Sammlung und Nutzung freier Ressourcen in Weitverkehrsnetzen

Michael May

Institut für Informatik

Sammlung und Nutzung freier Ressourcen in Weitverkehrsnetzen

Michael May

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.(komm.) Dr. Thomas Ludwig
Prüfer der Dissertation:	1. Univ.-Prof. Dr. Arndt Bode
	2. Univ.-Prof. Dr.-Ing. Djamshid Tavangarian, Universität Rostock

Die Dissertation wurde am 3. April 2000 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 23. Juni 2000 angenommen.

Zusammenfassung

Globale Weitverkehrsnetze bieten oftmals einen enormen Überschuß von Ressourcen wie Rechenzeit, Hauptspeicher und Plattenplatz, deren Nutzung aber bisher an den hohen Kosten augrund ihrer Verteiltheit, Heterogenität und Unzuverlässigkeit scheitert. Die vorliegenden Arbeit entwickelt daher Modelle und Methoden, um brachliegende Ressourcen über eine einheitliche Schnittstelle in einem elektronischen Markt nutz- und handelbar zu machen.

Die Analyse bestehender Infrastrukturen zum uniformen Zugriff auf verteilte, heterogene Ressourcen aus den Bereichen des Meta- und Web-Computings fördert eine Reihe von Defiziten wie geringer infrastruktureller Wirkungsgrad, Orientierung auf Ressourcennachfrager statt Anbieter und fehlende Teilnahmeanreize zutage, die zu einer geringen Nutzung freier Rechenressourcen führen. Aufbauend auf dieser Analyse werden verschiedene Anforderungen identifiziert, um einen Markt für immaterielle Leistungen wie Rechenressourcen modellieren zu können. Diese Anforderungen – asymmetrische Modellierung, transparente Teilnahme, dezentrale Verwaltung, marktbasierte Zuteilungsverfahren und funktionierende Teilnahmeanreize – führen zur Entwicklung von drei Säulen, die das Fundament des vorgestellten Modells eines elektronischen Ressourcenmarktes sind: Ausnutzung multiplikativer Effekte, Ausnutzung der Lokalität und Anwendung von Rekursivität.

Aufbauend auf diesem Modell wurde eine generische Recheninfrastruktur Locust (LOw cost Computing Utilizing Skimmed idle Time) entworfen und implementiert, die überschüssige Ressourcen abschöpft und für grobgranulare parallele und verteilte Anwendungen verwendet. Locust erreicht eine hohe akkumulierte Rechenleistung durch die Einbeziehung einer Hierarchie von Zwischenhändlern (Multiplikatoren), die die Reichweite des Systems zur Sammlung und Entlohnung von Ressourcen vergrößern, durch die loakle, dezentrale, marktbasierte Verwaltung sowie die rekursive Aufteilung und Verwaltung von überlasteten Märkten bis zum Gleichgewichtszustand. Das Ziel aller drei Ansätze ist die weitest mögliche Reduktion der Transaktionskosten bei der Sammlung und Nutzung freier Ressourcen.

Der Nachweis der allgemeinen Anwendbarkeit und Leistungsfähigkeit der Infrastruktur erfolgt anhand der Evaluierung zweier Anwendungsklassen, dem verteiltem Hochleistungs- und Hochdurchsatzrechnen. Die erste Klasse wurde durch zwei grobgranulare parallele Anwendungen, dem verteilten RC5- und des Raytracing-Algorithmus, repräsentiert. Abgerundet wird der Evaluierung durch Implementierungskonzepte aus der zweiten Anwendungsklasse, dem *Content-Caching* und *-Delivery* sowie dem *Client-Server*-Test. Die Arbeit schließt damit die Lücke zwischen der geringen Leistungsfähigkeit und Transparenz des *Common Gateway Interface* (CGI) und den extrem hohen Kosten dedizierter Meta-Computing-Umgebungen zum Zugriff auf verteilte, heterogene Ressourcen.

Danksagung

Eine wissenschaftliche Arbeit ist für ihr Gelingen auf eine Reihe von Erfolgsfaktoren angewiesen. An dieser Stelle möchte ich all denjenigen danken, die für solche positiven Faktoren verantwortlich waren.

Einen ganz entscheidenden Anteil kommt der Arbeitsumgebung zu, in der eine wissenschaftliche Arbeit entsteht. Mein herzlicher Dank gebührt meinem Doktorvater, Prof. Dr. Arndt Bode, der es versteht, am Lehrstuhl für Rechnertechnik und Rechnerorganisation ein Arbeitsklima zu schaffen, das einen großen wissenschaftlichen Freiraum bietet, dabei aber stets ein offenes Ohr für die Anliegen seiner Mitarbeiter hat. Ebenso möchte ich Prof. Dr. Tavangarian für die Übernahme des Zweitgutachtens dieser Arbeit danken.

Einen weiteren Faktor bildet das wissenschaftliche Umfeld, das mit einen offenen Gedankenaustausch zur Präzisierung von unklaren Ideen beiträgt. Insbesondere die Diskussion mit meinen Kollegen aus der Anwendergruppe des Lehrstuhl und aus anderen Lehrstühlen war hierbei ein große Hilfe. Stellvertretend für alle anderen möchte ich an dieser Stelle Dr. Peter Luksch und Claudia Gold danken, die als kritische Begutachter dieser Arbeit viele Hinweise und Verbesserungsvorschläge machten. Dank gehört auch meinen KollegInnen Sabine Rathmayer, Günther Rackl, Jörg Trinitis, Detlev Fliegl, Markus Lindermeier und Philipp Drum, mit denen ich zum Ausgleich interessante fachliche und inbesondere nicht-fachliche Gespräche führen konnte. Besonderer Dank gilt meinem Kollegen Ivan Zoraja, der mir an nicht wenigen arbeitsreichen Abenden und Wochenenden in der Hochschule Gesellschaft leistete.

Eine wichtige Rolle nehmen ebenfalls studentische Arbeiten ein, die in Form von Diplomarbeiten und Systementwicklungsprojekten in eine wissenschaftliche Arbeit eingehen. An dieser Stelle möchte ich Fabian Loschek Lob und Dank für seine hilfreichen Hinweise und engagierten Implementierungsarbeiten zukommen lassen.

Neben den rein wissenschaftlichen Aufgaben fallen an einem Lehrstuhl leider auch viele Verwaltungs- und Administrationstätigkeiten an, die einen Doktoranden beträchtlich von seinem Forschungsvorhaben ablenken können. Ich möchte dem Sekreatariat, namentlich Frau Heike Eberhardt und Frau Brunnhuber, sowie der Systemadministration in Person von Klaus Tilk dafür danken, daß sie am Lehrstuhl für Rechnertechnik und Rechnerorganisation stets für einen reibungslosen Ablauf und Betrieb einer sehr heterogenen Verwaltungs- und Rechnerinfrastruktur gesorgt haben.

Schließlich danke ich meinen Eltern, meinem Bruder und seiner Frau dafür, daß sie sich während der letzten strapazenreichen Zeit um mein geistiges und leibliches Wohl gekümmert und mir so manchen lästigen Behördengang und Verwaltungsakt abgenommen haben, um mir so den Rücken für meine Arbeit freizuhalten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Das exponentielle Wachstum von Weitverkehrsnetzen, insbesondere des Internet mit seinen über 70 Millionen Rechenknoten aus mehr als 58 angeschlossenen Nationen, hat zu einer drastischen Vergrößerung des Pools verfügbarer Ressourcen und der immer dringender werden Notwendigkeit geführt, auf diese heterogenen Ressourcen uniform zugreifen zu können. Das Potential von global verteilten Rechenressourcen ist bereits in einer Reihe von schlagzeilenträchtigen verteilten Rechenprojekten zutage getreten, wie der Dechiffrierung verschiedener kryptografischer Verfahren wie DES [Pub93] und RC5 [Riv95] oder der Suche nach Mersenne Primzahlen [Rob54] und irregulären Spuren in kosmischer Strahlung [SET].
Die Notwendigkeit des uniformen Zugriffs auf verteilte Ressourcen hat ihrerseits eine Reihe von Entwicklern von Meta-Computing-, Web-Computing- und Ressourcenhandels-Infrastrukturen dazu bewegt, sich mit dieser Problematik näher auseinanderzusetzen. Während Meta-Computing-Systeme den Zugriff auf dedizierte und zuverlässige Hochleistungshardware und -netzwerke mit Hilfe von sehr effizienten, aber geschlossenen Protokollen und Programmiermodellen zur Verfügung stellen, richten sich Web-Computing-Infrastrukturen eher an gelegentliche Nutzer und temporäre Ressourcen des *World Wide Web* (WWW), die nur zeitweise online sind und für eine Nutzung zur Verfügung stehen. Im Web-Computing werden die unter Umständen weniger effizienten, aber offenen und allgegenwärtigen Protokolle und Programmiermodelle des WWW wie Java, Browser und HTTP verwendet.

Das Ziel dieser Arbeit ist die Erschließung und Nutzung der brachliegenden Rechenressourcen Tausender anonymer, nur zeitweise angeschlossener Teilnehmer des WWW. Dazu ist es notwendig, diese Ressourcen von ihrem rohen und heterogenen Grundzustand in eine standardisierte homogene Ressource zu transformieren, in Anlehnung an andere homogene, bereits im WWW handelbare Dienste, wie zum Beispiel Speicherplatz (*Web Space*), Email- oder Namens-Dienste. Diese Standarddienste sind inzwischen so homogen und austauschbar wie elektrischer Strom, bei dem es keine Rolle spielt, von welchem Anbieter er kommt. Homogene Dienste und Produkte erlauben Kunden, den für sie besten Anbieter auszuwählen und jederzeit bei Unzufriedenheit zu einem anderen Dienstleister wechseln zu können, ohne logistische oder technische Einbußen erleiden zu müssen. Homogenität und Standardisierung sind wichtige Eigenschaften einer allgegenwärtigen Infrastruktur wie zum Beispiel Telefon- und Telegraphie- oder IP-Netze. Falls es gelingt, verteilte freie Ressourcen ähnlich zu standardisieren und homogen anzubieten, entsteht ein “elektri-

scher Strom", der verteilte und parallele Anwendungen auf Rechen-Netzen der nächsten Generation (*Computational Grids*) antreiben kann.

Verteiltes Rechnen auf heterogenen Rechenknoten in Weitverkehrsnetzen bietet mehr Vorteile als auf den ersten Blick deutlich werden. Persönliche Workstations und PCs, die den Großteil der zeitweise angeschlossenen Rechenknoten des Internet ausmachen, sind oft mit den neuesten Prozessor- und Bustechnologien ausgestattet, sei es aus beruflichen Gründen – um beispielsweise große technisch-wissenschaftliche Anwendungen zu rechnen und zu visualisieren – oder aus privaten Gründen – um zum Beispiel die neuesten 3D Spiele zu spielen. Diese Prozessoren sind erst dann für den Einsatz in Parallelrechnern verfügbar, wenn die nächste Generation von PC Prozessoren den Markt betritt. Dadurch eilt die reine Rechenleistung von Workstations und PCs pro Prozessor massiv parallelen und symmetrischen Multiprozessor-Systemen ungefähr um ein bis zwei Jahre voraus. Diese Tatsache kann die höhere Latenz und geringere Bandbreite von vernetzten Arbeitsplatz-Rechnern teilweise ausgleichen. Nicht selten jedoch liegt die leistungsfähige Hardware brach, während der Benutzer Mail oder News liest oder Web-Seiten betrachtet.
Workstations sind teilweise bis zu 90 % ihrer Laufzeit untätig [GK94]. Wenn man annimmt daß die ans Internet angeschlossenen Rechenknoten geografisch gleichmäßig über die Erde verteilt sind und unter der Annahme, daß höchstens die Hälfte von ihnen Tageslicht liegt, wird sofort deutlich, daß mindestens 50 % aller an das Internet angeschlossenen Rechner im Dunkeln und damit eventuell brach liegen. Bei mindestens 58 angeschlossenen Nationen ist es fast sicher, daß 10 bis 14 Zeitzonen entfernt ganze Kontinente von Rechenknoten weitgehend ungenutzt sind.

Ein oft geäußerter Einwand gegen die Nutzung freier Ressourcen ist die Tatsache, daß sich Systeme zur gemeinsamen Nutzung von Ferienwohnungen oder zum *Car-Sharing* in der freien Marktwirtschaft kaum durchsetzen konnten. Auch wird gelegentlich auf die ständig sinkenden Preise für Rechenleistung als Argument gegen den Aufwand zur Sammlung ungenutzter Rechenressourcen hingewiesen. Diese Argumentation ist jedoch nicht schlüssig.
Es ist richtig, daß die reine Rechenleistung pro Dollar ständig steigt und damit immer preiswerter zu haben ist. Allerdings reizt die Hardwareindustrie die technologischen Möglichkeit neuer Systeme bis aufs Äußerste aus, da nur in den oberen Marktsegmenten ausreichende Gewinne zu erzielen sind. Dies führt dazu, daß die Einstiegspreise für neue Rechensysteme relativ konstant geblieben sind. Es werden damit zwar Rechenressourcen fortwährend preiswerter, nicht aber die Bereitstellung zusätzlicher Rechenleistung, die stets die Anschaffung zusätzlicher Rechensysteme zu konstanten Preisen erfordert. Rechenleistung ist über Neuanschaffungen nur in sehr diskreten Stufen und Preisen erweiterbar, ganz im Gegensatz zu einem Rechenzeit-Markt, der kontinuierliche Rechenressourcen anbieten kann.
Ein Rechenleistungs-Markt hat darüber hinaus den Vorteil, daß im Gegensatz zu Systemen zur gemeinsamen Ressourcennutzung in der freien Wirtschaft die Transaktionskosten, die den erzielbaren Nutzen mindern und eine große Hürde für die Akzeptanz darstellen, extrem niedrig gehalten werden können. Ein Großteil dieser Transaktionskosten wie Lager-, Transport- und Abschreibungskosten, entsteht in der freien Wirtschaft aus der Tatsache, daß materielle Güter gehandelt und vermietet werden, die der Abnutzung unterliegen. Immobilien nutzen sich ab und müssen instand gehalten werden, Automobile verschleißen

und müssen zum Ausgangspunkt zurückgebracht werden, Tatsachen, die den Wert einer Ressource subjektiv und objektiv mindern. Viele dieser Nebenbedingungen und -kosten, die die Attraktivität gemeinsam genutzter Ressourcen schmälern, treten bei der Nutzung immaterieller Güter wie zum Beispiel Rechenleistung nicht auf: Rechenressourcen nutzen sich nicht ab, sondern werden vollständig verbraucht, der Transport an den Bestimmungsort ist wesentlich preiswerter und schließlich können und müssen Rechenressourcen auch nicht gelagert werden.

1.2 Entwicklung des technisch-wissenschaftlichen und verteilten Hochleistungsrechnens

Zum Verständnis der Probleme, wie sie bei der heutigen Nutzung global verteilter Ressourcen auftreten, ist ein kurzer Abriß der Geschichte des technisch-wissenschaftlichen und verteilten Rechnens hilfreich. Aber auch als Beispiel für den synchronen Fortschritt von Hard- und Software-Infrastruktur, Anwendungen und Ansprüchen der Anwender innerhalb eines Fachgebietes, sei im folgenden kurz die Entwicklung des technisch-wissenschaftlichen Hochleistungsrechnens skizziert.

Die ersten Großrechensysteme der 50er und 60er Jahre boten dem Nutzer mit ihrer geringen Speicherausstattung, ihrer Maschinensprachen-Programmierung und ihren Stapelverarbeitungssystemen so gut wie keinen Bedienkomfort und konnten daher nur für numerische Probleme geringer Komplexität genutzt werden. Mit der Einführung von FORTRAN und den ersten Vektorrechnern wurde dann auch die Lösung größerer und komplexerer wissenschaftlicher Probleme möglich. Mit dem Aufkommen der Supercomputer und hoch-optimierender Fortran-Compiler hielten dann die ersten echten *High Performance (Scientific) Computing* (HPC)-Anwendungen komplexer Simulationen und neuer numerischer Algorithmen Einzug in die Rechenzentren. Mit kleineren und preiswerteren Parallelrechnern begann danach die Verbreitung des HPC in die Forschungseinrichtungen, Institute und größeren Firmen.

Die achtziger Jahre sorgten als Zeitalter der Workstations und Personal Computer zusammen mit grafischen Benutzeroberflächen und neuen Programmiersprachen wie C für eine Verlagerung der Rechenleistung von den bis dahin dominierenden zentralen Mainframes weg zu den *Personal Computern* auf den Schreibtischen. Diese Rechenleistung wurde hauptsächlich für interaktive Büroanwendungen benutzt, die Restkapazitäten insbesondere nachts und am Wochenende blieben ungenutzt. Durch die Entwicklung von Standard-Programmierumgebungen für den Nachrichtenaustausch auf netzgekoppelten Rechnern wie *Parallel Virtual Machine* (PVM) oder *Message Passing Interface* (MPI), ließen sich jedoch die brachliegenden Rechenkapazitäten von Workstation- und PC-Netzen zur Entwicklung und Ausführung paralleler Programme nutzen. Im Zuge der weiteren Leistungssteigerungen im Workstation- und PC-Bereich etablierten sich Workstation-Netze (*Network Of Workstations, NOW*) als preiswerte Parallelrechner und MPI als portabler Programmierstandard sowohl für NOWs als auch für massiv parallele Prozessoren. Der Standardisierung eines portablen Interfaces folgten auch bald die ersten kommerziellen HPC Softwareprodukte im Bereich der *Computation Fluid Dynamics* (CFD) und etablierten damit das HPC in der industriellen Praxis.

Die im Zuge der 90er Jahre voran geschrittene Globalisierung durch den stetigen Aus-

bau der Weitverkehrsnetze, insbesondere des Internet, ermöglicht nun die Koppelung auch sehr weit entfernter Rechner und Workstations zu einem Rechnerverbund und bietet dabei ähnliche Vorteile wie sie NOWs geboten haben. Die beteiligten Einzelrechner sind preiswert in der Anschaffung, stehen außerhalb des Rechnerverbunds auch für andere Aufgaben zur Verfügung und bieten die Möglichkeit, ungenutzte Rechenleistung für das HPC zu verwenden. Man stellt jedoch schnell fest, daß die bloße Übertragung der Methoden und Verfahren, wie sie zur Programmierung von NOWs ausreichend waren, auf das Meta- oder Web-Computing nicht genügt, um das enorme Potential von über Weitverkehrsnetze gekoppelten Rechnerverbünden vollständig auszunutzen. Es mangelt demnach an Modellen und Methoden, um der Vielzahl von verfügbaren Ressourcen in Weitverkehrsnetzen Herr zu werden und effizient von ihnen Gebrauch zu machen.

1.3 Rechnen mit freien Ressourcen

Werden die im Zusammenhang mit der Nutzung von NOWs eingesetzten Programmierumgebungen und gewonnenen Erfahrungen auf globale Netze übertragen, wird das Weitverkehrsnetz (*Wide Area Network*, WAN) nur unzureichend ausgenutzt. Bei einem solchen Vorgehen wird nur der infrastrukturelle Aspekt eines WANs verwendet, der "soziale" Aspekt, wie zum Beispiel die Vielzahl und Heterogenität der teilnehmenden Institutionen und Benutzer, die sich in unterschiedlicher Administration und unterschiedlichen Nutzungsgewohnheiten offenbart, bleibt unberücksichtigt. Das bedeutet, die beteiligten Nutzer eines Nahverkehrsnetzes (*Local Area Network, LAN*) sind homogen, zahlenmäßig gleichstark und betreiben einen Ausgleich oder Austausch von freien Ressourcen. Darüber hinaus unterliegen sie der gleichen oder zumindestens kooperierender administrativer Kontrolle. Diese Struktur ähnelt einer zentralen Planwirtschaft: reichen Ressourcen zur Erwirtschaftung der von einer zentralen Instanz vorgegebenen Aufgaben nicht aus, können zusätzliche Ressourcen von anderen Betrieben und Institutionen entliehen oder "beschafft" werden, ohne daß die entliehenen Ressourcen bezahlt und bilanziert werden müssen.
Die Benutzer eines Weitverkehrsnetzes unterscheiden sich dagegen sehr stark von denen eines LANs, sie sind äußerst heterogen, zahlen- und ortsmäßig sehr stark gestreut, und unterliegen verschiedener administrativer Kontrolle. Verfahren und Programmiermodelle, die Eigenschaften eines LANs voraussetzen und auf diesen aufbauen, sind daher zur Nutzung global verteilter Ressourcen völlig ungeeignet. Es werden vielmehr Methoden und Modelle benötigt, die die Strukturen eines WANs wie Heterogenität, Asymmetrie und lokale Administration berücksichtigen und widerspiegeln und sich damit an einer freien Marktwirtschaft anlehnen. Es ergeben sich demnach eine Reihe von Anforderungen an Modelle und Verfahren zur Sammlung und Vermarktung immaterieller Güter wie Rechenleistung und Speicherkapazität in Weitverkehrsnetzen.

