Principles of Database Buffer Management*; ACM Transaction on Database Systems (TODS), ACM Press, Volume 9, Number 4, pages 560 - 595, New York, USA, December 1984.

Exab01 Exabyte Corporation: *Exabyte Mammoth-2 Tape Drive – Product Specification*; Exabyte Corporation, Colorado, USA, September 2001.

File03a FileTek Incorporation: *Transparent Oracle Access to StorHouse – Exploit Your Data Assets While Protecting Your Oracle Investment*; FileTek Incorporation product sheet, 2003.

File03b FileTek Incorporation: *StorHouse/UDB Link – Extending IBM DB2 Universal Database (UDB) Capabilities with StorHouse/UDB Link*; FileTek Incorporation product sheet, 2003.

File03c FileTek Incorporation: *Transparent SQL Server 7.0 Access to StorHouse – Extend Your Database Capabilities While Protecting Your SQL Server 7.0 Investment*; FileTek Incorporation product sheet, 2003.

Furt99 Furtado P.: *Storage Management of Multidimensional Arrays in Database Management Systems*; Dissertation, Fakultät für Informatik, Technische Universität München, Deutschland, 1999.

GG98 Gaede V., Günther O.: *Multidimensional Access Methods*; ACM Computing Surveys Volume 30, Number 2, 1998.

GKDT02 Gheller C., Kleese van Dam K., Diedrich E., Toussaint F., Samsonova M. Robin E.: *HPC with and without innovations of the ESTEDI project*; ESTEDI project review meeting, Munich, Germany, May 2002.

GMW01 Giles H., McGowan P., Weekley S.: *Quantum DLTtape Handbook – Your Complete Guide to Today's Hottest Storage Technology*; Quantum Corporation Handbook, 8th Edition, California, USA, 2001.

GMW95 Golubchik L.: *Analysis of Striping Techniques in Robotic Storage Libraries*; Proceedings of the 14th IEEE Symposium on Mass Storage Systems, Monterey, California, USA, March 1995.

Grae93 Graefe G.: *Query Evaluation Techniques for Large Databases*; ACM Computing Surveys, 25(2), pages 73 - 170, 1993.

Gutt84 Guttman A.: *R-Trees: A dynamic index structure for spatial searching*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), pages 47 - 54, Boston, USA, 1984.

Hahn05 Hahn, K.: *Parallele Anfrageverarbeitung für multidimensionale Array-Daten*; Dissertation, Fakultät für Informatik, Technische Universität München, Deutschland, erscheint 2005.

Hilb1891 Hilbert D.: *Über die stetige Abbildung einer Linie auf ein Flächenstück*; Mathematische Annalen 38, Seiten 459 - 460, 1891.

HMD02 Hughes J., Milligan C., Debiez J.: *High Performance RAIT*; Proceedings of the 19th IEEE Symposium on Mass Storage Systems, Maryland, USA, April 2002.

Hoag85 Hoagland A. S.: *Information Storage Technology: A Look at the Future*; IEEE Computer Volume 18, Number 7, pages 60 - 67, 1985.

HPSS03 IBM Corporation: *HPSS – Basics of the High Performance Storage System*; IBM Global Services, Texas, USA, 2003.
http://www4.clearlake.ibm.com/hpss/about/HPSS-Basics.pdf

HR02 Hahn K., Reiner B.: *Intra-Query Parallelism for Multidimensional Array Data*; Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China, August 2002.

HR99 Härder T., Rahm E.: *Datenbanksysteme – Konzepte und Techniken der Implementierung*; Springer Verlag, Berlin, Deutschland, 1999.

HRH03 Hahn K., Reiner B., Höfling G.: *Parallel Query Support for Multidimensional Data: Intra-object Parallelism*; Proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA), Prague, Czech Republic, September 2003.

HRHB02 Hahn K., Reiner B., Höfling G., Baumann P.: *Parallel Query Support for Multidimensional Data: Inter-object Parallelism*; Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA), Aix en Provence, France, September 2002.