Asymmetrische Modellierung – Traditionelle Modelle und Methoden zur gemeinsamen Nutzung von Ressourcen in LANs und WANs basieren auf einer symmetrischen Modellierung der beteiligten Parteien und Ressourcen. Das WAN dient lediglich zur Überbrückung größerer Entfernungen als sie in LANs möglich sind, die Nutzung unterscheidet sich jedoch nicht von der in einem LAN und folgt einem symmetrischen Modell (*peer-to-peer*). Die Geber und Nehmer eines LANs, sowohl menschlicher als auch infrastruktureller Na-

tur wie Maschinenpark und Netzwerk-Ressourcen, sind meist homogen und unterliegen der gleichen oder zumindest kooperierender administrativer Kontrolle. Die Benutzer eines Weitverkehrsnetzes unterscheiden sich dagegen stark von denen eines LANs, sie sind äußerst heterogen, zahlen- und ortsmäßig verteilt und unterliegen verschiedener administrativer Kontrolle. Zur Überwindung von Heterogenität und Verteilung sind WANs auf spezielle Middleware Protokolle und Sprachen angewiesen, die sich jedoch zum heutigen Zeitpunkt allesamt durch relativ geringe Effizienz auszeichnen. Diese Ineffizienz kann in einem skalierenden System durch eine sehr hohe Zahl von Teilnehmern ausgeglichen werden. Auch finden sich in der Regel sehr viele Benutzer, die bereit sind, Rechenleistung gegen Unkostenerstattung abzugeben, aber abhängig von den Kosten unter Umständen nur sehr wenige Nachfrager. Eine Nutzung freier Rechenressourcen in WANs ist daher inhärent asymmetrisch, ein Umstand, der sich auch in der Modellierung und Implementierung einer Infrastruktur widerspiegeln muß, die die Sammlung und den Handel von Ressourcen ermöglichen soll.

Transparente Teilnahme – Im Gegensatz zu etablierten nachrichtenbasierten Rechen-Infrastrukturen und Meta-Computing Systemen, die sich ausschließlich an erfahrene, wissenschaftliche Benutzer richten, müssen gelegentliche Benutzer von WANs in einem asymmetrischen Modell ebenfalls als zeitweise Anbieter von freien Rechenkapazitäten mit einbezogen werden. Gelegentliche Nutzer des WWW stellen dabei im allgemeinen zu wenig Rechenressourcen zur Verfügung, als daß sich die Einrichtung und Pflege eines eigenen Benutzerkontos zur eindeutigen Identifizierung und Abrechnung mit realen Währungen rechtfertigt. Aus der hoher Anzahl und dem geringen Kenntnisstand von Gelegenheitsbenutzern ergibt sich darüberhinaus die Notwendigkeit, auf aktive Teilnahme an einer Ressourcensammlung mit Benutzerkonten zu verzichten und stattdessen eine automatische und anonyme Teilnahme ohne aktive Beteiligung zu ermöglichen.

Marktbasierte, dezentrale Verwaltung – Die große Zahl sehr heterogener Teilnehmer läßt sich von einer allwissenden zentralen Steuerkomponente nicht bewältigen. Lösung dieses Problems sind marktbasierte Ressourcenzuteilung-Strategien, die auf dezentraler Entscheidungsfindung mit Hilfe lokaler Informationen beruhen. Dezentrale Verwaltungsmechanismen ermöglichen die gleichzeitige Erfüllung sehr vieler Nebenbedingungen und Präferenzen von Marktteilnehmern und fördern die Entwicklung lokaler Handelspolitiken, die örtliche Besonderheiten hinsichtlich Transaktionskosten, Sicherheitsanforderungen und lokalisierten Teilnahmeanreizen widerspiegeln.
Traditionelle prioritätenbasierte Ressourcen-Zuteilungsverfahren haben große Schwierigkeiten bei der Vergabe mehrerer gleichberechtigter Ressourcen und außerhalb geschlossener Systeme. Sie sind asymmetrisch, zentral und geschlossen und sind aus diesem Grund für die Zuteilung mehrerer Ressourcen in sehr großen WANs unter verschiedenen administrativen Bereichen nicht geeignet. Marktbasierte Allokationsverfahren dagegen sind in der Lage, mehrere Ressourcen durch dezentrale und lokale Entscheidungsfindung effizient zuzuteilen, und liefern Ergebnisse, die selbst in sehr großen Systemen einer optimalen Zuteilung sehr nahe kommen. Rechenressourcen sind typische Vertreter knapper Ressourcen, deren Wert bei Nichtgebrauch verfällt. Ein marktbasiertes Zuteilungsverfahren ermöglicht den Export und Import von Ressourcenüberschüssen und erleichtert hierdurch den Aufbau von Ressourcenvorräten.

Funktionierende lokalisierte Anreize – Funktionierende Anreize sind Voraussetzung für die Nutzung eines Marktsystems. Diese fundamentale Tatsache ist bei einer Reihe universitärer Projekte zwangsläufig unberücksichtigt geblieben, da sie keine langfristig funktionierenden Anreize zur Nutzung ihrer Infrastrukturen bieten konnten und wollten. Eine Buchführung und Abrechnung aller Teilnehmer, die unter Umständen nur winzige Ressourcenmengen ein- oder ausführen, ist aber schon allein aus Gründen der Organisation und Verwaltung nicht skalierbar möglich. Aus diesem Grund wird zur weitmöglichsten Reduktion der Transaktionskosten der Einsatz von Mikrowährungen[1] zur Entschädigung von Marktteilnehmern niedriger hierarchischer Ebenen des Systems vorgeschlagen. Ein Austausch und eine Auszahlung in realen Währungen erfolgt nur an Zwischenhändler in übergeordneten Hierarchien des Händlersystems für größere Mengen angehäufter Ressourcen.

Hierarchische Verwaltung – Durch die Einbindung einer großen Zahl von technischen Laien entstehen Probleme, die Umsicht bei der Architektur erfordern. Um Hunderttausende von Teilnehmern mit geringem Ressourcenumsatz verwalten zu können, werden in einer Hierarchie organisierte Zwischenhändler benötigt, die Management, Kontenverwaltung und Abrechnung für eine Vielzahl von kleinen, anonymen Lieferanten übernehmen. Durch die hierarchische Organisation kann das Management des Ressourcenmarktes lokalen Händlern und ihren Teilnehmern überlassen werden, die jeweils ihre eigenen Präferenzen für Sammlung und Nutzung von Ressourcen verfolgen. Diese Präferenzen können durch Zielgruppen, Transaktionskosten, administrative Bereiche oder andere Parameter bestimmt sein.
Über eine dezentrale und hierarchische Verwaltung ist ebenfalls eine vereinfachte, lokalisierte Marktteilnahme erreichbar. Die Teilnehmer in den unteren Ebenen der Organisationshierachie werden durch lokale Händler verwaltet, die mittels lokaler Politiken zur Sammlung und Nutzung von Ressourcen dafür sorgen (können), daß die Voraussetzungen für eine einfache Teilnahme wie Anonymität, automatische Teilnahme, geringe Transaktionskosten, funktionierende, lokalisierte Anreize und Bedingungen usw. in ihrem Teilmarkt erfüllt sind.

Aus den genannten Voraussetzungen kristallisieren sich drei Säulen heraus, die einen Markt für überschüssige Rechenressourcen tragen können: Multiplikation, Lokalität und Rekursivität.

Ausnutzung multiplikativer Effekte – Es werden deutlich mehr Anbieter als Verbraucher benötigt, um mit freien Ressourcen höhere akkumulierte Rechenleistung zu erzielen. Als Zielgruppen müssen daher neben technisch fachkundigem Publikum (als Abnehmer der Rechenleistung) auch Durchschnittsbenutzer des WWW als Anbieter in Betracht genommen werden. Die effiziente Verwaltung einer derart großen Teilnehmergruppe kann nur durch eine dezentrale Organisation mit Zwischenhändlern als Multiplikatoren bewältigt werden, die die Reichweite und Kapazität eines Ressourcenmarktes durch die Übernahme des Verwaltungs- und Buchführungsaufwands für einen Teilmarkt vervielfachen.

Berücksichtigung der Lokalität – Die Verwaltung einer sehr großen Zahl von Endteilnehmern ist von einer zentralen, allwissenden Steuerkomponente nicht möglich. Statt-

[1] wie Online-Inhalten oder Informationen

dessen ist eine dezentrale Entscheidungsfindung auf Basis marktbasierter Verfahren, die zum größten Teil auf lokalen Informationen beruhen, besser geeignet, Ressourcen effizient zuzuteilen. Verschiedene Teilmärkte besitzen darüberhinaus auch sehr unterschiedliche Anforderungen an Transaktionskosten, Sicherheitsanforderungen und Teilnahmebedingungen und -anreize. Mit Hilfe lokalisierter Politiken zur Sammlung, zum Handel und zur Abrechnung von Ressourcen, können Multiplikatoren lokale Besonderheiten berücksichtigen und gezielt zur weitest möglichen Reduzierung der Verwaltungs- und Transaktionskosten ausnutzen.

Anwendung von Rekursivität – Zum Aufbau und zur Verwaltung hierarchisch organisierter Zwischenhändler und Teilmärkte findet das Rekursivitätsprinzip Anwendung. Zu große, inhomogene Märkte und überlastete zugehörige Server-Dienste werden rekursiv in kleinere und homogenere Einheiten aufgeteilt, hierarchisch gekoppelt und dezentral von den untergeordneten Server-Diensten aus verwaltet. Dieser Prozeß terminiert, falls eine weitere Aufteilung ist nicht mehr möglich ist, daß heißt die entstandenen Teilmärkte sind homogen geworden und ihre Server-Dienste befinden sich in einem Gleichgewichtszustand.

Aufbauend auf dem Gesagten wurde das prototypische Ressourcen-Handelssystem Locust (LOw cost Computing Utilizing Skimmed idle Time) entworfen und implementiert, das auf diesen drei Säulen aufsetzt. Haupt-Zielgruppe dieses Systems sind gelegentliche Nutzer des WWW, die nur durch den Besuch einer Web-Seite oder *Web Site* automatisch und anonym Rechenressourcen zur Verfügung stellen. Eine Anmeldung oder Registrierung, Installation oder selbst der bewußte Start einer Anwendung ist hierzu nicht notwendig.
Verwaltet wird die große Zahl anonymer Anbieter von Rechenressourcen über ein hierarchisch organisiertes Netz von Haupt- und Zwischenhändlern, die in mehreren hierarchischen Ebenen die Sammlung und den Handel der ihnen untergeordneten Händler und, auf unterster Ebene, WWW Teilnehmer verwalten. Die Rolle der Zwischenhändler wird bei Locust von Cluster-Verwaltern und *Content Providern*, das heißt Erstellern von *Online*-Inhalten, übernommen, die Locust-Applets (*Locii*) in ihre Web-Seiten einbetten. Betreiber von Workstation-Clustern können brachliegende Ressourcen ihrer Systeme durch die Installation einer eigenständigen Locust-Applikation zur Verfügung stellen[2].
Locust-Marktteilnehmer werden durch den Inhalt der Seiten entlohnt, der dadurch die Funktion einer Mikrowährung erfüllt. Die meisten Benutzer nehmen dabei gar nicht wahr, daß sie einen Teil ihrer Ressourcen für den Zugriff auf *Online*-Inhalte wie Börsenkurse, Suchanfragen oder Nachrichten eintauschen. Aggregatoren bzw. Multiplikatoren wie Content-Ersteller und Cluster-Betreiber erhalten im Locust-System virtuelles Geld für die Sammlung von Rechenressourcen. Durch die Ansammlung der heterogenen Ressourcen agiert Locust als Kommissionär und Weiterverarbeiter und ist daher in einem Real-Betrieb – außerhalb eines wissenschaftlichen Umfelds – in der Lage, durch die Schaffung von Mehrwert auf den rohen Eingangsressourcen Gewinn zu erwirtschaften. Der geschaffene Mehrwert besteht aus verschiedenen Basisdiensten – "elektrischer Strom" – zum Bau und Unterhalt von Meta-Computing, Web-Computing und Rechennetz-Infrastrukturen.

[2]und als Teilmarkt mit Ressourcenüberschuß an der Locust-Infrastruktur teilnehmen

1.4 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt. Das zweite Kapitel beschäftigt sich mit aktuellen Techniken zum global verteilten Rechnen, dem Meta- und Web-Computing. Besonderer Beitrag dieses Kapitels zur vorliegenden Arbeit ist die Erarbeitung einer aufeinander aufbauenden Nomenklatur, um Ordnung in die inkonsistent, teilweise sogar widersprüchlich in der Fachliteratur verwendeten Begriffe zu bringen. Anschließend folgt abschnittsweise eine Vorstellung der drei grundlegenden Bereiche dieses Fachgebietes, nämlich des Meta-Computings, des Web-Computings und der Rechen-Netze (*Computational Grids*). Jeder Abschnitt besteht aus einer genauen Beschreibung der entsprechenden Systeme und einer Abgrenzung zu den anderen Bereichen sowie einer Vorstellung eines oder mehrerer Implementierungen einer solchen Infrastruktur. Abschließend folgt eine Klassifikation und Bewertung der bestehenden Systeme im Hinblick auf die Eignung zur Sammlung und Nutzung global verteilter freier Ressourcen.

Einen Einblick in reale und elektronische Märkte bietet das dritte Kapitel. Nach einer allgemeinen Einführung in traditionelle prioritätenbasierte Ressourcen-Zuteilungsverfahren mit ihren Stärken und Schwächen insbesondere in offenen, verteilten Systemen, werden marktbasierte Verfahren zur gleichzeitigen Allokation mehrerer Ressourcen eingeführt. Auf Basis einiger grundlegender Begriffe und Definitionen aus der Marktwirtschaft werden die für den Rest der Arbeit notwendigen marktbasierten Koordinations- und Zuteilungsverfahren vorgestellt. Der Hauptbeitrag dieses Kapitels zur ganzen Arbeit ist die Identifizierung und Behandlung der Besonderheiten beim Handel mit zeitkritischen und opportunistischen Ressourcen mit unendlich hoher Abschreibung. Es folgt eine theoretische Abschätzung der Komplexität und Leistungsfähigkeit dieser Verfahren im Hinblick auf die Anforderung an den Handel von Rechenleistung. Anschließend folgt ein Überblick über konkrete Implementierung von Marktsystemen, sogenannten elektronischen Märkten, der mit einer Klassifizierung und Bewertung im Vergleich zu den Zielen dieser Arbeit das Kapitel abschließt.

Aufbauend auf den bisher erarbeiteten Grundlagen wird in Kapitel 4 eine Modellierung eines Ressourcenmarktes in Weitverkehrsnetzen vorgestellt. Dazu werden zunächst fünf grundlegende Anforderungen zur Etablierung eine Ressourcenmarktes identifiziert, die andere Ansätze nicht oder nur unzureichend berücksichtigt haben. Diese Anforderungen sind asymmetrische Modellierung der Marktteilnehmer orientiert an Ressourcenanbietern, transparente Systemteilnahme, marktbasierte Ressourcenzuteilung, dezentrale, hierarchische Verwaltung über Zwischenhändler sowie die Bereitstellung funktionierender Anreize zur Marktteilnahme. Aus diesen Anforderungen werden drei Säulen abgeleitet, die ein funktionierendes Ressourcen-Handelssystem tragen können, die Ausnutzung von multiplikativen Effekten, von Lokalität und von Rekursivität. Nach Entwicklung dieser Komponenten wird aus ihnen das Gesamtmodell des Ressourcen-Handelssystems Locust entwickelt, dessen Implementierung Mittelpunkt des folgenden Kapitels ist.

Die Architektur und Implementierung des im vorherigen Kapitel vorgestellten Ressourcen-Handelsmodells ist Gegenstand des 5. Kapitels. Zunächst werden eine Reihe von Entwurfszielen wie Skalierbarkeit, Portabilität, Wiederverwendbarkeit und Geschwindigkeit erarbeitet, die Grundlage der im Anschluß beschriebenen Implementierung von Locust sind. In drei Abschnitten wird die technische Realisierung der drei Hauptkomponenten – der Locust-Server, der Locust-Client und der Locust-Markt – und ihr

Zusammenspiel zur Bildung eines verteilten elektronischen Ressourcenmarkt beschrieben, dessen Leistungsfähigkeit im folgenden Kapitel nachgewiesen werden soll.

Kapitel 6 beschäftigt sich mit Anwendungen zur Nutzung der Locust Infrastruktur sowie der Messung und Bewertung ihrer Leistungsfähigkeit. Auf Grundlage verschiedener Anwendungsklassen, die den Grundprinzipien des Locust-Modells – Ausnutzung der Multiplikation, Lokalität und Rekursivität – folgen, werden mögliche und im Rahmen dieser Arbeit implementierte Applikationen vorgestellt. Das Kapitel beginnt zunächst mit der Beschreibung der im Rahmen dieser Arbeit implementierten und evaluierten Locust-Anwendungen, der parallelen RC5 Dechiffrierung und der parallelen Raytracing Anwendung. Ihre erfolgreiche Evaluierung bilden die Basis für den Nachweis der Leistungsfähigkeit der Locust-Modells und seiner Implementierung. Es folgen detaillierte Konzepte weiterer Anwendungen aus dem dem *Web-Caching*, dem *Client-Server*-Tests sowie *Distributed Object Computing*, die mit ausgearbeiteten Implementierungsplänen versehen sind und die allgemeine Anwendbarkeit des Locust-Modells untermauern. Insgesamt erfolgt damit der Nachweis der Nützlichkeit und Leistungsfähigkeit des Systems.

Die Arbeit wird im 7. Kapitel zusammengefaßt und einer kritischen Bewertung unterworfen. Aus dieser Bewertung leiten sich mögliche Ansätze für weitere zukünftige Forschungsarbeiten ab.

Kapitel 2

Meta-Computing und Web-Computing

Das vergangene Jahrzehnt ist durch eine Paradigmen-Migration weg vom *Personal Computing* hin zu einem verteilten Programmiermodell geprägt, dessen bekanntester Vertreter das Internet ist. Dieses globale Weitverkehrsnetz erfreut sich einer exponentiellen Verbreitung und stellt über das *Client-Server*-Modell in erster Linie Informations- und Kommunikationsdienste zur Verfügung. Jedoch bieten Weitverkehrsnetze wie das Internet neben Informationen weitere, teilweise brachliegende Ressourcen für das verteilte Rechnen an.

Der im Zuge der 90er Jahre vorangeschrittene Ausbau der Netzkapazitäten ermöglicht die Koppelung auch sehr weit entfernter Rechner zu einem Rechenverbund und bieten dabei ähnliche Vorteile wie sie *Networks of Workstations* (NOWs) geboten haben. Insbesondere die Zusammenschaltung von bis zu hunderttausend Rechenknoten eröffnet dem technisch-wissenschaftlichen Rechnen Probleme und Problemgrößen – die sogenannten *Grand Challenges* – die auch auf Superrechnern und großen Workstation Netzen nicht lösbar sind. Darüberhinaus ergeben sich aus der Überbrückung weiter Entfernungen in Ort und Zeit auch neue Anwendungsfelder. Dazu zählt die Tele-Kooperation, also die zeitlich gemeinsame, aber örtliche getrennte Bearbeitung von Anwendungen und Dokumenten in der Wirtschaft, Ausbildung oder Medizin. Als Überwindung zeitlicher Unterschiede beim verteilten Rechnen kann die Miete und Vermietung zusätzlicher Rechenleistung zum Ausgleich der Spitzen bei der Auslastung von Rechensystemen gesehen werden. Zur Optimierung der Auslastung wird dabei zu Belastungsspitzen zusätzliche Rechenleistung zugemietet und überschüssige Leistung vermietet, falls das lokale System nicht ausgelastet ist.
Für die Vernetzung verteilter Systeme sprechen demnach eine Reihe von Vorteilen.

Information und Kommunikation – Die mit großem Abstand meist genutzte Anwendung in WANs ist die Informationsbeschaffung und Kommunikation. Voraussetzung hierfür ist eine Abkehr vom Paradigma des *Personal Computing*, das in den achtziger Jahren vorherrschend war, hin zum verteilten Rechen-Paradigma, das die neunziger Jahr dominiert. Bei dem sich in den letzten Jahren explosionsartig verbreitenden *World Wide Web* handelt es sich um ein verteiltes Informationssystem nach dem *Client-Server* Modell, das in erster Linie zum Abruf von elektronischen Informationen und Dateien sowie zur Kommunikation mittels *email, chat* und *IRC* verwendet wird.

Steigerung der Leistungsfähigkeit – Durch Zusammenschalten von Rechenknoten läßt sich die Leistungsfähigkeit eines einzelnen Rechners, die aus technischen, finanziellen und logistischen Gründen beschränkt ist, weiter erhöhen. Die Leistungssteigerung kann dann zur schnelleren Lösung bisheriger Aufgaben oder aber zur Behandlung

größerer Probleme genutzt werden. Diese Anwendung vernetzter Systeme führt zum verteilten Höchstleistungsrechnen.
Eine andere Möglichkeit zur Nutzung der höheren Leistung vernetzter Systeme liegt in der Anwendung neuer Techniken und Verfahren, für die das bisher vorherrschende Paradigma des *Personal Computing* nicht geeignet ist. Zu diesen neuen Techniken gehören die Tele-Kooperation oder die Fernsteuerung entfernter technisch-wissenschaftlicher Geräte. Oft gehörte Schlagworte in diesem Zusammenhang sind *Global Engineering Village* als gemeinsame, interdisziplinäre Ausführungsumgebung für *Grand Challenges* und *Tele-Medizin*.

Markt für Rechenleistung – Das größte Hindernis bei der Markteinführung paralleler Rechner sind – neben der schwierigen Programmierung – die hohen Anschaffungskosten solcher Rechensysteme. Oft wird zudem die zusätzliche Rechenleistung nur zeitweise benötigt, so daß eine Anschaffung eines Parallelrechners noch unrentabler wird. Daher bietet es sich an, solchen Nutzern Rechenleistung zur Miete zur Verfügung zu stellen. Auf einem homogenen und transparenten Rechenleistungs-Markt kann dann jede gewünschte zusätzliche Rechenkapazität zugemietet werden. Darüberhinaus kann durch die Vermietung überschüssiger Rechenleistung die Rentabilität der sehr schnell an Wert verlierenden parallelen Rechen-Hardware eines Rechenzentrums oder Instituts verbessert werden. Auch die Reduzierung fixer Kosten wie Operatoren oder Strom- und Kühlungskosten ist durch Vermietung von Ressourcen auf einem Rechenzeit-Markt möglich.
Neben reiner Rechenleistung ist ebenfalls das Angebot von vollständigen Rechen-Dienstleistungen, also Rechenleistung gebündelt mit entsprechender Software und Anwendungen, vorstellbar. Solche Szenarien werden in der Literatur unter den Stichworten *Computation on demand* und *Application Service Providing* (ASP) geführt.

2.1 Begriffsbildung

Vor einer Beschäftigung mit der Thematik weit verteilter Rechenverbunde steht zunächst einmal die Begriffsfindung und -abgrenzung. Dabei stellt man fest, daß in den wissenschaftlichen Publikationen eine verwirrende Zahl von Begriffen für gleiche oder unterschiedliche Dinge in Gebrauch sind. Verwendet werden *Meta-Computing, Web-Computing, Hyper-Computing, Cluster-Computing* oder auch *Computational Grids*, wobei die verschiedene Autoren durchaus auch mehrere dieser Begriffe zur Unterscheidung verschiedener Szenarien nebeneinander verwenden, die sich jedoch bei den jeweiligen Autoren in der Bedeutung überschneiden.
Verschärft wird diese Problematik durch die Eignung der Begriffe, weniger interessante Veröffentlichungen und Bücher als modische Schlagworte zu schmücken und aufzuwerten. Daher werden diese Begriffe durchaus inflationär verwendet und tauchen auch in relativ fachfremden Publikationen auf, noch dazu mit unklarer Bedeutung. Es existieren damit keine allgemein anerkannten Begriffsdefinitionen für obige Ausdrücke.

Daher folgen zunächst zur Klärung einige Definitionen der Begriffe, die in dieser Arbeit verwendet werden.

Definition 2.1 (Verteiltes System) *Unter einem verteilten System wird eine Anzahl*

verteilter, unter Umständen heterogener, Rechenressourcen wie CPUs, Hauptspeicher, Plattenspeicher und Netzwerke verstanden, die über ein gemeinsames logisches Netz miteinander in Verbindung stehen und ein gemeinsames Problem bearbeiten.

Die an das verteilte System angeschlossenen Ressourcen können dabei dediziert oder opportunistisch sein.

Definition 2.2 (Dedizierte Ressourcen) *Dedizierte Ressourcen garantieren ihre Verfügbarkeit für einen bestimmten Zeitraum.*

Definition 2.3 (Opportunistische Ressourcen) *Opportunistische Ressourcen bieten unzuverlässige Dienste für einen unbestimmten Zeitraum an.*

Die nachfolgenden Abschnitte bauen auf dem Begriff der *Infrastruktur* auf, so daß auch seine genaue Definition gerechtfertigt ist. Der Duden [GD83] definiert eine Infrastruktur als "*Notwendiger wirtschaftlicher und organisatorischer Unterbau einer hochentwickelten Wirtschaft (Verkehrsnetz, Arbeitskräfte und andere).*"

Infrastrukturen entstehen nicht planmäßig, sondern als Folge synergetischer Wechselbeziehungen aus den Möglichkeiten aktueller Technologien, dem Bedarf und Nutzen möglicher Anwendungen und bereits vorhandenen anderen Infrastrukturen. Diese Komponenten können jeweils für sich alleine betrachtet unterhalb eines zur Etablierung neuer Technologien notwendigen Schwellwertes liegen, sich jedoch gegenseitig verstärken und hochschaukeln bis gemeinsam die zur Durchsetzung einer neuen Infrastruktur notwendige kritische Masse erreicht ist.
Beispielsweise orientierten sich menschliche Siedlungen in der Vergangenheit an natürlichen und künstlichen Verkehrswegen wie Flüssen, Häfen oder Eisenbahnstrecken. Aufstieg und auch Niedergang von bedeutenden Handelsplätzen basieren meist auf dem Vorhandensein und dem Wegfall günstiger infrastruktureller Lage oder besonderer Eignung für bestimmte Produktgruppen wie zum Beispiel der Rinderzucht oder dem Getreideanbau. Auf diese Weise können Infrastrukturen auf anderen aufbauen, sie ergänzen oder völlig verdrängen. Beispielsweise ergänzt die augenblickliche Infrastruktur des Internets die dominierenden Informations-Infrastrukturen der Telefon- und Telekommunikations-Anbieter, wird diese jedoch langfristig vollständig ablösen. Schon jetzt wird die bestehende Telekommunikations-Infrastruktur vom Internet für Stand- und Wählleitungen mitbenutzt. In Zukunft ist davon auszugehen, daß auch Telefon- und Faxverkehr über IP Netze befördert werden, eine Tendenz die unter dem Stichwort *Voice over IP* gehandelt wird.

Auch entstehen Infrastrukturen nicht gleichmäßig, sondern breiten sich eher punktuell aus. Ausgehend von lokalen Inseln werden regionale Bereiche untereinander zu größeren Gebieten verbunden bis eine flächendeckende Infrastruktur aufgebaut ist. Allerdings macht erst die allgegenwärtige Verfügbarkeit eine Infrastruktur aus, also schafft erst der Zusammenschluß aus lokalen Technologien eine Infrastruktur. Beispiele für solches punktuelles Wachstum sind Telefon-, Strom- und Eisenbahnnetze, die zunächst aus lokalen oder regionalen Netzen entstanden um schließlich zu nationalen Netzen mit kompletter Abdeckung anwuchsen. Auch die nächste Generation einer Informations-Infrastruktur, das Internet, entstand aus einem regionalen Netz, dem ARPANET, das vom amerikanischen *Department of Defense* unter Verwendung offener Standards und Protokolle wie Paketvermittlung und TCP/IP entwickelt wurde. Deren frei verfügbare Technologien konnten verwendet werden, um auf anderen Kontinenten lokale Netze ähnlicher Struktur zu etablieren,

die schließlich alle zu *dem* Internet zusammengeschlossen wurden. Der Zusammenschluß lokaler Netze, die nicht der gemeinsamen Verwaltung unterliegen, erfordert allerdings die Festlegung und Einhaltung von (offenen) Standards als Schnittstellen lokaler Netze, die die Koppelung und den Zugang vereinheitlichen und konsistent gestalten.

Allen Infrastrukturen gemeinsam ist die Notwendigkeit, das zu transportierende Gut in alle, unter Umständen weit entfernte, Teile der Infrastruktur transportieren zu können. Dazu ist eine dezentrale Verwaltung notwendig, die mit ausgefeilter Lagerhaltung (*Caching*) zur effizienten Verteilung und zum Ausgleich bei Überlastspitzen gekoppelt ist. Diese Mechanismen sind in allen derzeit existierenden Infrastrukturen beobachtbar, so auch bei Verkehrs-, Strom- und Informationsnetzen.

Die eigentlich wichtigste Eigenschaft für eine Infrastruktur folgt zum Schluß, da sie derart selbstverständlich ist, daß sie stillschweigend vorausgesetzt wird. Eine Infrastruktur muß zuverlässig sein und gleichbleibende Qualität bereitstellen, das heißt ein Benutzer erhält stets quantitativ und qualitativ die Leistung, die von der Infrastruktur in Aussicht gestellt wurde. Ohne eine voraussagbare Qualität ist eine Infrastruktur sowohl für Privat- als auch Geschäftspersonen nicht interessant. Geschäftliche Anwender können auf einer Infrastruktur keine aufbauenden Mehrwert-Dienste aufsetzen und auch Privatnutzer werden ihre "kostbare" Freizeit nicht einer unzuverlässigen Infrastruktur anvertrauen.
Die wichtigsten Eigenschaften einer Infrastruktur sind daher zusammenfassend Verfügbarkeit, Konsistenz, dezentrale Verwaltung und Zuverlässigkeit. Aufbauend auf dem eben gesagten folgt nun eine Definition einer Infrastruktur für diese Arbeit.

Definition 2.4 (Infrastruktur) *Als Infrastruktur wird ein Medium bezeichnet, das über standardisierte Schnittstellen universell verwendbar, flächendeckend und preiswert verfügbar und von vorhersehbarer, homogener Qualität ist.*

Diese Definition ist für unsere Zwecke noch zu allgemein. Insbesondere enthält sie keine Aussage über die Komponenten und Dienste sowie die Art der zu transportierenden Güter in der Infrastruktur. Aus diesem Grund erweitern wir die Definition zu einer Rechen-Infrastruktur.

Definition 2.5 ((Verteilte) Rechen-Infrastruktur) *Als (verteilte) Rechen-Infrastruktur wird die abstrakte Softwareschicht mit all ihren Schnittstellen und Protokollen bezeichnet, die die heterogenen und verteilten Ressourcen eines verteilten Systems zur Verfügung stellt und die Eigenschaften einer Infrastruktur besitzt, nämlich Konsistenz, Verfügbarkeit und Zuverlässigkeit.*

Aus dieser Definition einer Rechen-Infrastruktur lassen sich nun alle in diesem Kapitel relevanten Begriffe wie Rechen-Netze, Meta-Computing und Web-Computing ableiten.

Definition 2.6 (Computational Grid (Rechennetz)) *Ein* Computational Grid *oder Rechennetz ist eine verteilte Rechen-Infrastruktur, die Rechenleistung transportiert.*

Laut dieser Definition ist ein Rechennetz eine Infrastruktur zum Transport von Rechenleistung und damit die Voraussetzung für das verteilte oder parallele Rechnen in Weitverkehrsnetzen. Es stellt das Fundament für Basisdienste einer verteilten Rechenumgebung dar, ob sie nun privater, institutioneller oder öffentlicher Natur ist. Um mit einem verteilten System sinnvoll arbeiten zu können, sind neben den reinen Basisdiensten noch

weitere Softwareschichten notwendig, die den Anwendungsprogrammierer von möglichst vielen technischen Details der zugrundeliegenden Infrastruktur abschirmen. Diese werden zum Teil von Meta- und Web-Computing Systemen bereitgestellt, deren Definition nun folgt.

Definition 2.7 (Meta-Computer) *Unter einem Meta-Computer wird eine verteilte Rechen-Infrastruktur verstanden, die als geschlossenes System mit exklusiv verfügbaren Ressourcen und proprietären Schnittstellen und Protokollen Rechenleistung zur Verfügung stellt.*

Damit ist ein Meta-Computing System ein verteiltes System, das mittels einer Softwareschicht dem Benutzer seine Heterogenität verbirgt. Die Verteiltheit des Systems bleibt weiterhin sichtbar, ist also nicht transparent, sondern *transluzent*. Seine Leistungsfähigkeit erreicht ein Meta-Computing System durch den Einsatz dedizierter Hochleistungshardware und speziellen, im allgemeinen proprietären, Schnittstellen und Protokollen. Beispiele für Meta-Computing Systeme mit dedizierter Hardware bzw. Ressourcen und geschlossenen Protokollen und Schnittstellen sind die im nachfolgenden näher beschriebenen Infrastrukturen Globus, Legion oder SuperWeb. Bei diesen sehr anspruchsvollen Projekten handelt es sich um universitäre und institutionelle Rechenumgebungen (*Global Engineering Village*) zur globalen, verteilten Lösung von *Grand Challenges* Problemen, deren Testumgebungen wie I-WAY [DFP+96] und GUSTO aus Supercomputern bestehen, die über Hochgeschwindigkeitsnetze auf FDDI und ATM Technologie verbunden sind. Die Ressourcen des Systems, also die angeschlossenen Superrechner und die Hochgeschwindigkeitsnetze stehen den Benutzern der Meta-Computing Infrastruktur exklusiv zur Verfügung. Anwendungen müssen demnach nicht um Ressourcen miteinander konkurrieren, wie es bei gemeinsam genutzten Systemen wie NOWs üblich ist.

Definition 2.8 (Web-Computer) *Unter einem Web-Computer wird eine verteilte Rechen-Infrastruktur verstanden, deren Ressourcen und Schnittstellen nicht dediziert und proprietär, sondern frei und offen zugänglich sind.*

Im allgemeinen werden dabei die Werkzeuge, Ressourcen und Protokolle des *World Wide Webs* wie zum Beispiel eventuell nur temporär angeschlossene, opportunistische Web-Terminals mit Browser, Java und die Protokolle TCP/IP und HTTP verwendet. Damit ist eine Web-Computing Infrastruktur ein verteiltes System, das mittels einer Softwareschicht dem Benutzer seine Heterogenität und den Opportunismus seiner Ressourcen verbirgt, nicht aber seine Verteiltheit. Wie beim Meta-Computing ist die Verteiltheit transluzent und nicht transparent.

Beiden Spielarten des globalen Rechnens, also Meta- und Web-Computing, sind jedoch auch einige Eigenschaften gemein, die sie von klassischen parallelen Rechnen unterscheiden. Dazu zählen die weitaus größere Heterogenität der beim globalen Rechnen beteiligten Ressourcen wie Knotenrechner und Netzwerk. Die Leistungsfähigkeit dieser Ressourcen ist durch eine bedeutend weitere Streuung gekennzeichnet als entsprechende Bestandteile eines massiv parallelen Prozessors (MPP) oder NOWs. Verschärft wird dies durch die Tatsache, daß auch der Mittelwert deutlich unter dem von traditionellen parallelen Systemen liegt.
Ausgehend von der niedrigeren theoretisch erzielbaren Leistungsfähigkeit der beteiligten

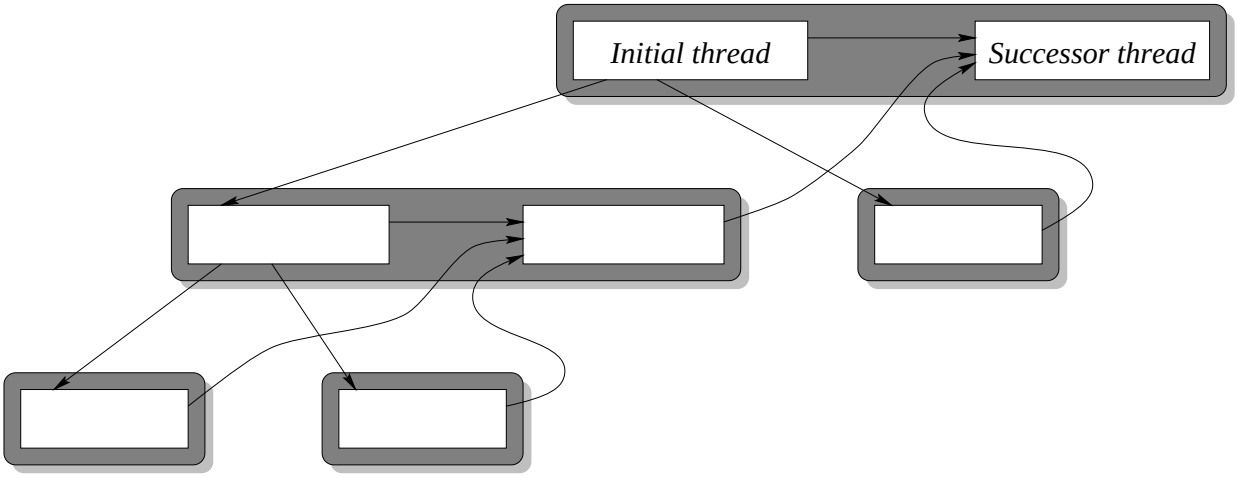

Abbildung 2.1 An Cilk angelehntes Programmiermodell von ATLAS [BBB96a]

Komponenten unterliegt auch die tatsächlich erreichte Leistung einer Ressource starken dynamischen Schwankungen bis zum kompletten Ausfall. Daher haben die Ressourcen auch eine geringere durchschnittliche Fehlerfreiheit (*Mean Time Between Failure* (MTBF) als traditionelle Komponenten.

2.2 Meta-Computing Infrastrukturen

Aufgrund der exponentiell wachsenden Größe von Weitverkehrsnetzen und der zunehmenden Leistungsfähigkeit der Netzwerktechnologie existieren seit kurzer Zeit speziell im universitären Umfeld Anstrengungen, entfernte Hochleistungsrechner mit dedizierten Hochgeschwindigkeitsverbindungen zu Meta-Computing-Infrastrukturen zu koppeln. Im folgenden Abschnitt werden die wichtigsten Vertreter solcher Umgebungen vorgestellt und bewertet.

2.2.1 ATLAS

ATLAS [BBB96a] ist als Ergebnis einer anspruchsvollen Spezifikation mit grundlegenden Anforderungen entstanden, die an eine Meta-Computing Infrastruktur gestellt wurde. Die Spezifikation sieht die Skalierbarkeit des Systems auf Millionen von Rechenknoten, Sicherheitsmechanismen für Benutzer und Ressourcen, ein intuitives und einfaches Benutzer-Interface, ausreichende Leistungsfähigkeit sowie die Adaptivität der parallelen Anwendungen vor.

Bei ATLAS wurde bei der Spezifikation das Programmiermodell streng vom Laufzeitsystem getrennt. Das gegenwärtige Programmiermodell von ATLAS nimmt große Anleihen von dem mehrfädigen Laufzeitsystem Cilk [DFC+96], das die parallele Ausführung von baumartig aufgebauten und rekursiven Anwendungen unterstützt, allerdings auch auf solche beschränkt ist (Vergleiche Abbildung 2.1). Cilk wurde dazu an die Programmiersprache Java angepaßt und schirmt den Anwendungsprogrammierer von allen tatsächlichen physikalischen Rechenressourcen ab. Der Anwender programmiert damit nur eine verteilte virtuelle Maschine und drückt den Parallelismus und die Lokalität seiner An-

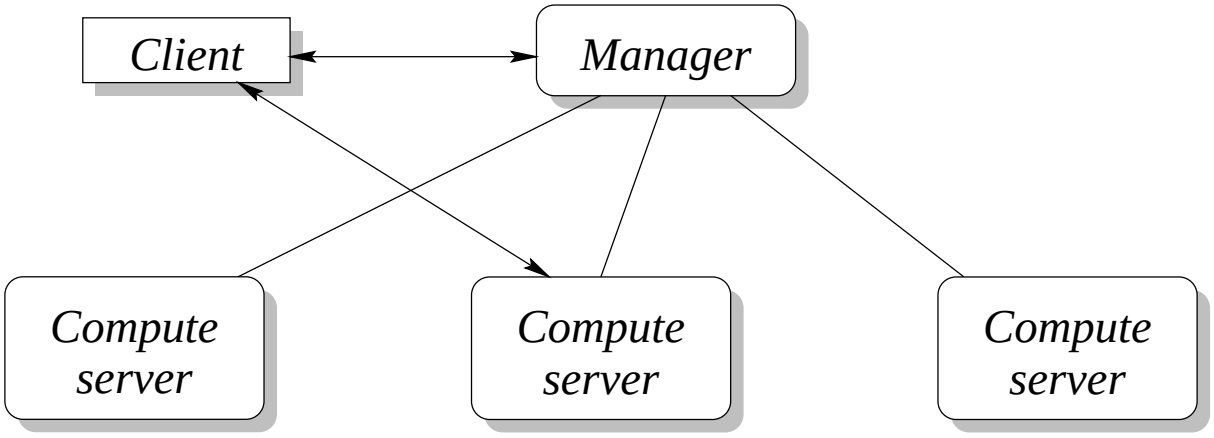

Abbildung 2.2 ATLAS Systemarchitektur [BBB96a]

wendung mit Hilfe von *Threads* aus und überläßt die Abbildung auf die real vorhandene Hardware dem ATLAS Laufzeitsystem, das die Zuteilungs- und Protokolldetails übernimmt. Allerdings werden zur Zeit native Bibliotheken verwendet, die zwangsläufig zu Portabilitätsproblemen führen.