HS00 Heuer A., Saake G.: *Datenbanken - Konzepte und Sprachen*; MITP-Verlag, 2. Auflage, Bonn, Deutschland, Januar 2000.

HS03 Harms U., Stiller A.: *Datenfluten und Sackgassen*; c't Magazin für Computertechnik, Heise Zeitschriften Verlag, Seite 54, Heft 15, Hannover, Deutschland, 2003.

HS96 Hillyer B. K., Silberschatz A.: *Random I/O Scheduling in Online Tertiary Storage Systems*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), pages 195 - 204, Montreal, Quebec, Canada, June 1996.

IBM03a IBM Corporation: *Tivoli Space Manager for UNIX using the Hierarchical Storage Management Clients*; IBM Corporation, Version 5, Release 2, April 2003.

IBM03b IBM Corporation: *Backup-Archive Clients Installation and User's Guide*; IBM Corporation, Version 5, Release 2, April 2003.

Ingr04 Ingram Micro Distribution GmbH, Großhändler für Produkte der Informationstechnologie, August 2004.
http://www.ingrammicro.de

Inmo02 Inmon B.: *The Infinite Data Warehouse – Cost-Efficient Extending Stroage Capacity with Near-Line Technologies*; White paper, Legato Systems Incorporation, USA, 2002.

JM98 Jaedicke M., Mitschang B.: *On Parallel Processing of Aggregate and Scalar Functions in Object-Relational DBMS*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), pages 379 - 389, Santa Barbara, California, USA, May 2001.

JM99 Jaedicke M., Mitschang B.: *User-Defined Table Operators: Enhancing Extensibility for ORDBMS*; Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), pages 494 - 505, Edinburgh, Scotland, September 1999.

Jni01 JNI Corporation: *An Introduction to InfiniBand – Bringing I/O up to speed*; White Paper, Version 1.1, February 2001.

KE99 Kemper A., Eickler A.: *Datenbanksyteme – Eine Einführung*; R. Oldenburg Verlag, 3. Auflage, München, Deutschland, 1999.

Knut98 Knuth D. E.: *The Art of Computer Programming – Volume 3 Sorting and Searching*; Addison-Wesley Verlag, October 1998.

Lamb94 Lamb P.: *Tiling Very Large Rasters*; Proceedings of the 6th International Symposium on Spatial Data Handling, pages 449 - 460, Edinburgh, Scotland, 1994.

Laut02 Lautenschlager M.: *Daten-Infrastruktur bei M&D und DKRZ*; Workshop on Definition of a Community Climate Data Archive at DKRZ, Hamburg, Deutschland, März 2002.

Laut95 Lautenschlager M.: *Data Handling in the Climate Model Archive at DKRZ*; Proceedings of the 9th International Symposium on Computer Science for Environmental Protection - Space and Time in Environmental Information Systems, pages 287 - 294, Marburg, Deutschland, 1995.

Lavs03 Lavsa O.: *Entwicklung und Implementierung von Object Framing zum adaptiven Zugriff auf multidimensionale Datenbanken und zur erweiterten Datenanalyse*; Diplomarbeit, Fakultät für Informatik, Technische Universität München, Deutschland, Dezember 2003.

Lega03a Legato Systems Incorporation: *DiskXtender for Unix – System Administrator Guide Release 2.5*; Legato System Incorporation, California, USA, February 2003.

Lega03b Legato Systems Incorporation: *DiskXtender Database Edition*; Legato Systems Incorporation product sheet, 2003.
http://portal2.legato.com/resources/datasheets/D112.pdf

Lijd03 Lijding M. E. M.: *Real-Time Scheduling of Tertiary Storage*; Dissertation, University of Twente, Enschede, Netherlands, 2003.

LRZ03 Leibniz Rechenzentrum München: *TSM-Server am LRZ*; LRZ-Mitteilung, München, Deutschland, Oktober 2003.
http://www.lrz-muenchen.de/services/datenhaltung/adsm/adressen/

LT01 Lautenschlager M., Thiemann H.: *Semantic oriented data access and storage at MPIM/DKRZ*; Proceedings of the 18th IEEE Symposium on Mass Storage Systems, pages 79 - 84, San Diego, USA, April 2001.