Die ATLAS Systemarchitektur besteht wie in Abbildung 2.2 ersichtlich aus *Clients, Managern* und *Compute Servern*. Ein Client mit Rechenbedarf kontaktiert einen lokalen ATLAS Manager, der die URL eines freien Compute Server zur Verfügung stellt. Daraufhin läßt der Client die Anwendung auf dem Compute Server laufen. Während der Laufzeit haben untätige Compute Server die Möglichkeit, Arbeit von den ausgelasteten Servern zu 'stehlen'. Diese als *Work Stealing* bezeichnete Lastverteilungsstrategie sorgt langfristig für eine Verteilung der Berechnung auf alle verfügbaren Rechenressourcen. Die Compute Server verwalten die ihnen zugewiesenen Threads in einem Keller. Threads werden vom Keller entnommen, bearbeitet und falls der Thread weitere Söhne erzeugt, werden diese wieder auf den Keller aufgebracht. Falls der Keller leer wird, stiehlt der Compute Server Threads von anderen Servern, die auf Anfrage ihr unterstes, also ältestes Kellerelement abgeben. Server arbeiten daher in *Depth-First*-Ordnung und stehlen dagegen in *Breadth-First*-Ordnung.

Die Lastverteilung wird hierarchisch über ein Netz von Ressource-Managern implementiert (vgl. Abbildung 2.3), was eine gute Skalierbarkeit des Systems sicherstellt. Während Compute Server zwischen Geschwistern (innerhalb einer Ebene) Arbeit bzw. Threads verteilen, sorgen die Manager für einen Lastausgleich zwischen den Teilbäumen des ATLAS Systems. Falls ein Manager-Teilbaum zu viel Arbeit zu erledigen hat, werden ihm Threads von oben gestohlen, hat er dagegen zu wenig zu tun, stiehlt er Arbeit von seinen Geschwistern. Das hierarchische Work Stealing sorgt auf natürliche Weise dafür, daß Berechnungen soweit möglich in einem lokalen Teilbereich des Meta-Computers ausgeführt werden, in dem meist bessere Bedingungen hinsichtlich Bandbreite und Latenz herrschen als in weiter verteilten Gruppen. Außerdem entsprechen die Teilbäume der Manager im allgemeinen auch der administrativen Kontrolle von Teilnetzen, so daß die lokalen Manager besondere Richtlinien festlegen und deren Einhaltung überwachen können, zum Beispiel ob Threads den Teilbaum betreten oder verlassen können oder nicht.

Eine einfache Fehlertoleranz wird dadurch erreicht, daß jeder ATLAS Task atomar

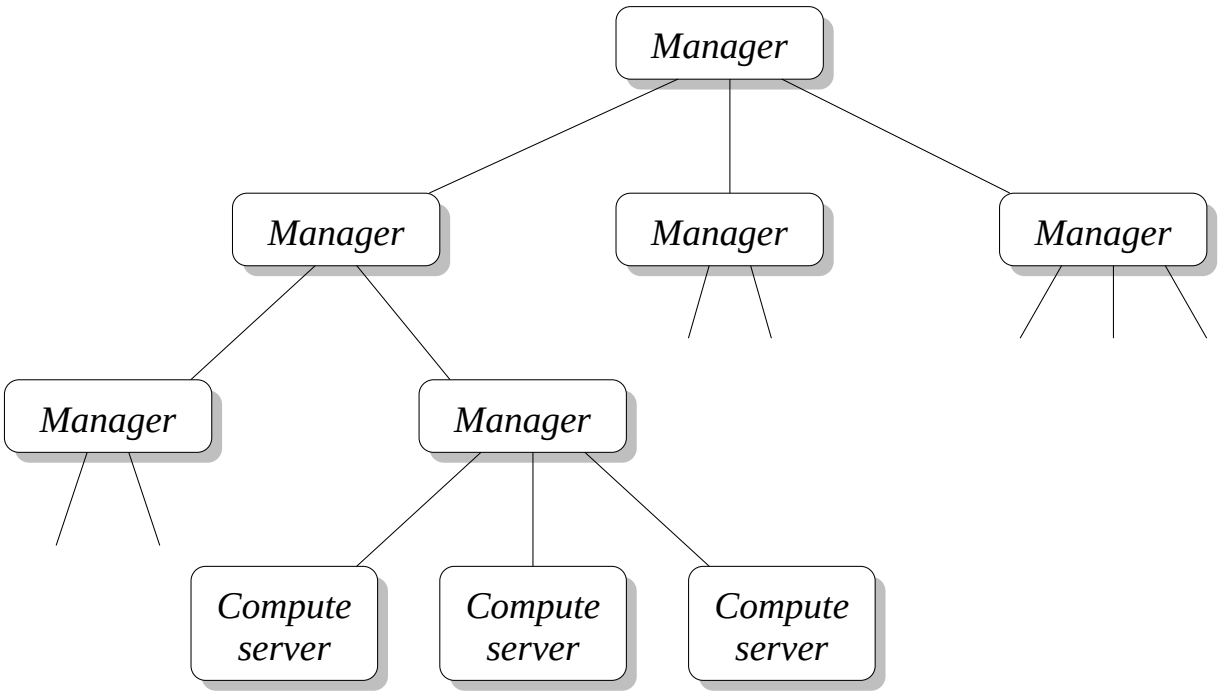

Abbildung 2.3 Hierarchie von Ressourcen-Managern im ATLAS System [BBB96a]

ist, das heißt, falls ein (Sub)Task, der durch einen Teilbaum in der Server Hierarchie bearbeitet wird, nicht rechtzeitig fertig wird, wird die Berechnung anhand eines lokalen Sicherungspunktes in dem entsprechenden Teilbaum neu aufgesetzt.

2.2.2 Charlotte

Das Charlotte System [BBB96b] besteht aus zwei Hauptkomponenten, *Managern*, die Tasks auf Arbeiter Applets verteilen und synchronisieren, und *Workern*, das heißt Teilnehmern mit einem Java-fähigen Browser. Charlotte unterscheidet sich von anderen Infrastrukturen grundlegend durch die strikte Trennung zwischen Programmier- und Laufzeitsystem. Charlotte schirmt den Anwendungsprogrammierer wie ATLAS von der Heterogenität und Verteiltheit des Meta-Computing durch eine *virtuelle Maschine* ab, die vom Laufzeitsystem auf die gerade aktuelle Konfiguration im WWW abgebildet wird. Die virtuelle Maschine von Charlotte ist jedoch nicht mit der von Java, der JVM, zu verwechseln.

Programmiermodell Als Programmiermodell kommt transparenter verteilter gemeinsamer Speicher (DSM) zur Anwendung, der durch Java mit speziellen Klassenerweiterungen implementiert ist. Änderungen an dem verteilten gemeinsamen Speicher werden dabei allerdings erst nach der Beendigung eines Tasks für die übrigen Tasks sichtbar. Aus dieser Atomarität ergibt sich ein einfacher Fehlertoleranzmechnismus.
Programme für die virtuelle Maschine von Charlotte bestehen aus einer Abfolge von sequentiellen und parallelen atomaren Schritten (*fork-join* Modell). Die sequentiellen Schritte dienen dabei zur Steuerung und Synchronisation der parallelen Anwendung, die von

den parallelen Schritten realisiert werden. In diesen parallelen Schritten wird die Parallelität durch nebenläufige Routinen (als *threads* bezeichnet) erreicht, deren Anzahl von den beteiligten Teilnehmern abhängig ist.

Laufzeitsystem Das Laufzeitsystem von Charlotte bildet die virtuelle Maschine und ihre Anwendungen auf die real existierende Hardware Konfiguration ab. Es stellt dabei in erster Linie Lastverteilung und Fehlertoleranz durch den Mechanismus des *Eager Scheduling* sicher, das heißt, solange es noch unerledigte, aber eventuell bereits vergebene Tasks gibt, werden diese weiter an untätige Arbeiter vergeben. Voraussetzung hierfür ist allerdings, daß der Code auch mehrfach ausführbar ist, ohne daß es zu Inkonsistenzen oder Inkonvergenzen kommt. Die mehrfache Vergabe von Tasks führt naturgemäß zur unsparsamen Verwendung von Ressourcen, implementiert jedoch auf einfache Weise gleichzeitig Fehlertoleranz und Lastausgleich ohne daß hierfür eigene Klassen, die diese Dienste zur Verfügug stellen, notwendig sind.

2.2.3 Legion

Das Hauptziel des Legion Projekts [GW94] war die Bereitstellung eines skalierbaren *Middleware* Systems mit sicheren gemeinsamen Namens- und Objekträumen, anwendungskontrollierter Fehlertoleranz und Unterstützung der Interoperabilität vieler Implementierungssprachen für parallele und verteilte Anwendungen. Zur Interoperabilität zählen die Legion Entwickler auch die Anbindung von altem Code, sogenannten *Legacy*-Anwendungen, die meist in den Programmiersprachen COBOL, Fortran und PL/1 vorliegen. Diese können auf drei verschiedene Arten in das Legion System eingebunden werden: über die Generierung von Objekthüllen, sogenannten *Wrappern*, durch die Zurverfügungstellung von Compiler-Backends, die Legion Code erzeugen können oder durch eine Kombination beider Verfahren. Über solche in den fremden Code eingesetzte *Stubs* können populäre andere Middleware Systeme wie CORBA [Gro98], OSF DCE [OHE96] oder RPC [OHE96] angebunden werden. Darüberhinaus bietet Legion Compiler-Entwicklern den Export der Schnittstelle des Laufzeitsystems an, um ein Legion Frontend zur Generierung lauffähiger Legion Programme zu entwickeln. Teil der ersten Implementierung des Legion Systems ist das objektorientierte parallele Laufzeitsystem Mentat, das – wie eine Interface Definition Language (IDL) – zur Kapselung von Methoden systemfremder Programmiersprachen verwendet werden kann.

Die Besonderheit der Legion Infrastruktur [GW94] ist ihre objektorientierte Implementierung, die alle für die Ausführung auf dem heterogenen Meta-Computer wichtigen Parameter und Informationen wie Rechner, Benutzer, Anwendungen und deren Daten als transparente Objekte anbietet. Die Objektorientierung ermöglicht eine weitere Besonderheit der Legion Philosophie, nämlich die möglichst weite Konfigurierbarkeit des Systems an Wünsche und Bedürfnisse des Benutzers. Dabei unterscheidet Legion zwischen der Art und dem Grad einer angebotenen Funktionalität, die beide vom Benutzer steuerbar sind. Objekte und Methoden sind hierfür soweit wie möglich parametrisierbar und konfigurierbar sein, auch die Überlagerung und Ersetzung von Klassen durch die Nutzer ist vorgesehen.

Obwohl nicht explizit angegeben handelt es sich bei den von Legion verwalteten und dem Nutzer angebotenen Ressourcen um dedizierte Hard- und Software.

2.3 Rechen-Netze - *Computational Grids*

Laut Definition stellt ein Rechennetz das Fundament verteilter Rechenumgebungen dar, indem es Basisdienste zum Transport und Einsatz von Rechenleistung zur Verfügung stellt. Um mit einem verteilten System sinnvoll arbeiten zu können, sind neben den reinen Basisdiensten noch weitere Softwareschichten notwendig, die den Anwendungsprogrammierer von möglichst vielen technischen Details der zugrundeliegenden Infrastruktur abschirmen. Aus diesem Grund ist die Grenze zwischen Rechen-Netzen, die nur Infrastruktur und Basisdienste bieten, und verteilten Rechenumgebungen, die von infrastrukturellen Eigenheiten wie Verteiltheit, Heterogenität und Ausfall von Ressourcen abstrahieren, fließend.

Foster und Kesselman haben in ihrer Arbeit [FK98b] Rechen-Netze genauer untersucht. Sie prognostizieren eine stark zunehmende Bedeutung von Rechen-Netzen in der internationalen und interdisziplinären Forschung und Entwicklung. Um die Bedeutung und die Wechselwirkungen zwischen dem Aufbau einer Infrastruktur und den darauf aufsetzenden Anwendungen zu untermauern, schaffen sie den Begriff des Rechen-Netzes (*Computational Grid*) in Analogie zum elektrischen Stromnetz (*Powergrid*). Sie analysieren Klassen von Rechennetz-Anwendungen und -anwendern und stellen Anforderungen an die Architektur von Rechen-Netzen zusammen, die auf bereits bestehenden Systemen aufbauen und diese zu einer verteilten Rechen-Infrastruktur erweitern können.

2.3.1 Rechennetz-Anwendungen

Rechen-Netze stellen auf Anforderung große Mengen verteilter Rechenressourcen mehr oder weniger transparent zur Verfügung. Als prototypische Anwendungsklassen kommen daher das Höchstleistungsrechnen, datenintensive und durchsatzoptimierte Anwendungen, *On-Demand* Anwendungen sowie Tele-Kooperation und Tele-Immersion in Frage.

Verteiltes Höchstleistungsrechnen – Beim verteilten Höchstleistungsrechnen handelt es sich um die Lösung von parallelen numerischen Problemen, die auf einzelnen Rechnern nicht oder nur mit reduzierter Auflösung berechenbar sind. Verteilte Super-Computing Anwendungen besitzen enorme Anforderungen an Ressourcen, die die Möglichkeiten einzelner Institutionen sprengen. Hierbei ermöglichen Rechen-Netze – auch in ihrer jetzigen primitiven Form – eine Berechnung von Problemen in ungeahnten Größenordnungen. Das Spektrum möglicher Probleme reicht dabei von Strömungssimulationen auf einer Reihe von Arbeitsplatz-Rechnern bis hin zu über Hochgeschwindigkeitsnetze gekoppelten Supercomputern zur Lösung von *Grand Challenge* Problemen wie sie im Maschinen- und Fahrzeugentwurf, bei der Halbleiterherstellung und Klimasimulation vorkommen.

Durchsatzoptimierung – Zur Optimierung des Durchsatzes eines *Computational Grids* kann Stapelverarbeitung eingesetzt werden, das große parallele Programme als Batch-Jobs verwaltet und einsetzt. Diese Optimierungstechnik bietet sich vor allem dann an, wenn eine hohe Zahl von Programmläufen mit nur geringfügig veränderten Parametersätzen, sogenannte Parameterstudien, gefahren werden sollen.

On-Demand Anwendungen – Für Probleme und Anwender, die nur temporär große Ressourcenmengen brauchen, bietet sich der netzbasierte Zugriff auf zusätzliche Re-

chenleistung als ökonomische Alternative zu Neuanschaffungen an. Das Spektrum der zumietbaren Ressourcen reicht dabei von der bloßen Rechenressource allein bis hin zur Miete einer kompletten Anwendung, die eine Problembeschreibung entgegen nimmt und das Ergebnis zurückliefert, wie es in dem universitären Projekt NetSolve [CD98] bereits realisiert wurde.

Tele-Kooperation – Rechen-Netze unterstützen durch ihre Infrastruktur-Eigenschaften, wie Verfügbarkeit, Konsistenz und Zuverlässigkeit, die entfernte, interdisziplinäre Kooperation sehr weit verteilter Wissenschaftler und Mitarbeiter. Beispielanwendungen der Tele-Kooperation sind Workflow Management, Global Engineering Village, Tele-Medizin und rechnergestützte Ausbildung.

Datenintensive Anwendungen – Ebenfalls aufgrund ihrer Verfügbarkeit eignen sich Rechen-Netze zur Sammlung und Auswertung sehr großer, verteilter Datenmengen. Datenintensive Anwendungen erfordern die Interoperabilität zwischen *Legacy*-Anwendungen und Rechennetz-Diensten für Zugriff sowie die Visualisierung, um die anfallenden Datenmengen im Bereich von Tera- bis Pentabyte in den Griff zu bekommen. Beispiele solcher Anwendungen sind Finanzmarktanalysen, Auswertung astronomischer Daten [SET] und die Klimasimulation.

Tele-Immersion – Unter Tele-Immersion werden verteilte Anwendungen virtueller Realität verstanden, um verteilte Anwendungen und Gruppenarbeit zu implementieren und zu visualisieren. Simulationen und Datensätze sind dabei vom Anzeigegerät des Benutzers räumlich und logisch getrennt. Anwendungen solcher Tele-Immersion sind beispielsweise interaktive Simulationen zur Visualisierung wissenschaftlicher Daten, zum Training und zur Ausbildung sowie zum Industrie- und Architekturdesign.

2.3.2 Rechennetz-Gemeinschaften

Da Rechen-Netze in erster Linie eine Infrastruktur zur gemeinsamen Nutzung von (Rechen-) Ressourcen sind, muß der Ertrag einer gemeinsamen Nutzung die Kosten, die eine teilnehmende Institution zu tragen hat, übertreffen. Daher wird von Foster und Kesselman eine Entwicklung von individuellen Rechen-Netzen vorausgesehen, die Gemeinschaften mit einigen wenigen Interessen und Ressourcen miteinander verbinden und auf die Erfüllung dieser Benutzerinteressen spezialisiert sind [FK98b]. Beispiele für solche Rechennetz-Gemeinschaften sind nationale, private, virtuelle und öffentliche Rechen-Netze.

Nationale Rechen-Netze – Eine der naheliegendsten Rechennetz-Gemeinschaften sind nationale Rechen-Netze, die Landesvertreter, Planer und Wissenschaftler vereinigen, die mit nationalen Aufgaben wie der Katastrophenhilfe, Landesverteidigung und langfristiger Forschung und Entwicklung beschäftigt sind. In einer solchen Rechennetz-Gemeinschaft ist eine kleine Zahl sehr leistungsfähiger Superrechner gegenüber einer Vielzahl kleiner Rechensysteme zu bevorzugen, da ein nationales Netz zwei Hauptaufgaben dient: als Rechenleistungsreserve zur Berechnung oder Simulation großer Probleme in Krisenzeiten und als Substrat zur Tele-Kooperation bei wissenschaftlichen und umwelttechnischen Problemen auf nationaler Ebene.

Private Rechen-Netze – Am unteren Leistungsspektrum von Rechen-Netzen stehen die privaten Netze als Zusammenschluß einer kleinen Anzahl von Höchstleistungstrechnern und einigen Workstations, Datenbanken und Standard (*off the shelf*) Komponenten, die unter einer gemeinsamen, zentralen Verwaltung stehen. Als Beispiel für ein privates Grid sei ein Rechennetz genannt, das Ärzte, Administratoren, Geräte und Ressourcen städtischer Krankenhäuser zu einer medizinischen Infrastruktur zusammenfaßt. Durch die Koppelung von Superrechnern, Workstations, medizinischen Bilddatenbanken und anderen Spezialinstrumente wird eine ganze Reihe neuer Verfahren wie computergestützte Diagnoseverfahren, datenbankgestützte Suche nach symptomatischen Ähnlichkeiten, Anwendungen der Tele-Medizin und langfristige computergestützte Forschung und Entwicklung ermöglicht.

Virtuelle Rechen-Netze – In jüngster Zeit ist der Begriff der virtuellen Firma aufgekommen, der im Kontext betrieblicher Organisationen für Unternehmensstrukturen verwendet wird, die ihren jeweiligen vorgesehenen Zweck erfüllen, ohne dabei mit den sonst üblichen Eigenschaften "normaler" Unternehmen versehen zu sein, wie fester Firmengebäude, sozialversicherungspflichtiger Arbeitsverhältnisse oder entsprechender Produktionsstätten. Das Grundkapital einer virtuellen Firma ist hauptsächlich immaterieller Natur, wie zum Beispiel Marken, Patente, Software oder *Know-How*, und das in erster Linie durch soziale und logistische Verknüpfungen zusammengehalten wird. In der Praxis sind verschiedene Stufen virtueller Unternehmungen beobachtbar, die sich durch ihre Komplexität unterscheiden.

1. Virtuelle Arbeitsplätze sind Arbeitsplätze, die nicht mehr fest im Unternehmen installiert sind, sondern in der Privatwohnung, unterwegs im Außendienst oder in mobilen Arbeitsplätzen, und die die Erbringung der Arbeitsleistung an anderen Orten ermöglichen.
2. Virtuelle Teams sind durch Infrastrukturen zur Fernkontrolle (Telematik, vgl. Definition 3.24) verknüpfte Arbeitsgruppen, die projektbezogen und temporär gemeinsam arbeiten. Dabei werden Zeitvorteile ausgenutzt, die durch bessere Synchronisation und Parallelisierung entstehen. Mitglieder können nach Qualifikation und Eignung anstelle von räumlicher Nähe ausgesucht werden.
3. Virtuelle Unternehmen bestehen aus einem Netzwerk von Einzelunternehmen oder Einzelpersonengesellschaften, die sich nach außen als eine logische Einheit darstellen, ohne durch einen dauerhaften rechtlichen Rahmen miteinander verbunden zu sein. Auch virtuelle Unternehmen sind nicht durch lokale Randbedingungen in ihrer Teilnehmerauswahl eingeschränkt. Allen genannten Stufen ist der starke Einsatz von Telemedien und Telematiken wie (Mobil-) Telefon und Videokonferenz, email und WWW Technologien zur Überbrückung der Entfernungen und Aufrechterhaltung einer nahtlosen und einheitlichen Fassade.

Ausgehend vom eben Gesagten ist ein virtuelles Netz eine Rechennetz-Gemeinschaft, die durch ihre dynamisch veränderlichen Teilnehmer geprägt ist und hauptsächlich zur Kommunikation und Koordination seiner Mitglieder verwendet wird. Die Abgrenzung zu den virtuellen privaten Netzwerken (*Virtual Private Networks*) ist dabei

fließend. Unter letzteren werden im allgemeinen private Lokal- und Weitverkehrsnetzwerke (LAN, WAN) verstanden, die mittels Authentifizierung und Verschlüsselung sicher über unsichere öffentliche Netze wie dem Internet geroutet werden.

Öffentliche Rechen-Netze – Die größte mögliche Rechennetz-Gemeinschaft ist keine Gemeinschaft im eigentlichen Sinne, sondern ein offener, transparenter Markt von Rechen-Dienstleistungen. Ein solches öffentliches Netz spricht eine enorme Zahl von Benutzern an, die keine besondere Gemeinsamkeit mehr verbindet – außer daß sie mit bestimmten Gewinnabsichten in einem solchen Markt auftreten, sei es als Anbieter oder als Nachfrager von Rechen-Dienstleistungen oder reinen Ressourcen wie CPU, Speicher oder Netzwerk. Ob ein öffentliches Grid sich zu einem symmetrischen System, also ob Teilnehmer genauso häufig in beiden Rollen auftreten, oder eher zu einem asymmetrischen System entwickelt mit relativ wenigen festen Anbietern und vielen Nachfragern, bleibt abzuwarten.

2.3.3 Anwender von Rechen-Netzen

Die Bedeutung verteilter Rechen-Netze für zukünftige internationale Forschung und Entwicklung ergibt sich aus der Tatsache, daß sie nicht alleiniges Territorium weniger Wissenschaftler und Ingenieure sind, sondern sehr vielen Anwendern Nutzen und Problemlösungen bieten. Insofern unterscheiden sich die Anwender von Rechen-Netzen kaum von anderen populären Rechen- und Programmierumgebungen.

Endanwender – Die meisten Benutzer von Rechen-Netzen werden, wie auch schon heute, Endanwender sein, die nur angebotene Software verwenden und mit der Softwareentwicklung nicht in Berührung kommen wollen. Diese Software muß hohe Anforderungen an Benutzerfreundlichkeit und Intuitivität erfüllen. Insbesondere darf die Verteilung nur bei Telekooperations-Anwendungen sichtbar und muß sonst transparent sein. Der laienhafte Benutzer ist also von den Aspekten der Parallelität und Verteilung abgeschirmt, fortgeschrittene Anwender behalten jedoch die Möglichkeit, zur Optimierung "unter" die Abstraktionsschicht zu sehen.

Anwendungsentwickler – Anwendungsentwickler haben in der Vergangenheit stets viel Energie und Zeit durch die Arbeit auf niedrigem Abstraktionsniveau, wie zum Beispiel auf Socket und TCP/IP Ebene, verloren. Erst durch die Bereitstellung von standardisierten Programmiermodellen und -umgebungen wie PVM und MPI wurde die Voraussetzung für eine relativ komfortable, plattformunabhängige Programmierung verteilter Systeme geschaffen. Solche Programmiermodelle und -umgebungen sollten daher auf Rechen-Netze übertragen und weiter verbessert werden.

Werkzeugentwickler – Grundlage von Programmierumgebungen für Rechen-Netze könnte ein Toolkit wie das *Globus Toolkit* sein, das Basisdienste für Kommunikation, Datenzugriff, zum Ressourcen-Management, zur Sicherheit und Fehlertoleranz zur Verfügung stellt und in [FK97] näher beschrieben ist. Solche Toolkits und Basisdienste werden von Werkzeugentwicklern implementiert und fortlaufend weiterentwickelt.

Rechennetz-Entwickler – Bei den Rechennetz-Entwicklern handelt es sich um die kleinste Gruppe von Nutzern/Entwicklern. Ihre Aufgabe ist die Bereitstellung und Wartung der Basisdienste.

Systemadministratoren – Systemadministratoren binden lokale Ressourcen in das Rechennetz ein und vermitteln zwischen den Interessen der lokalen Netzbetreiber und den übrigen Netzanwendern durch die Einhaltung und Überwachung einer bestimmten lokalen Ressourcenpolitik innerhalb ihres Hoheitsbereichs (*administrative domain*). Darüberhinaus sind sie für die Abrechnung der abgegebenen Ressourcen zuständig.

2.3.4 Rechennetz-Architektur

Die Entwicklung einer Rechennetz-Architektur umfaßt sowohl die Berücksichtigung von bestehenden Infrastrukturen und *Legacy* Systemen – also "Altlasten", die zwar technisch veraltet und unzureichend sind, aber aufgrund ihrer weiten Verbreitung nicht entsorgt werden können – als auch die Planung von Hard- und Softwaretechnologien, die auf diesen Komponenten aufbauen und diese für die Zwecke einer neuen Rechen-Infrastruktur fortentwickeln. Im Folgenden findet sich hierzu eine Analyse der bestehenden Komponenten von Weitverkehrsnetzen, ein Anforderungskatalog an darauf aufsetzende Dienste sowie die nähere Beschreibung einer konkreten Implementierung eines Rechennetz Testbettes.

Rechen-Plattformen – Als Rechen-Plattformen in einem Rechennetz kommt eine Vielzahl von Endsystemen in Frage, von Standard (*off-the-shelf*) Rechnern bis hin zu extrem leistungsfähigen Spezialcomputern und Verbindungstechnologien wie Myrinet [NCR+95], Servernet [DW97], *Virtual Interface Architecture* (VIA) [FK98b] oder Netzwerkspeicherlösungen (*network attached storage*). Darüberhinaus können auch sogenannte zusammengesetzte Elemente (*Simple Composite Elements*, SCE) [JA97] eingesetzt werden, die aus Basiselementen zusammengesetzt sind und um spezielle Hard- und Software ergänzt worden sind, um besondere Funktionalitäten, wie zum Beispiel hochverfügbare Namensdienste, zu implementieren. Abhängig vom Grad der Leistungsfähigkeit und der Skalierung lassen sich Rechenknoten nach folgenden Gesichtspunkten einordnen: Einzelrechner, Cluster, Intra-Netzwerke, Weitverkehrsnetze und Internet.

Einzelrechner sind zum Beispiel PCs, Workstations oder symmetrische Multiprozessor-Systeme, aber auch Web-Terminals – also Terminals, die nur über genügend Rechenleistung zum Betrieb eines Browsers verfügen – fallen in diese Gruppe.

Bei Clustern handelt es sich um homogene, gekoppelte Systeme mit verteiltem Speicher wie beispielsweise NOWs oder MPPs. Die Systeme sind jedoch mehr oder weniger homogen, abhängig davon, bezüglich welcher Komponente gleichartige Eigenschaften vorhanden sind, wie zum Beispiel Prozessor, Hauptspeicher, Betriebssystem oder Anwendung oder alle gleichzeitig. Die Erfahrung zeigt überdies, daß selbst homogene Systeme oft, beispielsweise aufgrund unterschiedlicher Konfiguration, relativ inhomogenes Verhalten zeigen.

Legacy-Software – Auf Softwareseite sind auf Einzelrechnern gebräuchliche PC-Betriebssysteme und verschiedene UNIX-Derivate im Einsatz. Außer TCP/IP stehen

dabei im allgemeinen keine weiteren Protokolle oder Programmiermodelle zur Anbindung an Rechen-Netze zur Verfügung. Als Programmiermodell für Cluster von PCs findet meist explizite Nachrichtenvermittlung (*message passing*) Anwendung, jedoch ist auch verteilter gemeinsamer Speicher nicht unüblich.
In LANs kommen eine Reihe verschiedener *Middleware* Systeme zur Anwendung, um zwischen der vorherrschenden Architekturheterogenität zu vermitteln. Hierzu zählen *Distributed Computing Environment* (DCE) [OHE96], *Message Oriented Middleware* (MOM), *Distributed Transaction Processing Monitors* (DTPM), *Object Request Brokern* (ORB) wie CORBA [Gro98] und DCOM [Mic98b] oder anderer Middleware wie der entfernte Methodenaufruf von Java [Sun97].
In Weitverkehrsnetzen wie dem Internet existieren mehrere Ansätze zur Realisierung verteilter Anwendungen. Die nahe liegendste Vorgehensweise ist sicherlich die Verwendung von allgegenwärtigen Standardtechnologien in einer 3-Schichten-Architektur. Mittels eines Web-Browser basierten Frontends (Schicht 1) können Nutzer über eine Middleware (Schicht 2) auf Serveranwendungen zugreifen (Schicht 3), die bestimmte Dienste anbieten. Dieses Modell wird von den Web-Computing Infrastrukturen Javelin [CCN+97] und Popcorn [RN98] implementiert. Das Standardmodell kann durch objektorientierte Methoden in Schicht 1 und 2 aufgewertet werden, um die Benutzer- und Programmierfreundlichkeit zu erhöhen, wie es im Meta-Computing System Legion der Fall ist.

Hochleistungs-Scheduler – Rechen-Netze sind als Plattform für Anwendungen vorgesehen, die extrem hohe Leistung benötigen. Die Vergangenheit hat gezeigt, daß zur Erreichung solch hoher Rechenleistung das Scheduling verteilter heterogener Ressourcen von entscheidender Bedeutung ist. Da sich Nutzer und Anwendungen bei der Sicherstellung von hoher Leistung nicht auf den Rechen-Netzen zugrunde liegende Infrastruktur verlassen können, sind Hochleistungsscheduler als Erweiterung dieser Infrastruktur notwendig, die mittels verschiedener Techniken wie Leistungsvorhersage, Ressourcenauswahl, Planung und Scheduling für eine optimale Leistung sorgen.

Sicherheit, Abrechnung und Qualitätssicherung – Neben den klassischen Sicherheitsanforderungen wie dem Schutz vor Eindringlingen in das System, ergeben sich in Rechen-Netzen neue Gesichtspunkte wie die Überprüfung der Leistungsfähigkeit und Zuverlässigkeit von Service Anbietern und der Herkunft fremden Codes oder die Auflösung widersprüchlicher Sicherheitsrichtlinien. Hierfür sind mindestens fünf verschiedene Sicherheitsdienste nötig, Authentifizierung, Authorisierung, Qualitätssicherung, Abrechnung und Aufzeichnung. Für einige dieser Anforderungen existieren bereits geeignete Technologien und Systeme wie PGP zur Verschlüsselung von Dateien und Nachrichten, *Firewalls*, Kerberos [BT94] oder Netzverschlüsselungstechnologien wie IPSec. Dennoch werden mindestens drei Sicherheitsaspekte von aktuellen Technologien nicht abgedeckt. Dazu gehören sichere gruppenübergreifende Kommunikation über unsichere Netze und administrative Grenzen hinweg, zuverlässige Abrechnungsverfahren, um den Ressourcenverbrauch steuern und begrenzen zu können sowie flexible Methoden zur Definition und Überwachung konfliktfreier, lokaler und globaler Sicherheitspolitiken von Rechen-Netzen.

Netzwerkinfrastruktur – Als Netzwerkinfrastruktur wird das sehr heterogene Verbindungsnetz bezeichnet, das aus Lokal- und Weitverkehrsnetzen, Transportprotokollen und -diensten und Routing-Algorithmen besteht. In vielfacher Hinsicht kann das Internet als prototypische Implementierung eines Rechen-Netzes angesehen werden, das auf den Zugriff auf Dateien und Kontrolldiensten ausgerichtet ist anstatt auf Rechendienste. Dennoch kann eine Implementierung eines Rechen-Netzes viel vom heutigen Internet lernen, jedoch sind dabei einige grundlegende Erweiterungen und Verbesserungen notwendig. Dazu gehören eine breitere Palette von Servern mit diversifizierteren Dienstleistungen, verbesserte Benutzungsoberflächen, ein auf Diensten basierender Namensraum anstelle von URLs sowie neue und effizientere Netzwerktechnologien und -protokolle.

Quality of Service Netzwerkprotokolle – Traditionelle Netzwerkprotokolle unterstützen nur den Transport auf Bit-Ebene. Von der Kommunikation in Rechen-Netzen wird jedoch zunehmend die Unterstützung der in der jeweiligen Anwendung vorherrschenden Kommunikationsart erwartet. Die Netzwerkbedürfnisse von Rechennetz-Anwendungen lassen sich bezüglich vier Netzwerk-Klassen einordnen: Datentransportprotokolle wie es beispielsweise mit TCP vorliegt, Streaming-Protokolle zur Übertragung von Audio, Video und Multimedia, Gruppenkommunikations-Protokolle sowie Transportprotokolle, die verteilte Objekte unterstützen. Weiterhin ist die Überwachung und Einhaltung der Konsistenz verteilter replizierter Objekte durch die Netzwerkschicht wünschenswert. In der jüngsten Zeit wurden geeignete Protokolle entwickelt, die für den Einsatz in Rechen-Netzen geeignet erscheinen, wie zum Beispiel *Xpress Transfer Protocol* (XTP) [WBA95] für Cluster-weit arbeitende verteilte und parallele Anwendungen, *Scalable Reliable Multicast* (SRM) [SVC+97] und das *InterGroup Protocol* (IGP) für virtuelle kollaborative Umgebungen, das *Real-Time Transport Protocol* (RTP) [HSRV96] und das *Resource reSerVation Protocol* (RSVP) [LSD+93] für Anwendungen der Tele-Immersion, die CORBA Architektur und das *Internet Inter-ORB Protocol* (IIOP) [OMG] und schließlich Java RMI [Sun97].

Hochleistungsrechnen für die Massen – Allgegenwärtige Softwaretechnologien wie CORBA, DCOM, JavaBeans und andere Web- und Netzwerkkonzepte können zum Aufbau eines dreischichtigen Rechen-Netzes verwendet werden, in dem Anwendungs-Server zwischen speziellen Basisdiensten und Web basierten Frontends vermitteln. Die Drei-Schichten-Architektur ist daher geeignet, ein Rechennetz auf Basis von allgegenwärtigen Massentechnologien aufzubauen.

2.3.5 Testumgebungen

Testumgebungen bieten die Möglichkeit, neue Technologien und Anwendungen zu entwickeln, zu evaluieren und weiter zu verbessern ohne bestehende Systeme durch unausgereifte Software zu gefährden. Zum einen ist der Aufbau und Test neuer Technologien nur integriert, also als Ganzes möglich, ein testweiser Betrieb einzelner Komponenten alleine innerhalb etablierter, traditioneller Technologien ist nicht möglich. Durch Testbetten

lassen sich darüberhinaus die teilnehmenden Benutzergruppen besser überwachen und auswerten als in realen Infrastrukturen. Schließlich lassen sich durch Testbetten die Risiken bei der Einführung neuer Technologien durch die regionalen und finanzielle Schranken limitieren.

Die Globus Infrastruktur Unter den Ansätzen zur Konstruktion eines Rechen-Netzes befindet sich die als Globus *Toolkit* bezeichnete Sammlung von Basisdiensten für ein Rechen-Netz, auf denen eine Reihe höherer Programmiermodelle aufsetzen können. Dieses *Toolkit* ist als Gruppe orthogonaler Komponenten und Dienste wie Ressourcenmanagement, Kommunikations-, Informations- und Sicherheitsdiensten mit der Ausrichtung konzipiert, heterogene Ressourcen mit Hilfe von Schnittstellen zu verwalten zu können anstelle sie vor dem Anwender zu verstecken.

Im Gegensatz zu den bisher vorgestellten Infrastrukturen muß Globus [FK97] eher als spezielle experimentelle Ausführungsumgebung für die *Grand Challenges* der 90er Jahre betrachtet werden. Supercomputer, Spezialdatenbanken, wissenschaftliche Instrumente und unkonventionelle Visualisierungssysteme werden dabei über Hochgeschwindigkeitsnetze zu einem Substrat verknüpft, das internationalen Wissenschaftlern die verteilte, kooperative und interdisziplinäre Arbeit an aktuellen wissenschaftlichen Projekten ermöglicht. Bei Globus handelt es sich um die Testumgebung der Schöpfer des *Computational Grid* Begriffs (Abschnitt 2.3). Zum Nachweis der Machbarkeit der Thesen ihres Buches wurde von den Autoren die Arbeit an diesem Meta-Computing System begonnen, dessen Grundprinzip die Erweiterung lokal nicht genügend oder gar nicht vorhandener Ressourcen darstellt. Um die an eine Meta-Computing Infrastruktur gestellten Anforderungen hinsichtlich Transparenz bezüglich Heterogenität, Verteiltheit und Unzuverlässigkeit zu erfüllen, setzt Globus dabei auf ein vielschichtiges Abstraktionsmodell zur Implementierung ihrer Systemsoftware, Werkzeuge und Anwendungen.
Basisdienste stehen im *Globus Toolkit* zur Verfügung und bieten jeweils definierte Schnittstellen zum Zugriff von Diensten einer höheren, abstrakteren Schicht an. Zu den wichtigsten Basisdiensten gehören Ressourcenfindung, Ressourcenzugriff, Monitoring, Authentifizierung auf Basis von Kerberos und SSL (*Secure Socket Layer*) sowie Kommunikations-Basisdienste wie *Message-Passing, Distributed Shared Memory* und *Remote Procedure Call*. Diese bilden die Grundlage der darauf aufsetzenden höheren Dienste. Zu diesen Diensten gehören u.a. verschiedene Programmiermodelle und parallele Implementierungen diverser Programmiersprachen wie MPI, parallel C++, FortranM.

Als Testumgebungen für Globus Anwendungen dienen I-WAY [DFP+96] und GUSTO (*Globus ubiquitious supercomputing testbed* [FK97]). Bei I-WAY handelt es sich um ein staatlich gefördertes Universitätsprojekt, das Supercomputer aus 17 nordamerikanischen Städten über ein Hochgeschwindigkeitsnetz zu einem Meta-Computer verbindet, während GUSTO mehrere Tausend kleinerer Workstations von mehreren Dutzend Universitäten und Instituten über eine ATM Verbindung koppelt.

2.4 Web-Computing

Neben dem Meta-Computing, das die Koppelung dedizierter Hochleistungshardware zu virtuellen Superrechnern zum Ziel hat, gibt es auch Anstrengungen das enorme Poten-

tial brachliegender Rechner und anderer Ressourcen von Weitverkehrsnetzen, die meist unter fremder administrativer Kontrolle liegen und sich opportunistisch verhalten, durch Web-Computing Umgebungen zu erschließen. Im folgenden Abschnitt finden sich die wichtigsten Vertreter solcher Infrastruktur vorgestellt und bewertet. Unter den ausgereifteren Projekten, die Sicherheits- und Plattform-Eigenschaften der plattformunabhängigen Programmiersprache Java nutzen, gehören – neben einer Reihe anderer, weniger weit entwickelter Ansätze ([Van97],[Far96]) – Javelin [CCN+97] und SuperWeb

2.4.1 SuperWeb

SuperWeb [AISS97] ist ein hybrides System aus Meta-Computing Infrastruktur und elektronischem Markt. Das System bietet einen durch einen *Broker* implementierten elektronischen Marktplatz an, der zwischen sogenannten *Hosts*, die Rechenzeit anbieten, und *Clients*, die die Ressourcen nachfragen, hinsichtlich der gewünschten Eigenschaften der Ressourcen (*Quality of Service*) vermittelt. Zu diesen Anforderungen gehören zeitliche Fristen, Plattenkapazität, Bandbreite, Fließkommagenauigkeit, usw.

Von seinen Entwicklern wird SuperWeb als Infrastruktur für weltweites Rechnen im WWW eingeordnet, der hauptsächliche Einsatzort wird aber zunächst in geschlossenen Intranets innerhalb großer Organisationen und Institute mit einer großen Anzahl homogener oder auch heterogener Rechner gesehen. Neben der Unterstützung paralleler und verteilter Anwendungen mit geringer Kommunikations-Anforderung wie der Primzahlenzerlegung, Monte-Carlo und grob-granularen Simulationen soll SuperWeb auch als Server für das *Network Computing* Anwendung finden.