LVDS03 Lyman P., Varian H. R., Dunn J., Strygin A., Swearingen K.: *How Much Information?*; Study of University of California at Berkeley, California, USA, 2003.

Mass97 Massiglia P.: *The Raid Book: A Source Book for Raid Technology*; Raid Advisory Board, January 1997.

MD02 Messmer H., Dembowski K.: *PC Hardwarebuch – Aufbau, Funktionsweise, Programmierung*; Addison-Wesley Verlag, Bonn, Deutschland, 2002.

Mech01 Mecher E.: *Erhöhung der Sensitivität photorefraktiver holographischer Speichermedien auf Basis von amorphen organischen Materialien*; Dissertation, Fakultät für Chemie und Pharmazie, Ludwig-Maximilians-Universität München, Deutschland, 2001.

Mell02 Mellanox Technologies Incorporation: *InfiniBand in the Internet Data Center – Reliability, Availability and Serviceability – Past, Present and Future*; White Paper, Document Number WP260800150, 2002.

Milz03 Milz O.: *Top-K Vektor Anfragen im multidimensionalen Datenbanksystem RasDaMan*; Diplomarbeit, Fakultät für Informatik, Technische Universität München, Deutschland, Dezember 2003.

Moor02a Moore, F.: *Storage Manifesto*; STORAGETEK Storage Technology Corp., Louisville, USA, 2002.

Moor02b Moore, F.: *Disk and Tape Pricing Guideline*; White Paper, Horison Information Strategies, Colorado, USA, 2002.

Moor65 Moore G. E.: *Cramming more components onto integrated circuits*; Electronics, Volume 38, Number 8, April 1965.

Mort66 Morton G. M.: *A computer oriented geodetic data base and a new technique in file sequencing*; Technical report, IBM Corporation, Ottawa, Canada, 1966.

MS98 Midtstraum R., Sandsta O.: *Random I/O Performance of a Tandberg MLR1 Tape Drive*; Norwegische Informatikkonferenz (NIK), pages 91 - 102, Kristiansand, Norwegen, November 1998.

Nall00 Nall, J.: *Choosing the Right Tape Technology*; DELL Technologies Power Solutions Articles, pages 62 - 65, 2000.

OCL99 O'Connell W., Carino F., Linderman G.: *Optimizer and Parallel Engine Extensions for Handling Expensive Methods Based on Large Objects*; Proceedings of the 15th International Conference on Data Engineering (ICDE), IEEE Computer Society Press, pages 304 - 313, Sydney, Australia, March 1999.

OGS97 Open Group Standrad: *Systems Management: X/Open Data Storage Management (XDSM) API*; Open Group Technical Standard, U.K., February 1997.

Olso92 Olson M. A.: *Extending the POSTGRES Database System to Manage Tertiary Storage*; Master Thesis, University of California, Berkeley, USA, 1992.

OM84 Orenstein J. A., Merrett T. H.: *A class of data structures for associative searching*; Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, pages 181 - 190, Ontario, Canada, April 1984.

OOW93 O'Neil E. J., O'Neil P. E., Weikum G.: *The LRU-K Page Replacement Algorithm For Database Disk Buffering*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), Washington, USA, May 1993.

Optw04 Optware Corporation: *What's New: The world's first movie recording on a performatted holographic disc*; Yokohama, Japan, September 2004. http://www.optware.co.jp/english/what_040823.html

PAA97 Prabhakar S., Agrawal D., El Abbadi A.: *Impact of Media Exchanges in Robotic Storage Libraries*; Technical Report TRCS97-07, Department of Computer Science, University of California, Santa Barbara, 1997.

PAAS96 Prabhakar S., Agrawal D., El Abbadi A., Singh A.: *Efficient I/O Scheduling in Tertiary Libraries*; Technical Report TRCS96-26, Department of Computer Science, University of California, Santa Barbara, 1996.