Als typische Web-Computing-Infrastruktur stützt sich SuperWeb auf offene Sprachen und Werkzeuge, die auf praktisch allen *Web*-Terminals als vorhanden angesehen werden können wie WWW Browser, Java und einer JVM. Diese erlauben das Versenden von Tasks ohne direkten Zugang zu fremden Rechensystemen über eine Kennung bzw. ohne Betriebssystem-Modifikationen. Durch Verwendung solcher als sicher und ausgereift geltenden Hilfsmittel soll eine sehr breite Schar von Teilnehmern gewonnen werden. Über deren Eigenschaften hinausgehende Sicherheitsanforderungen, wie zum Beispiel Korrektheitsnachweise der zurückgelieferten Ergebnisse oder die Verschlüsselung sensitiver Daten oder Algorithmen, sollen durch weitere Verfahren abgedeckt werden. Zu diesen gehören statistische Prüfungen und einfache Gegenproben der Ergebnisse zur Entdeckung von falscher oder vorgetäuschter Programmausführung oder die verschlüsselte Ausführung von sensitiven Algorithmen und die Aufteilung sensitiver Daten in Segmente, die klein genug sind, um einzeln für einen unberechtigten Dritten keine sinnvolle Informationen mehr zu enthalten.

SuperWeb's Marktmodell basiert auf Mikrowährungen, um auch den Handel mit sehr kleinen Ressourcenmengen rentabel und den Tausch von Ressourcen unterschiedlicher Art, Zeit und Menge zwischen *Clients*, *Hosts* und dem *Broker*, deren Rollen nicht fixiert sind durchaus wechseln kann, zu ermöglichen. Dieses mikroökonomische Handelsmodell soll gleichzeitig auch die Tätigkeit des *Brokers* als Mittelsmann mit einer Kommissionsgebühr entlohnen, der damit seine fixen und variablen Kosten zur Aufrechterhaltung dieses Dienstes bestreiten kann, und als Anreiz für Anbieter zur Verfügungstellung von Rechenzeit dient.

System	Ressourcen	Schnittst.	skaliert	Fehlertoleranz	Teilnahme	Legacy
ATLAS	dediziert	geschlossen	ja	Atomarität	Reg.+ Inst.	nein
Charlotte	dediziert	geschlossen	nein	Eager Sched.	Inst.	nein
Globus	dediziert	geschlossen	ja	Basis-Dienst	Reg. + Inst.	nein
Legion	dediziert	geschlossen	ja	Basis-Dienst	Reg. + Inst.	ja
Javelin	opportun.	offen	nein	Work Stealing	Registr.	ja
SuperWeb	opportun.	offen	nein	Basis-Dienst	Registr.	ja

Tabelle 2.1 Klassifikation von Meta- und Web-Computing Infrastrukturen

2.5 Klassifikation und Bewertung der Systeme

Die Vorstellung der verschiedenen Infrastrukturen zum weltweit verteilten und parallelen Rechnen in den vorausgegangenen Abschnitten hat verdeutlicht, daß sich die Systeme wesentlich in der Art der verwendeten Ressourcen und der Schnittstellen und Protokolle unterscheiden. Die im Anschluß vorgenommene Betrachtung der Meta-, Web- und Rechennetz-Rahmenwerke erfolgt daher sowohl unter allgemeinen Aspekten, aber insbesondere auch unter den Gesichtspunkten der verwendeten Ressourcen und Schnittstellen. Die Ergebnisse der Betrachtungen der vorangegangenen Abschnitte finden sich zusammengefaßt in Tabelle 2.1 und bilden Grundlage des folgenden näheren Vergleichs der Systeme.

Vom Gesichtspunkt der Art der verwendeten Ressourcen und Schnittstellen betrachtet, zeichnet sich eine deutliche Trennung zwischen Meta- und Web-Computing Infrastrukturen ab, wie sie auch Grundlage der in diesem Kapitel erarbeiteten Nomenklatur darstellt. Beide Denkansätze unternehmen den Versuch, höhere Rechenleistung zu erzielen und dabei die Verteilung soweit wie möglich oder technisch sinnvoll zu verbergen. Dieser Anspruch wird jedoch auf sehr unterschiedliche Weise umgesetzt. Meta-Computing Systeme wie ATLAS, Globus oder Legion sind ohne Ausnahme durch die Verwendung dedizierter Ressourcen und geschlossener Schnittstellen zur Systemsoftware gekennzeichnet, das heißt sie erzielen die Erhöhung der Rechenleistung durch Erhöhung der *Qualität* der beteiligten Komponenten im Hinblick auf Leistungsfähigkeit und Zuverlässigkeit, auch wenn dadurch nur wenige Teilnehmer den Qualitätsanspruch erfüllen und zur Teilnahme zugelassen werden können.
Ganz im Gegensatz dazu steigern Web-basierte verteilte Infrastrukturen die Rechenleistung durch die *Quantität* der beteiligten Komponenten, auch wenn diese unter Umständen mit mangelnder Qualität einhergeht. Dazu muß eine verteilte Rechen-Infrastruktur eine große Anzahl von Architekturen, Protokollen und Ressourcen von möglicherweise geringerer Qualität unterstützen, um einen möglichst hohen Teilnehmerkreis zu erschließen. Die geringere Qualität kann sich dabei in geringerer Leistung, Effizienz, Zuverlässigkeit oder Sicherheit äußern. So zeichnen sich Web-Computing Infrastrukturen wie Javelin und SuperWeb durch opportunistische Ressourcen und offene Schnittstellen zur Systemsoftware aus, die oft zum Großteil aus weit verbreiteten WWW Softwaretechnologien wie Browser, Java und HTTP besteht. Insofern sind zur Sammlung freier Ressourcen Technologien und Protokolle des Web-Computings besser geeignet als solche

des Meta-Computings.

Im Hinblick auf die breite Unterstützung vieler Architekturen und Protokolle zeichnet sich wiederum eine Trennung zwischen Meta- und Web-Computing Umgebungen ab, allerdings weit weniger deutlich. So unterstützen alle Web-Computing Infrastrukturen *Legacy* Programme, also nicht speziell für das jeweilige Programmiermodell des Systems entwickelte Software, während kein Meta-Computing Rahmenwerk mit Ausnahme von Legion derartige Programme unterstützt. Im Gegenteil, es ist die Installation zusätzlicher Software notwendig, um die Funktionalität der Systeme nutzen zu können, eine Tatsache, die den Teilnehmerkreis stark beschränkt. Alle Umgebungen erfordern die Registrierung von Benutzern bevor eine Teilnahme am System möglich ist, ein Erfordernis, das mit Sicherheit *nicht* einer skalierenden Infrastruktur Paroli bieten kann.

Obwohl zumindest ein Teil der Infrastrukturen ihr Leistungspotential aus einer hohen Teilnehmerzahl schöpft, stellen einige der Web basierten Umgebungen kein skalierendes System zur Verfügung, so zum Beispiel Charlotte, Javelin und SuperWeb. Diese Infrastrukturen konzentrieren ihre Verwaltungsfunktionalität auf einen einzigen Server und stellen keine Optionen zur Verteilung dieses Dienstes zur Verfügung. Ausgleichend verfügt jede Rechenumgebung über ausreichende Funktionalität zur Fehlertoleranz und zu einem einfachen clientseitigem Lastausgleich.

Web-Computing-Infrastrukturen wurden bereits als Mittel zur Sammlung und Nutzung von freien, opportunistische Ressourcen in Weitverkehrsnetzen identifiziert. Aufgrund der vorliegenden Klassifikation wird jedoch klar, daß es den vorgestellten Web-Computing Systemen an Eigenschaften und Verfahren mangelt, um die für eine hohe akkumulierte Rechenleistung notwendige Quantität von Ressourcen zu erreichen. Diese ist zum Ausgleich ihrer geringeren Effizienz gegenüber Meta-Computing Systemen notwendig. Die eingeschränkte Skalierbarkeit und notwendige Registrierung und Installation einerseits und fehlende Anreize zur Teilnahme andererseits stehen diesen Systemen zur Erreichung dieses Ziels eindeutig im Wege.

Zu den notwendigen Verfahren gehören Maßnahmen, die zur Senkung der Transaktionskosten beitragen, sowohl in Hinsicht auf die Organisation und Verwaltung des Systems als auch hinsichtlich der Teilnahme an der Infrastruktur. Um eine hohe akkumulierte Rechenleistung anzusammeln, sind Größenordnungen mehr Anbieter als Nachfrager von Ressourcen notwendig, ein Umstand, der sich in einer asymmetrischen Modellierung der Ressourcensammlung und ihrer Teilnehmer äußern muß. Durch die sehr hohe Teilnehmeranzahl ist die Führung individueller Benutzerkonten, die eingerichtet, verwaltet und ausbezahlt werden müssen, nicht praktikabel. Aufgrund der breiteren Orientierung eines solchen Systems, an laienhafte Benutzer, ist eine automatische, installationslose und transparente Teilnahme besser geeignet, um skalierbar eine hohe Zahl von Teilnehmern zuzulassen. Auch läßt sich in einem sehr großen System eine allwissenden Steuerkomponente nicht mehr aufrecht erhalten. Stattdessen ist eine dezentrale Organisation mit Hilfe von marktbasierten Zuteilungsverfahren, die Gegenstand des nächsten Kapitels sind, besser geeignet, sehr verteilte, heterogene und umfangreiche Ressourcen, Teilnehmer und Strukturen zu verwalten.

Kapitel 3

Elektronische Märkte

Die freie Marktwirtschaft ist eines der leistungsfähigsten und verbreitetesten Verfahren zur effizienten Allokation von knappen Ressourcen. Unter diesem Gesichtspunkt betrachtet macht es daher durchaus Sinn, sich mit ihrer Anwendung in der Informatik, die sich zu einem großen Teil auch mit der effizienten und fairen Zuteilung von Ressourcen beschäftigt, zu befassen. Ausgehend von einer Einführung in die traditionelle prioritätenbasierte Ressourcenzuteilung, dem *Scheduling* in geschlossenen Prozeßsystemen, stellt das folgende Kapitel eine Einleitung und Begriffsfindung in die marktbasierte Ressourcenzuteilung zur Verfügung und führt grundlegende Eigenschaften und Begriffe elektronischer Märkte ein. Es beschreibt verbreitete marktbasierte Zuteilungsverfahren und stellt einige experimentelle und konkrete Implementierungen elektronischer Märkte und ihrer Eigenschaften vor. Eine Diskussion und Klassifikation dieser Märkte im Hinblick auf die Ziele dieser Arbeit schließt dieses Kapitel ab.

3.1 Prioritätenbasierte Ressourcenzuteilung

Die Ursprünge herkömmlicher in der Informatik eingesetzter Ressourcen-Zuteilungsverfahren reichen zurück in die Zeit der Zentralrechner (*Mainframes*), die eine große Anzahl angeschlossener Dateneingabegeräte (*Terminals*) ohne eigene Verarbeitungskapazität bedienten. Ausgehend von einfachen Stapelverarbeitungssystemen, die auf Lochkarten vorliegende Programme abarbeiteten und die Ergebnisse an den Auftraggeber zustellten, über einfache *Time-Sharing* Systeme, die mehrere Benutzer quasi gleichzeitig bedienten, entwickelten sich die Ressourcen-Verwaltungssysteme hin zu Schedulern aktueller Arbeitsplatz- und Parallelrechner, wie sie heute allgemein üblich sind.

Allen diesen Ressourcen-Zuteilungsverfahren ist gemein, daß sie eine zentrale, ausgezeichnete Komponente eines geschlossenen Systems darstellen, die über sämtliche Informationen des Systems verfügen muß, um seine Ressourcen sinnvoll zuteilen zu können. Die Geschlossenheit und der Zentralismus sind wechselweise notwendig, da eine zentrale Instanz eines offenen Systems nicht allwissend sein kann. Genauso ist die Synchronisierung verteilter Instanzen zur Konsistenzerhaltung selbst in einem geschlossenen System sehr aufwendig. Ziel der Ressourcenzuteilung ist stets ein effizienter Einsatz der verfügbaren Ressourcen, wobei Effizienz im Gegensatz zu anderen wissenschaftlichen Disziplinen im Sinne einer Maximierung des Systemdurchsatzes verstanden wird. Dabei herrscht eine asymmetrische Sichtweise auf die zu vergebenen Ressourcen, das heißt eine Ressource, nämlich die des Prozessors, ist ausgezeichnet, während andere Ressourcen wie Hauptspei-

cher und Plattenkapazität gar nicht oder nur untergeordnet bei der Zuteilung berücksichtigt werden.

3.1.1 Prozeßsysteme

Ressourcenzuteilungs-Verfahren beschäftigen sich mit der Auflösung nebeläufiger, konkurrierender Zugriffe auf exklusive Ressourcen innerhalb eines Prozeßsystems. Daher ist die vor einer Beschäftigung mit den Zuteilungsverfahren selbst eine Einführung und Klärung der Begriffe eines Prozeßsystems und Nebenläufigkeit notwendig.

Ein Prozeßsystem besitzt keine Nebenläufigkeit, sie muß serialisiert werden oder über nicht-transparente Konzepte wie beispielsweise Seiteneffekte nachgebildet werden. Dazu bieten übliche Betriebssysteme verschiedene Mechanismen und Nomenklaturen zur Behandlung von Nebenläufigkeit an.

Traditioneller Ansatz ist die Verwendung mehrerer Prozesse: Fast alle Server Anwendungen besitzen einen sog. *listener process*, der bei Eintreffen einer Anfrage eines Clients mittels `fork()` einen neuen Prozeß erzeugt, der sich ausschließlich um die Anfrage kümmert und sich dann beendet. Diese Strategie führt auch in Einprozessor-Systemen zu erhöhtem Durchsatz, da ein einzelner Prozess oft durch Ein- oder Ausgabe blockiert ist und keine neuen Anfragen entgegennehmen kann. `fork()` ist jedoch ein sehr teurer Systemaufruf, der allein die Erzeugung der kooperierenden Prozesse sehr aufwendig macht. Da außerdem jeder Prozess über seinen eigenen, getrennten Adreßraum verfügt, müssen kooperierende Prozeße über Interprozeßmechanismen wie den Austausch von Nachrichten (*message passing*) oder gemeinsamer Speicher (*shared memory*) kommunizieren. Diese Kommunikation erhöht den notwendigen *Overhead* von nebenläufigen Prozeßen weiter. Außerdem sind die nebenläufigen Steuerflüsse mehrerer Prozesse nicht im Quelltext imperativer Sprachen sichtbar, sondern müssen durch Seiteneffekte sequentieller Sprachmittel nachgebildet werden [Sve95].

Sequentielle Programmiersprachen bieten in der Regel nur einen abstrakten Steuerfluß an. Es existieren in diesen Sprachen keinerlei Sprachkonstrukte um Nebenläufigkeit auszudrücken obwohl in heutigen System immer häufiger mehrere Prozessoren anzutreffen sind, die koordiniert werden müssen, wie zum Beispiel bei der Behandlung asynchroner Ausnahmen oder bei ereignisgesteuerten Anwendungen. Um beispielsweise die Nebenläufigkeit eines Timesharing-Prozess- oder Thread-Systems formulieren zu können, muß auf fehlerträchtige und unklare Konstrukte wie Prozeduren mit bestimmten Seiteneffekten wie `fork()` zurückgegriffen werden. Diese Seiteneffekte bilden die nebenläufigen Steuerflüsse für die Anwendung nach. Die Abbildung nebenläufiger Prozesse durch sequentielle syntaktische Sprachmittel ist jedoch nicht unproblematisch. Es können keinerlei Angaben über die relative Geschwindigkeit und die Kontextwechsel der einzelnen Steuerflüsse gemacht werden. Außerdem ist die Nebenläufigkeit im Programmtext nicht sichtbar und einzelne Aktivitäten können den entsprechenden Prozessen nicht zugeordnet werden, was im Hinblick auf eine gute Les- und Wartbarkeit des Quellcodes wünschenswert wäre.

Nebenläufigkeit und Parallelismus

Nebenläufigkeit kann von der System- oder Anwendungsschicht bereitgestellt werden. Das Betriebssystem kann systemweite Nebenläufigkeit durch mehrere Steuerflüsse (*hot threads*)

[Gra95] innerhalb eines Prozesses zur Verfügung stellen, denen unabhängig voneinander Betriebsmittel, insbesondere Rechenzeit, zugewiesen werden. Auch bei Uniprozesssoren kann Nebenläufigkeit zu verbessertem Durchsatz führen, da blockierende Instruktionen durch andere Flüße verdeckt werden.
Anwendungen können Nebenläufigkeit über Benutzer-Bibliotheken bereitstellen. Solche Anwendungsflüsse (*cold threads*) [Gra95] bleiben dem Betriebssystem verborgen und müssen von der Bibliothek selbst verwaltet und ihnen Betriebsmittel zugeteilt werden. Sie stellen keinen Parallelismus zur Verfügung, da Anwendungsflüsse nicht parallel fortschreiten können. Sie bieten jedoch eine natürliches Sprachmittel zur Formulierung nebenläufiger Anwendungen.

Im folgenden werden schrittweise aufeinander aufbauende Möglichkeiten beschrieben, um nebenläufige Steuerflüsse auch in einer imperativen Programmiersprache zu formulieren und sichtbar zu machen [Sve95]. Die Prozeßstruktur muß dabei auf die syntaktischen Mittel der Sprache abgebildet werden, wobei eine kleine Menge von Hilfskonstrukten mit Seiteneffekten unumgänglich ist. Gleichzeitig kann damit auch ein genauerer Einblick in die interne Arbeitsweise eines (Leichtgewichts-) Prozeßsystems geworfen werden. Die einzelnen Schritte auf dem Weg zu einem Prozeßsystem sind dabei im Folgenden:

1. Koroutinen mit/ohne internem Scheduling – Ausgehend von gewöhnlichen Prozeduren werden diese über eine besondere Ablauf-Umgebung als Koroutinen abgearbeitet, das heißt die Prozeduren werden nicht vollständig abgearbeitet und kehren dann zu ihrer Aufrufstelle zurück, sondern der Steuerfluß wird an einem beliebigen Punkt innerhalb der Prozedur an eine andere Prozedur abgegeben, wo die Bearbeitung fortgesetzt wird. Koroutinen sind durch zwei Besonderheiten eng miteinander gekoppelt. Der Umschaltpunkt sowie die Nachfolge-Routine sind explizit im Programmtext festgelegt. Durch Parametrisierung bzw. Auflösung dieser Statik dieser beiden Eigenschaften entstehen aus den Koroutinen unabhängige Prozesse. Verlagert man die Auswahl der Nachfolge-Routine aus dem Quellcode in eine zentrale Instanz, dem *Scheduler*, spricht man von Koroutinen mit internem Scheduling, sonst von Koroutinen ohne Scheduling.

2. Koroutinen mit externem Scheduling – Verlagert man weiterhin den Scheduler von der abgeschirmten Koroutinenschicht in die Anwendungsschicht, entsteht ein Koroutinen-System mit externem Scheduler. Ein externer Scheduler besitzt zur Auswahl der nächsten Koroutine sämtliche, u.U. sehr dynamische Informationen der Anwendungsschicht und kann deshalb intelligentere Strategien zur Ressourcenzuteilung durchsetzen als ein interner Scheduler.

3. präemptive Koroutinen / Threads – Entkoppelt man auch die Umschaltoperation von den Koroutinen und veranlaßt die Umschaltoperation durch einen äußeren, privilegierten Prozeß, entsteht ein System unabhängiger Prozeße, das als (Leichtgewichts-) Prozeßsystem bezeichnet wird, da es durch einen gemeinsamen Adreßraum verbunden ist.

Definition 3.1 (Steuerfluß) *Ein Steuerfluß (*Thread of Control*), oder einfach nur Fluß (*Thread*), bezeichnet eine Folge ausführbarer Schritte mit einem Einstiegspunkt und einem Endpunkt.*

Zu einem Steuerfluß gehören alle Ressourcen, die zum Fortschritt der Ausführung notwendig sind, wie der Keller, Programm- und Datenbereich sowie der Registersatz. Diese Ressourcen bezeichnet man auch als Kontext des Steuerflusses. Zur Erzeugung von Programmen mit einem Steuerfluß steht der Systemaufruf `fork()` zur Verfügung.

Definition 3.2 (Kontext) *Zum Kontext eines Steuerflusses gehören alle Ressourcen, die zu seiner Ausführung notwendig sind, wie Keller-, Programm- und Datenbereich sowie der Registersatz.*

Definition 3.3 (Prozeß) *Unter einem Prozeß wird üblicherweise die schrittweise Abarbeitung der Instruktionen eines Programms durch eine Ausführungseinheit verstanden.*

Definition 3.4 (Ausführungseinheit) *Ein Ausführungseinheit ist ein aktives Element, das durch Operationen (Instruktionen) Zustandsänderungen einer (virtuellen) Maschine ausführt.*

Die Ausführungseinheiten müssen nicht unbedingt Prozessoren sein, sondern können auch Steuerflüsse des Betriebssystems sein. Nach der erfolgreichen Ausführung des aktuellen Befehls wird der Befehlszähler inkrementiert (Sequenz) oder durch eine Instruktion auf einen anderen Wert gesetzt (Sprung), um den nächsten Befehl im Hauptspeicher zu adressieren. Ausführungseinheiten können reale Prozessoren oder aber auch virtualisierte Prozessoren sein, falls mehrere Prozesse nebeneinander vorhanden sind. Virtuelle Prozessoren unterstützen mehrere, nebenläufige Steuerflüsse.

Definition 3.5 (Nebenläufigkeit) *Nebenläufigkeit ist der maximale Grad gleichzeitiger Ausführung, den eine Anwendung bei unbegrenzter Anzahl von Prozessoren zuläßt. Sie ist von der Anwendung bzw. von der Anzahl Steuerflüsse, die gleichzeitig vorhanden sind, abhängig.*

Definition 3.6 (Parallelismus) *Parallelismus ist der Grad tatsächlicher, in der Praxis beobachtbarer gleichzeitiger Ausführung und ist damit von der Anzahl verfügbarer Aktivitätsträger begrenzt.*

3.1.2 Ressourcenzuteilung

Nachdem ein Steuerfluß mittels `fork()` oder `thread_create()` erzeugt wurde, wird ihm der Zustand rechenbereit zugewiesen und er wird in eine Warteschlange rechenbereiter Steuerflüsse eingetragen. Ein mittels `fork()` erzeugter Prozeß besitzt mindestens einen Steuerfluß, der bei `main()` beginnt und nach `exit()` endet. Falls der Prozeß eine Leichtgewichtsprozess-Bibliothek verwendet und entsprechende `thread_create()` Aufrufe ausführt, enthält der Prozeß sogar mehr als einen Steuerfluß. In jedem Fall wird mindestens ein rechenbereiter Steuerfluß in die Rechen-Schlange eingetragen.

In einem Koroutinen-System erfolgt die Übergabe des Kontrollflusses von einer Koroutine durch expliziten Aufruf der nächste Koroutine mit einem Aufruf im Quellcode [1]. Diese enge Koppelung zwischen den Koroutinen eines Systems läßt sich durch die Delegation der expliziten Angabe der nächsten Koroutine an eine zentrale Koroutine, den *Scheduler* teilweise aufheben.

[1] mittels der Prozedur `transfer()`

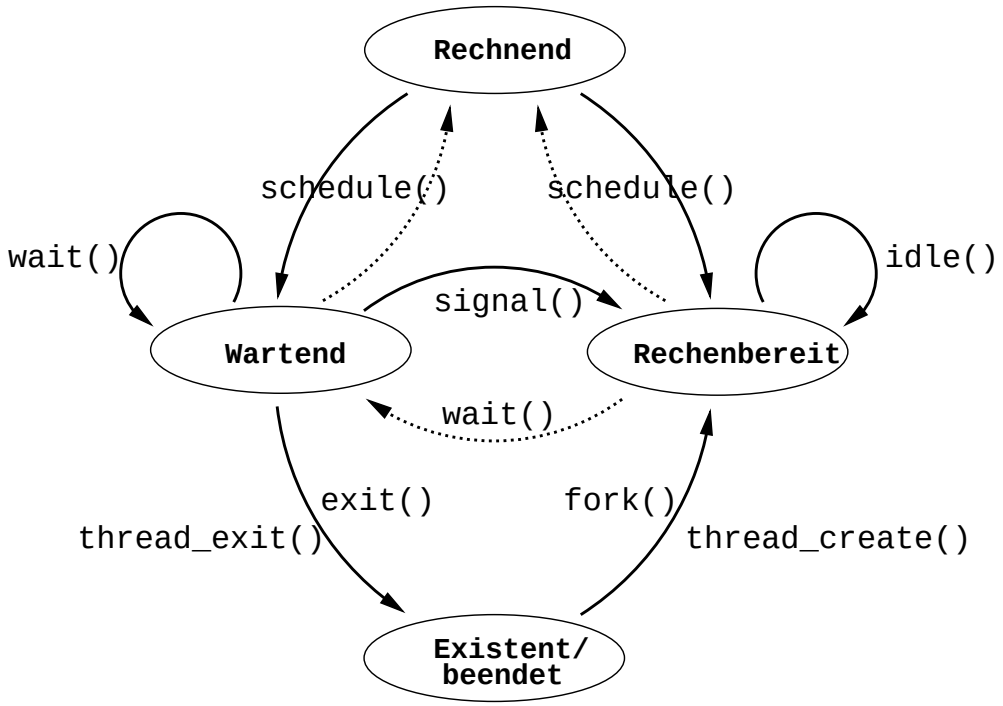

Abbildung 3.1 Die fünf Zustände von Steuerflüssen

Definition 3.7 (Ressourcenzuteilung) *Unter Ressourcenzuteilung (*Scheduling*) versteht man die Zuteilung der lauffähigen Steuerflüsse an die vorhandenen Ausführungseinheiten durch den* Scheduler*.*

Definition 3.8 (Scheduler) *Allgemein ist ein Scheduler eine Instanz zur Zuteilung von u.U. exklusiven Betriebsmitteln (Ressourcen) an konkurrierende (Leichtgewichts-) Prozesse. Die im Zusammenhang mit Scheduling oft implizit angenommene Ressource eines exklusiven Prozessors stellt dabei nur einen Spezialfall dar.*

Im Wesentlichen existieren drei Zustände eines Steuerflusses, *wartend*, *rechenbereit* und *rechnend*, wie in Abbildung 3.1 illustriert. Daneben gibt es noch zwei Zustände, die jeder Steuerfluß nur einmal innerhalb seiner Lebensspanne annehmen und aus denen er nicht zurückkehren kann, *existent* und *beendet*.

Zu jeder Zeit gibt es innerhalb eines (Leichtgewichts-) Prozeßsystems eine Gruppe rechenbereiter Steuerflüsse, die in einer globalen Warteschlange – oder im Fall eines Multiprozessors in mehreren Warteschlangen – verwaltet werden. Eine solche Warteschlange besteht aus einem Vektor von Zeigern auf jeweils eine verkettete Liste von rechenbereiten Steuerflüssen [2]. Die Warteschlangen nehmen sowohl Steuerflüsse des Benutzer- als auch des Kernbereiches auf. Damit jederzeit mindestens ein Steuerfluß rechenbereit ist, existiert ein sogenannter *Idle-thread*, der die niedrigste mögliche Priorität besitzt. Neben Anwendungs- und Kernsteuerflüssen existieren noch sogenannte Unterbrechungssteuerflüsse mit allerhöchster Priorität, die zur Behandlung von Unterbrechungen vorgesehen sind.

[2] eine verkettete Liste des Typs `kthread_t`

Im Gegensatz zu den Zuständen wartend, rechenbereit und rechnend existiert keine Schlange für den Zustand existent. Ein Steuerfluß gelangt vom Zustand existent in den Zustand rechenbereit mittels `fork()` oder `thread_create()`. Ist ein Steuerfluß beendet, wird er in die sogenannte `thread_deathrow` Schlange eingetragen. Wenn neue Steuerflüsse erzeugt werden, werden die Datenstrukturen beendeter Steuerflüsse aus dieser Schlange wiederverwendet, ein Vorgang, der effizienter als die Initialisierung einer neuen Datenstruktur ist.

Definition 3.9 (Kontextwechsel) *Das Umschalten zwischen den Zuständen eines Steuerflusses bezeichnet man als Kontextwechsel.*

Definition 3.10 (Präemption) *Unter einer Präemption (vorzeitige Beendigung) versteht man einen Kontextwechsel, der zur Beendigung des gerade aktiven und zur Bearbeitung eines anderen Steuerflusses führt.*

Der zuletzt aktive Steuerfluß wird in die Liste rechenbereiter Steuerflüsse eingetragen. Eine Preemption erfolgt wenn die dem Steuerfluß zugeordnete Zeitscheibe abgelaufen oder falls ein Steuerfluß höherer Priorität rechenbereit geworden ist.

Definition 3.11 (Unterbrechung) *Eine Unterbrechung ist im Gegensatz zur Präemption eine kurzzeitige Unterbrechung des gerade aktiven Steuerflusses zur Erledigung einer höher priorisierten Anfrage.*

Eine solche Anfrage kann von der Hardware wie die Fertigstellung von Ein-/Ausgabe oder Netzwerkverkehr oder von der Software wie die Zustellung eines Signals stammen. Der davor aktive Steuerfluß verliert dabei nicht die Kontrolle sondern ihm werden anschließend wieder Betriebsmittel zugeteilt.

Im allgemeinen sind Unterbrechungen weniger kritisch als Präemptionen, da der zuletzt aktive Steuerfluß nicht ausgelagert werden muß, so daß der Kontext schnell wieder zur Verfügung steht. Probleme treten bei der Präemption immer dann auf, wenn Sorge getragen werden muß, einen aktiven Steuerfluß nicht innerhalb eines kritischen Bereiches vorzeitig zu beenden. In nicht echtzeitfähigen Betriebssystemen sind nur Benutzersteuerflüsse vorzeitig beendbar [Val96]. Solange ein Steuerfluß im Benutzermodus abläuft, kann er von jedem höher priorisierten Fluß vorzeitig beendet werden. Da in einem solchen Betriebssystem ein Steuerfluß des Kerns nicht vorzeitig beendet werden kann, muß bei der Implementierung der Betriebsmittelvergabe nicht so viel Aufwand zur Behandlung nebenläufiger Zugriffe auf den Kern betrieben werden. Andererseits kann ein solches Betriebssystem unter Umständen nicht kurzfristig auf Betriebsmittelanforderungen der Anwendung reagieren, da es durch einen hochpriorisierten Steuerfluß des Kerns beschäftigt ist, und daher nicht echtzeitfähig ist.

3.1.3 Prioritäten

Jedem Steuerfluß ist eine Priorität zugeordnet, die anwendungsspezifisch ist und in der Regel an Kind-Steuerflüsse weiter vererbt wird, also weitere Flüsse, die vom Steuerfluß während seiner Laufzeit erzeugt worden sind. Die Priorität wird als Einsprung-Index in den Vektor von Warteschlangen verwendet, die jeweils eine Reihe von Steuerflüssen

verwalten. Da zu einem beliebigen Zeitpunkt mehr als ein Steuerfluß auf ein bestimmtes Ereignis, wie zum Beispiel die Prozessorzuteilung, warten kann, sind die Steuerflüsse in einer Reihenfolge verkettet.

Definition 3.12 (Rechenschlange) *Die verkettete Liste der Steuerflüsse im Zustand* rechenbereit *wird als Rechenschlange bezeichnet.*

Eine Rechenschlange ist demnach eine verkettete Liste rechenbereiter Steuerflüsse gleicher Priorität. Falls eine CPU unbeschäftigt ist, wählt der Scheduler den ersten Steuerfluß aus der höchsten nicht leeren Rechenschlange, der per Definition der Datenstruktur der am höchsten priorisierte rechenbereite Steuerfluß ist. In der Regel wird ein Round-Robin Verfahren zur Verwaltung von Steuerflüssen gleicher Priorität verwendet, daß heißt der nächste aktive Steuerfluß wird vom Anfang der verketteten Liste entnommen, ihm Betriebsmittel zugewiesen und nach Ablauf seiner Zeitscheibe an das Ende der Liste angefügt.

Definition 3.13 (Warteschlange) *Die verkettete Liste der Steuerflüsse im Zustand* wartend *wird als Warteschlange bezeichnet.*

Ein gerade blockierter Steuerfluß wird an das Ende der Schlange angefügt und vom Anfang entnommen, falls das Ereignis eingetroffen ist. Dieser Algorithmus implementiert ein First-In-First-Out (FIFO) Schema. Dieses Schema gilt nicht, falls ein *broadcast* Ereignis eintrifft. In diesem Fall werden alle auf dieses Ereignis wartenden Steuerflüsse aus der Schlange entfernt. Gewöhnlicherweise betrifft ein eintreffendes Ereignis nur eine einzige der Warteschlangen.

Prioritätenklassen Es werden verschiedene Scheduling-Klassen unterschieden, nämlich Zeitscheiben-Klassen der Anwendungsschicht, System-Schedulingklassen sowie unter Umständen Echtzeit-Klassen [Val96]. Jede dieser Klassen besitzt verschiedene Parameter, die durch die klassenspezifischen Funktionen der entsprechenden Klassen implementiert werden. Ein Steuerfluß gehört zunächst der Schedulingklasse seines Vaters an, kann diese aber über Systemaufrufe [3] selbst verändern.

Zeitscheiben-Klassen – Die Zeitscheiben-Klasse ist die gebräuchlichste der Schedulingklassen. Alle Anwendungssteuerflüsse mit Ausnahme von Echtzeit-Anwendungen laufen in dieser Klasse. Die Zeitscheiben-Klasse verwendet Zeitscheiben oder -fenster, um die Ressource Prozessor(en) gleichmäßig auf die Steuerflüsse zu verteilen. Ein Steuerfluß erhält für die Dauer dieses Fensters – zwischen 20 und 200 Millisekunden – die Kontrolle über einen Prozessor. Durch einen anschließenden Präemption wird ihm die Kontrolle wieder entzogen. Die Länge der Zeitscheibe ist abhängig von der Priorität des Steuerflusses und ändert sich mit jedem Kontextwechsel. Die Standardeinstellungen können mit einem Systemaufruf [4] beeinflußt werden.

[3] durch den Aufruf `priocntl()`

[4] `dispadmin()`

System-Schedulingklassen – Die System-Schedulingklasse dient zur Ressourcenzuteilung an Steuerflüsse des Kerns. Diese Flüsse besitzen eine feste, unveränderliche Priorität und werden nicht über Zeitscheiben unterbrochen. Eine Anwendung kann keine Steuerflüsse in diese Klasse bringen, sondern ein Fluß gelangt automatisch in die System-Schedulingklasse, wenn er einen Systemdienst aufruft. Wenn ein Steuerfluß also einmal in die System-Schedulingklasse gelangt ist, dann läuft er solange, bis er blockiert, vorzeitig beendet oder ganz beendet ist.

Echtzeit-Klasse – Die Echtzeit-Klasse ist wieder ein Zeitscheiben-gesteuertes Ressourcen-Zuteilungsverfahren, allerdings mit statischen Prioritäten. Ein Echtzeit-Steuerfluß behält also nach dem Ablauf seiner Zeitscheibe seine Priorität bei und kann durch diese beim Wettkampf um die nächste Zeitscheibe Vorteile gegenüber Anwendungssteuerflüssen ausspielen. Geht er aus diesem Kampf als am höchsten priorisierter Steuerfluß hervor, kann er für die Dauer eines weiteren Zeitfensters voranschreiten.

Wartet ein Steuerfluß auf ein synchrones Ereignis wie zum Beispiel eine Sperre oder eine Nachricht, wird er aus dem Rechenschlangen-Vektor entnommen und in die Warteschlange eingetragen. Wie die Rechenschlange ist die Warteschlange eine verkettete Liste wartender Steuerflüsse gleicher Priorität. Wenn ein synchrones Ereignis eintrifft, wird der wartende Steuerfluß mit höchster Priorität aufgeweckt, das heißt wieder in die Rechenschlange eingetragen.

Die vorzeitige und zeitgemässe Beendigung eines Steuerflusses der Zeitscheiben-Klasse geschieht mittels der Funktion `preempt()`. `preempt()` wird aufgerufen, falls ein Steuerfluß höherer Priorität rechenbereit geworden ist oder falls die Zeitscheibe des gerade aktiven Steuerflusses abgelaufen ist. Die Funktion ruft wiederum eine Schedulingklassenspezifische Präemptionsroutine auf, deren Funktionalität stets die gleiche ist. Der zuletzt aktive Fluß wird abhängig von der Schedulingklasse und Priorität in die entsprechende Position der entsprechenden Rechenbereit-Schlange eingefügt. Falls die Präemption infolge eines Ablaufs der Zeitscheibe erfolgte, wird der Steuerfluß an das Ende seiner Rechenbereit-Schlange gesetzt, wurde jedoch ein höher priorisierter Fluß rechenbereit, gelangt er an den Anfang dieser Schlange. Dadurch steht ihm später wieder eine volle Zeitscheibe vor allen anderen Steuerflüssen gleicher Priorität zur Verfügung.

Die Präemption der System-Schedulingklasse ist einfacher, da keine Erschöpfungen der Zeitscheiben oder Anpassungen der Prioritäten vorkommen. Ein System-Steuerfluß läuft solange bis er die Kontrolle wieder abgibt. Dann wird er einfach an den Beginn seiner Rechenbereit-Schlange gesetzt. Die Funktion `preempt()` der Echtzeit-Klasse muß wieder überprüfen, ob ein Zeitscheiben-Ablauf oder eine Unterbrechung vorliegt. Im letzten Fall muß die Dauer der Zeitscheibe angepaßt werden und der Steuerfluß an das Ende seiner Rechenbereit-Schlange gesetzt werden. Sonst gelangt der Steuerfluß zurück an den Anfang dieser Schlange.

3.1.4 Unterbrechungsbehandlung

Bisher wurde die Erzeugung , die Ausführung sowie die vorzeitige Beendigung von Steuerflüssen betrachtet, die alle innerhalb einer Schedulingklasse abliefen. Es gibt allerdings auch eine wichtige Reihe von Steuerflüssen, die nicht innerhalb einer Schedulingklasse ablaufen, die Unterbrechungs-Steuerflüsse [Val96].

Tritt eine Unterbrechung auf und wird an eine CPU ausgeliefert, wird der dort laufende Steuerfluß unterbrochen, das heißt ihm wird durch einen Kontextwechsel die Kontrolle über die CPU entzogen und er wird an den Anfang der Rechenbereit-Schlange seiner CPU gesetzt. Er wird also solange unterbrochen, bis die Unterbrechung behandelt und beendet wurde. Ein Wechsel der CPU eines unterbrochenen Flußes ist nicht möglich. Aus einer Unterbrechungsbehandlungs-Tabelle weiß die CPU, welche Unterbrechungsbehandlungs-Routine für die Unterbrechung zuständig ist. Die Behandlungsroutine ist ebenfalls ein Steuerfluß mit allen bisher betrachteten Eigenschaften eines Flusses. Dazu gehören insbesondere der Kontext mit eigenem Keller, Registersatz und Programmzähler.

Da die Unterbrechungsbehandlungs-Routine ein Steuerfluß ist, ist die Ressourcenzuteilung an eine solche Routine nach dem bisher gesagten denkbar einfach nachzuvollziehen. Wenn eine Unterbrechung eintrifft, wird der ihm zugeordnete Steuerfluß mit hoher Priorität zur weiteren Bearbeitung ausgewählt. Unterbrechungen besitzen stets die höchst mögliche Priorität innerhalb des (Leichtgewichts-) Prozessystems. Eine Unterbrechungsbehandlung kann nur von einer höher priorisierten Unterbrechung vorzeitig unterbrochen werden.

Sehr eilige Unterbrechungsbehandlungen sind oft nicht unterbrechbar und laufen bis zur Beendigung ab. Diese Steuerflüsse besitzen keinen Kontext, weil sie nicht unterbrechbar sind und sind daher keine (Leichtgewichts-) Prozesse.

3.1.5 Schedulingstrategien

Das Verhalten einer Schedulingklassen ist in erster Linie durch ihre *dispatcher*-Tabelle gegeben. Die Schedulingklasse und die anfängliche Priorität eines Steuerflusses wird durch den Vater sowie bestimmte Systemaufrufe [5] festgelegt, alle übrigen Parameter wie die Länge der Zeitscheibe, die Priorität nach einer Präemption usw. werden durch Nachschlagen innerhalb einer klassenspezifischen *dispatcher* Tabelle ermittelt. Durch Manipulation dieser Tabelle kann Einfluß auf das Schedulingverhalten genommen werden. Dazu existieren entsprechende Systemaufrufe[6] .

Beim Blick auf die Standard-Zeitscheiben-Tabelle fällt auf, daß hohe Prioritäten ein kurze Zeitscheibe und niedrige Prioritäten eher lange Zeitscheiben besitzen. Das führt dazu, daß nicht-interaktive, rechenintensive Steuerflüsse niedrigere Prioritäten erhalten, um interaktive Anwendungen, die höhere Prioritäten besitzen, nicht zu verdrängen.

Strategien

Die einfachste Schedulingstrategie stellt die zyklische Auswahl aus der Liste aller Steuerflüsse im Zustand `rechenbereit` dar. Im Folgenden stellen wir einige andere Freiheitsgrade bei der Wahl der geeigneten Schedulingstrategie vor.

Deterministische Zuteilung – Implementiert der Scheduler eine feste Umschaltreihenfolge, beispielsweise mit Hilfe einer Umschalttabelle (Gantt-Tabelle) [Sve95], so ergibt sich wie bei einem Koroutinen-System eine streng deterministische Zuteilungsreihenfolge.