PAAS97 Prabhakar S., Agrawal D., El Abbadi A., Singh A.: *Efficient I/O Scheduling in Tertiary Libraries*; Proceedings of 8th International Workshop on Database and Expert Systems Applications (DEXA), IEEE Computer Society Press, Toulouse, France, September 1997.

PCS04 PC-Shop Datentechnik Geist, August 2004. http://www.pc-shop.de

PGK88 Patterson D. A., Gibson G., Katz R. H.: *A Case for Redundant Arrays of Inexpensive Disks (RAID)*; Proceedings of the International Conference on Management of Data (ACM SIGMOD), pages 109 - 116, June 1988.

Quan03a Quantum Corporation: *SDLT 600 Product Specification*; Quantum Corporation, USA, 2003.

Quan03b Quantum Corporation: *SDLT 320 Product Specification*; Quantum Corporation, USA, 2003.

Quan96 Quantum Corporation: *DLT2000/DLT2500/DLT2700 Cartridge Tape Subsystem Product Manual*; Quantum Corporation, California, USA, January 1996.

Rahm94 Rahm E.: *Mehrrechner-Datenbanksysteme – Grundlagen der verteilten und parallelen Datenbankverarbeitung*; Addison-Wesley Verlag, Oktober 1994.

Rasd02 Rasdaman GmbH: *RasDaMan Query Language Guide Version 5.0*; Rasdaman GmbH, München, Deutschland, 2002.

Rein01 Reiner B.: *Tertiary Storage Support for Multidimensional Data*; Workshop Supercomputing Databases at the 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy, September 2001.

Rein03a Reiner, B.: *3rd ESTEDI Newsletter*; periodical ESTEDI project newsletter, Munich, Germany, February 2003.

Rein03b Reiner B.: *Fusion von Datenbanktechnologie und Tertiärspeichersystem im Bereich sehr großer multidimensionaler Array-Daten – Stand der Forschung*; Technischer Forschungsbericht TUM-I0318, Fakultät für Informatik, Technischen Universität München, Deutschland, November 2003.

RH02a Reiner B., Hahn K.: *Tertiary Storage Support for Large-Scale Multidimensional Array Database Management Systems*; Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), Hong Kong, China, August 2002.

RH04a Reiner B., Hahn K.: *HEAVEN – A Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems*; Proceedings of the 9th International Conference on Extending Database Technology (EDBT), Heraklion, Crete, Greece, March 2004.

RH04b Reiner B., Hahn K.: *Optimized Management of Large-Scale Datasets stored on Tertiary Storage Systems*; IEEE Distributed Systems Online Magazin, Published by the IEEE Computer Society, May 2004.

RHHB02 Reiner B., Hahn K., Höfling G., Baumann P.: *Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems*; Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA), Aix-en-Provence, France, September 2002.

Rits99 Ritsch R.: *Optimization and Evaluation of Array Queries in Database Management Systems*; Dissertation, Fakultät für Informatik, Technische Universität München, Deutschland, 1999.

RSV02 Rigaux P., Scholl M., Voisard A.: *Spatial Databases with Application to GIS*; Morgan Kaufmann Publishers, Academic Press, San Francisco, USA, 2002.

Sara95a Sarawagi S.: *Query Processing in Tertiary Memory Databases*; Proceedings of the 21st VLDB Conference, Zurich, Swizerland, September 1995.

Sara95b Sarawagi S.: *Database Systems for Efficient Access to Tertiary Memory*; Proceedings of the 14th IEEE Mass Storage Systems Symposium, Monterey, California, USA, 1995.

Sara96 Sarawagi S.: *Query Processing in Tertiary Memory Databases*; PhD Thesis, University of California, Berkeley 1996.

Schw02 Schwarz, T.: *Magnetic Tape*; Proceedings of the 10th NASA Goddard Conference on Mass Storage Systems and Technologies, Maryland, USA, April 2002.

SH99 Saake G., Heuer A.: *Datenbanken Implementierungstechniken*; MITP-Verlag, Bonn, Deutschland, Mai 1999.

SM04 Das Storage Magazin: *Marktübersicht: Tape-Libraries*; July 2004. http://www.speicherguide.de/markt/marktuebersicht.asp?theID=3

SM99 Sandsta O., Midtstraum R.: *Improving the Access Time Performance of Serpentine Tape Drives*; Proceedings of the 15th International Conference on Data Engineering (ICDE), IEEE Computer Society Press, pages 542 - 551, Sydney, Australia, March, 1999.

SO93 Stonebraker M., Olson M.: *Large Object Support in POSTGRES*; Proceedings of the 9th International Conference on Data Engineering, Vienna, Austria, April 1993.

SRF87 Sellis T., Roussopoulos N., Faloutsos C.: *The R+-Tree: A Dynamic Index for Multi-Dimensional Objects*; Proceedings of the 13th International Conference on Very Large Data Bases (VLDB), pages 507 - 518, Brighton, England, 1987.

SS94a Sarawagi S., *Stonebraker M.: Efficient Organisation of Large Multidimensional Arrays*; Proceedings of the 10th International Conference on Data Engineering, Houston, Texas, USA, 1994.

SS94b Sarawagi S., Stonebraker M.: *Efficient Organization of Large Multidimensional Arrays*; Proceedings of the ICDE, pages 328 - 336, 1994.

Sun02 Sun Microsystems: *Sun SAM-FS and Sun SAM-QFS Storage and Archive Management Guide*; Sun Microsystems, California, USA, August 2002.

Sun04a Sun Microsystems: *Sun StorEdge 9970 and 9980 Systems*; Sun Microsystems Product Specification, California, USA, August 2004.
http://www.sun.com/storage

Sun04b Sun Microsystems: *Sun StorEdge 3310 SCSI Array*; Sun Microsystems Product Specification, California, USA, August 2004.
http://www.sun.com/storage/workgroup

Tane03 Tanenbaum A. S.: *Computernetzwerke*; Prentice Hall, Pearson Studium, 4. Auflage, München, Deutschland, 2003.

TG01 Tanenbaum A. S., Goodman J.: *Computerarchitektur: Struktur, Konzepte, Grundlagen*; Prentice Hall, Pearson Studium, 2. Auflage, München, Deutschland, 2001.

TG99 Triantafillou P., Georgiadis I.: *Hierarchical Scheduling Algorithms for Near-Line Tape Libraries*; Proceedings of the 10th International Conference and Workshop on Database and Expert Systems Applications (DEXA), Florence, Italy, September 1999.

Völz96 Völz H.: *Informationsspeicher – Grundlagen – Funktion – Geräte*; Expert Verlag, Renningen-Malmsheim, Deutschland, 1996.

Watk00 Watkins S.: *LTO Ultrium Delivers New Levels of Tape Data Protection*; Dell Technologies, Power Solutions Articles, pages 58 - 60, 2000.

WCMS01 Werner M., Coutts C., Marion M. Saavedra N.: *The IBM LTO Ultrium Tape Libraries Guide*; Redbook IBM Corporation, California, USA, July 2001.

Widh04 Widhopf-Fenk, R.: *Advanced Concepts and Applications of the UB-Tree*; Dissertation, Fakultät für Informatik, Technische Universität München, Deutschland, 2004.

Widm00 Widmann N.: *Efficient Operation Execution on Multidimensional Array Data*; Dissertation, Fakultät für Informatik, Technische Universität München, Deutschland, 2000.

WSC00 Wegner A., Shu Q., Cheng Y.: *Optical Head Design for 1 TB Optical Tape drive*; LOTS Technology Inc., Sunnyvale, California, USA, 2000.

WT99 Weiss R., Tsien P.: *Oracle and Nearline Storage: Strategies for Interoperability*; Technical Paper No. 753, Oracle Corporation, 1999.