[5] `priocntl()`
[6] `dispadmin()`

Nichtdeterministische Zuteilung – Besitzt der Scheduler nur einen Teil oder gar keine Informationen über die Steuerflüsse, kann das Ergebnis einer Auswahl und Zuteilung nicht mehr im voraus bestimmt werden. Um jedoch trotz der fehlenden Informationen gewisse Mindestanforderungen an die Zuteilung wie Fairneß garantieren zu können, stehen eine Reihe von Heuristiken für die nichtdeterministische Zuteilung zur Verfügung.

1. Lineares Zuteilen – Beim linearen Zuteilen sind alle bereiten Steuerflüsse bei der Auswahl gleichberechtigt. Verbreitete Verfahren zur Auswahl eines Steuerflusses aus der Menge der bereiten Flüsse sind
   (a) Round Robin – Aus der Liste der bereiten Steuerflüsse wird zyklisch die nächste ausgewählt.
   (b) First-In-First-Out (FIFO) – Der am längsten wartende Steuerfluß bekommt den Zuschlag. Diese Strategie wird üblicherweise mit statischen Prioritäten eingesetzt.
2. Verhältniszuteilung – Bei einer statischen Anzahl von Steuerflüssen lassen sich über Zuteilungshäufigkeiten unterschiedlicher Ablaufgeschwindigkeiten der Threads realisieren.
3. Prioritäten – Jedem Steuerflüß wird eine Priorität zugeordnet, die statisch über die gesamte Laufzeit des Programms konstant bleibt oder aber sich dynamisch an die Erfordernisse des Laufzeitsystem anpaßt. Die am höchsten priorisierte Thread der Bereit-Liste wird zugeteilt. Falls mehr lauffähige Threads gleicher Priorität als Ausführungseinheiten vorhanden sind, muß ein möglicher Konflikt durch die Schedulingstrategie (FIFO, Round-Robin) aufgelöst werden.
   (a) statische Priorität
   Die einmal zugewiesene Priorität bleibt über die gesamte Lebenszeit eines Threads bestehen.
   (b) dynamische Priorität
   Die Priorität wird von der Bibliothek angepaßt. Beispielsweise kann ein Thread bei Eintreffen eines Ereignisses, auf das er gewartet hat, höher priorisiert werden und niedriger priorisiert, falls seine Zeitscheibe abgelaufen ist.
4. Hierarchisches Zuteilen – Im Gegensatz zum linearen Zuteilen werden beim hierarchischen Zuteilen verschiedene Ebenen der Steuerflüsse berücksichtigt. Erzeugende und erzeugte Steuerflüsse, d.h. Eltern- und Kind-Flüsse sind bei der Zuteilung des Kontrollflusses nicht gleichberechtigt. Der Scheduler wählt einen der Steuerflüsse der untersten Hierarchieebene zur Ausführung aus. Besitzt dieser ein Kind, so wählt der Scheduler entweder das Elternteil oder aber eines der Kinder aus. Besitzt der Kind-Fluß ebenfalls ein Kind, pflanzt sich der Auswahlprozess rekursiv fort, bis entweder die Eltern-Fluß ein zweites Mal oder aber ein kinderloser Steuerfluß ausgewählt wurde.
5. Zuteilungsklassen – Beim Erzeugen eines Steuerflusses wird dieser statisch in eine von mehreren Prioritätsklassen eingetragen. Die Zuteilung erfolgt in der höchstpriorisierten Klasse solange es dort rechenbereite Flüsse gibt. Falls nicht

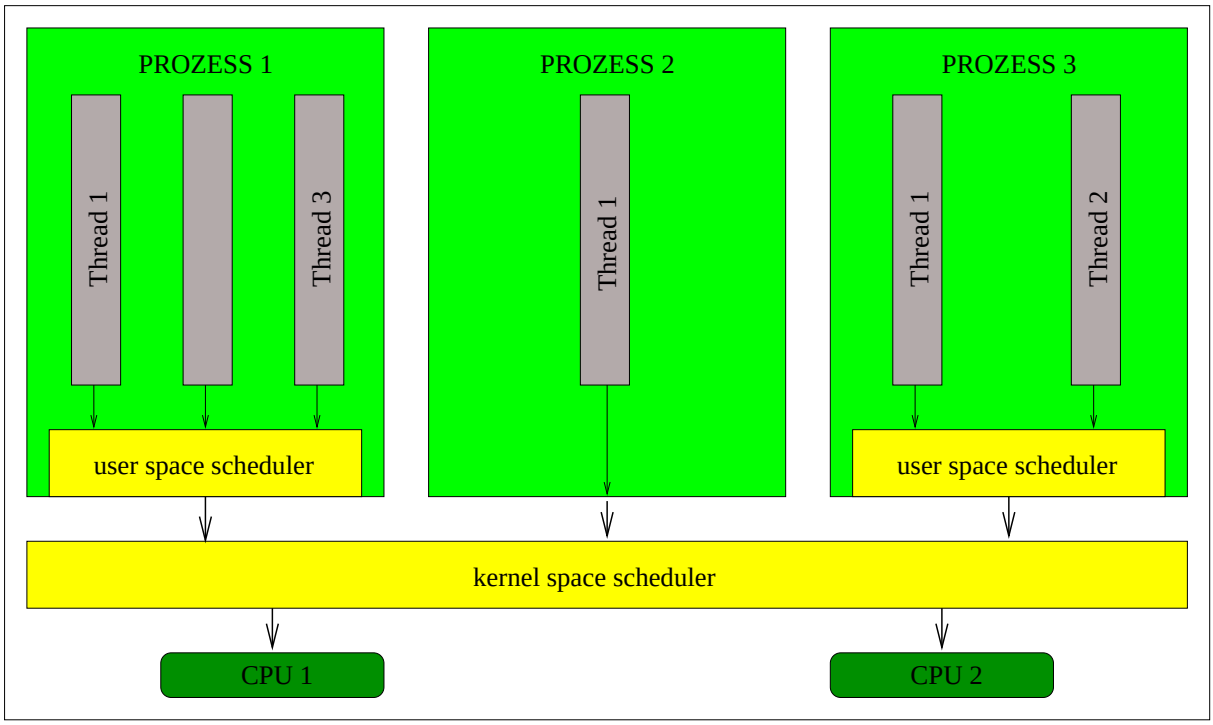

Abbildung 3.2 *User level* Thread-Implementierung mit mehrstufiger Zuteilung

kommen die Flüsse niedriger priorisierter Klassen zum Zug. Mit dieser, u.a. in der *light weight processes* (LWP) Bibliothek von SunOS [Gra95] verwendeten, Strategie können sehr kurze Reaktionszeiten auf bestimmte Ereignisse realisiert werden. Steuerflüsse hochpriorisierter Klassen zur Behandlung von wichtigen Ausnahmen sind dabei in der Regel suspendiert, so daß niedriger priorisierte Flüsse zugeteilt werden. Wird jedoch ein suspendierter Steuerfluß durch ein Ereignis aufgeweckt, wird er vom Scheduler sofort zugeteilt und kann die Ausnahme umgehend behandeln.

6. Mehrstufige Zuteilung – Abweichend von den bisherigen Heuristiken kann man anstelle eines einzelnen Schedulers auch zwei oder mehrere Scheduler über disjunkten Teilmengen von Flüssen betrachten. Jeder Scheduler wählt aus seiner Teilmenge einen Steuerfluß aus und ein übergeordneter *Meta*-Scheduler wählt aus den Vorschlägen einen Fluß aus. Dieser Ansatz läßt sich hierarchisch beliebig tief schachteln. Mehrstufiges Scheduling findet man bei Thread-Implementierungen als *User-level*-Bibliothek. Der Thread-Scheduler wählt innerhalb der Bibliothek einen rechenbereiten Thread aus und schlägt diesen dem Betriebssystem-Scheduler zur Zuteilung vor, wie in Abbildung 3.2 illustriert.

7. Externe Zuteilung – Werden zum intelligenten Umschalten mehr Informationen – insbesondere aus der Anwendungsschicht – benötigt als in den Verwaltungs-Datenstrukturen vorhanden, kann die Prozedur `schedule()` in die Anwendungsschicht ausgelagert werden. Ein externer Scheduler ist Bestandteil der Anwendungsschicht und verfügt damit über sämtliche Informationen dieser Schicht. Bei jeder Umschaltoperation wird nun der Steuerfluß an die Umschal-

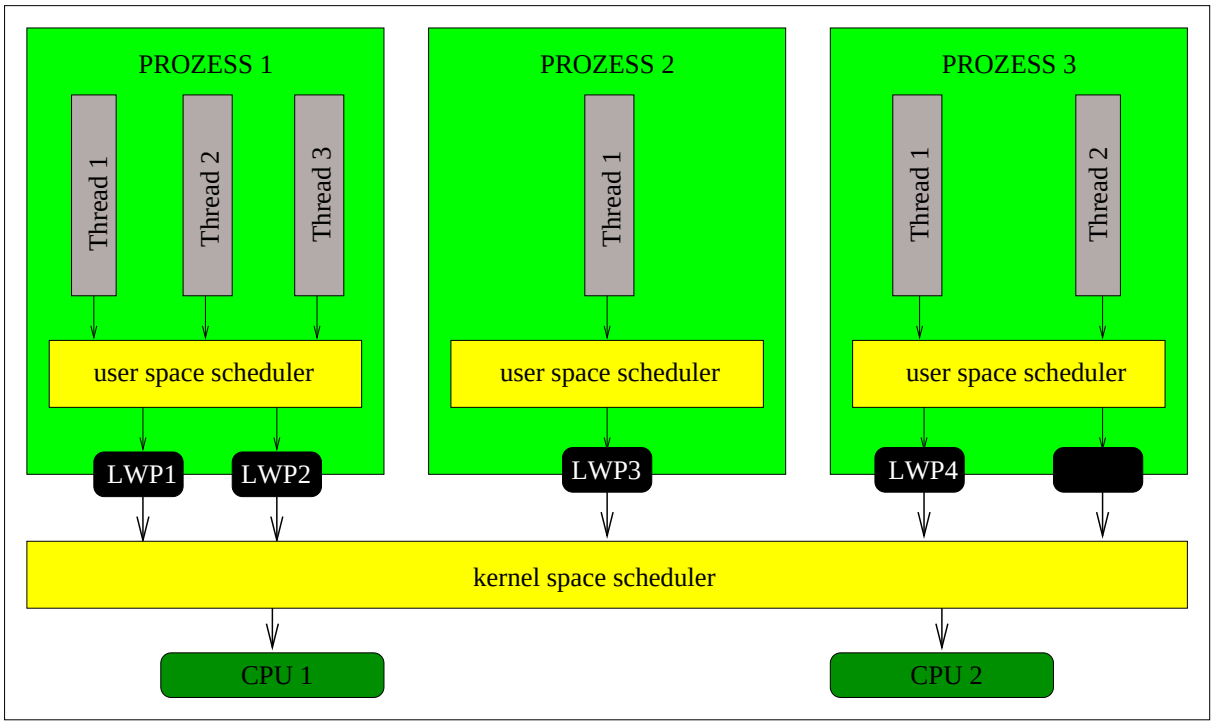

Abbildung 3.3 Hybride Thread-Implementierung mit mehrstufiger Zuteilung

tinstanz, den *Master*-Steuerfluß übergeben, dieser ruft wiederum den externen Scheduler auf, der den nächsten Steuerfluß bestimmt, das Ergebnis und den Steuerfluß zurück an die Umschaltinstanz gibt, die schließlich auf den ausgewählten Fluß umschaltet. Der Anwender-Scheduler kann wie ein normaler Steuerfluß während des Programmlaufs dynamisch erzeugt und zerstört werden, muß jedoch einige zusätzliche Anforderungen erfüllen.

- Er muß dem Master Steuerfluß als Scheduler bekannt sein.
- Er muß Zugriff auf die darunterliegenden Verwaltung-Strukturen haben.
- Es muß ein Datenaustausch bzw. Aufruf zwischen Umschaltinstanz und Scheduler stattfinden.
- Es finden bei jeder Umschaltung vier Umschaltoperationen statt [7].

Bewertung der Schedulingstrategien

Eine Schedulingstrategie sollte sich durch geringen Verwaltungsaufwand, effektive Lastbalanzierung, Effizienz bezüglich Speicherverbrauch sowie Ausnutzung der Lokalität von Threads auszeichnen. In [JE98] wird auf die Auswirkungen verschiedener Schedulingstrategien auf Zeit- und Platzbedarf von vielfädigen numerischen Programmen hingewiesen. Unter bestimmten Umständen kann die Erzeugung und Verwaltung einer zu großen Anzahl von Threads zu einem Performance-Einbruch infolge einer Erschöpfung von Rechenressourcen, insbesondere des Speichers, führen.

[7] mittels der Prozedur `transfer()`

Ordnet man die Erzeugung und Zerstörung von Threads innerhalb eines Programms als Berechnungsgraph an, mit Knoten als Symbol einer Berechnung und Kanten als Symbole für Abhängigkeiten, lassen sich einfach mögliche parallele Berechnungsreihenfolgen betrachten. Falls lauffähige Threads gleicher Priorität in einer FIFO Warteschlange verwaltet werden, wird der Berechnungsgraph in *Breadth-First* Ordnung durchlaufen und damit bedeutend mehr gleichzeitig aktive Threads erzeugt und verwaltet als mit einem LIFO (Last-In-First-Out) Keller. Bei letzteren Ansatz wird der Berechnungsgraph *Depth-First* durchlaufen und schränkt damit den möglichen Parallelismus, aber auch den Speicherverbrauch durch viel weniger gleichzeitig aktive Threads ein.
Ähnlich arbeitet auch ein Leichtgewichts-Prozeßsystem, das sowohl im Kernel- als auch im User-Raum implementiert ist wie die Solaris *Light Weight Processes* [Gra95]. Wie in Abbildung 3.3 illustriert besitzt jeder Prozess mindestens einen Steuerfluß sowie einen *Userlevel* Scheduler, der die Steuerflüsse des Prozesses verwaltet, aber nur jeweils einen rechenbereiten Fluß an einen *Light Weight Process* (LWP) verknüpft und an das Betriebssystem weiterreicht. Das Betriebssystem teilt dann den rechenbereiten *LWPs* aller Prozesse mit dem *Kernel-Space* Scheduler an den oder die Prozessoren zu. Auf diese Weise wird unter anderem verhindert, daß blockierte oder sonstwie nicht rechenbereite Steuerflüsse Rechenressourcen oder Datenstrukturen im Kernel verschwenden und somit stets eine, im Sinne traditioneller Ressourcenzuteilung, optimale Zuteilung gewährleistet ist.

In der Literatur existieren mehrere Ansätze, um maximale Speicherverbrauchsgrenzen zu garantieren. Multilisp [Hal85] benutzt LIFO Keller auf jedem Prozessor zur Einschränkung des möglichen Parallelismus. *Lazy thread creation* [EDR90] verzögert die Belegung von Rechenressourcen für einen Thread solange, bis der Thread auch wirklich parallel ausgeführt werden kann. Das Cilk multi-threaded System [DFC+96] garantiert eine maximale Speichergröße durch Restriktion der Ausführungsreihenfolge der Threads auf eine Untermenge aller möglichen Folgen.

3.1.6 Nachteile prioritätenbasierter Ressourcen-Zuteilungsverfahren

Herkömmliche Strategien zur Ressourcenzuteilung besitzen drei grundlegende Nachteile, die sie zur Verwaltung heterogener Ressourcen in Weitverkehrsnetzen ungeeignet machen. Traditionelle Verfahren sind asymmetrisch, abgeschlossen und zentral, drei Eigenschaften, auf die im folgenden genauer eingegangen wird.

Asymmetrie – Herkömmliche Methoden berücksichtigen in der Regel nur eine einzige Ressource (meist die des Prozessors) oder – falls mehrere Ressourcen verwaltet werden - zeichnen eine Ressource (wiederum meist die des Prozessors) aus. Diese asymmetrische Betrachtungsweise der Ressourcen hat das Ziel der maximalen Auslastung der Ressource des Prozessors innerhalb eines Systems. Durch eine Zuteilung der Prozessorkontrolle ebenfalls notwendig werdende zusätzliche Ressourcen wie Speicher oder Ein- und Ausgabeoperationen bleiben unberücksichtigt. Selbst im Fall einer asymmetrischen Verwaltung mehrerer, aber nicht gleichberechtigter Ressourcen, können diese nicht mit üblichen prioritätenbasierten Verfahren untereinander vergleichbar gemacht werden, da sie in inkompatiblen Einheiten gemessen werden.

Abgeschlossenheit – Prioritätenbasierte Verfahren sind per se abgeschlossen, da ihre Funktionsweise auf einer absoluten Ordnung der Prioritäten beruht. Im Fall der Kop-

pelung verschiedener Systeme mit jeweils inhärenten absoluten Ordnungen zu einem globalen System geraten diese Ordnungen ins Ungleichgewicht und es entsteht abhängig von den absoluten Werten der individuellen Prioritäten eine willkürliche neue globale Ordnung, die zu einer völlig anderen Ressourcenverteilung als in den Teilsystemen führt.

Zentralismus – Herkömmliche Zuteilungsverfahren basieren wie bereits erwähnt auf einer absoluten Ordnung bezüglich eines oder mehrerer ausgezeichneter Parameter (in den meisten Fällen einer Art Priorität) und benötigen daher eine allwissende, zentrale Instanz, die diese Parameter aller beteiligten Einheiten verwaltet und auswertet, um zu einer Zuteilungsentscheidung zu gelangen. Diese zentrale Instanz stellt bei genügender Größe des Systems oder ausreichend vielen Parametern einen Engpaß der Zuteilungsfindung und damit des gesamten Systems dar, da unter Umständen ungenutzte Ressourcen aufgrund des Engpasses nicht zugeteilt werden und verfallen.
Verschärft wird diese Tatsache durch den sehr dynamischen Ressourcen-Pool von offenen Systemen wie sie Weitverkehrsnetze darstellen. Zentrale Zuteilungsverfahren mögen geeignet sein, einen statischen Pool von Ressourcen innerhalb eines geschlossenen Systems effizient zuzuteilen, zur Verwaltung einer Vielzahl sehr heterogener und dynamischer Ressourcen sind sie völlig ungeeignet.

3.2 Marktbasierte Ressourcenzuteilung

Ein zunehmendes Problem bei der optimalen Auslastung verteilter Rechensysteme stellt die Bewältigung der Dynamik und Komplexität bei der Koordination vieler Aufgaben auf verschiedenen Rechnern und Prozessoren dar. Durch das stete Anwachsen verteilter Netzwerke bzw. den Zusammenschluß lokaler Netzwerke zu Globalen entwickelt sich die zentrale Allokation und Koordination verteilter Ressourcen und Jobs zu einer immer schwierigeren Aufgabe. Da Jobs und Daten verteilt sind und sich fortlaufend verändern, kann ein zentrales Kontrollorgan gar nicht über alle Informationen verfügen, die zu einer effizienten Planung und Zuteilung notwendig sind. Ausfallsicherheit und schnelle Reaktion auf lokale Veränderungen erfordern vielmehr ein verteiltes, lokales Management von Ressourcen und Aufgaben. Zur Allokation und Verwaltung globaler, verteilter Systeme werden daher effiziente Verfahren zur dezentralen Ressourcenzuteilung gebraucht.

Neben solcher Verfahren ist auch ein tieferes Verständnis großer Verbände lokal verwalteter, asynchroner und nebenläufiger Prozesse wünschenswert, die mit einer unvorhersehbaren Umgebung interagieren. Insbesondere die Zusammenhänge zwischen dem Verhalten des Gesamtsystems und das seiner lokalen Komponenten, deren Entscheidungen auf lokalen, unter Umständen unvollständigen, veralteten oder widersprüchlichen Informationen beruhen, sind dabei von besonderem Interesse. Wegen dieser Eigenschaften und Phänomene, die sich auch in biologischen und ökonomischen Systemen wiederfinden, werden solche Verbünde interagierender Prozesse als Informations-Ökosysteme (*computational ecosystems*, [MD88]) bezeichnet. Diese Form der Zuteilung wird auch als *offen* bezeichnet, da sie der in der Informatik verbreiteten Annahme eines geschlossenen Systems widerspricht.

Traditionelle Ressourcenzuteilungs-Verfahren basieren auf einer zentralen Kontrolle und Verwaltung zur Allokation von Ressourcen an Prozesse, eine Vorgehensweise, die für

große verteilte Systeme mit sehr dynamischem Verhalten nicht geeignet ist. Der Problematik des Schedulings in verteilten Systemen widmeten sich bereits eine Reihe von Forschungsprojekten, deren Schwerpunkt meist auf deterministischen mathematischen Modellen oder der Formulierung in der Praxis funktionierender Heuristiken lag. Die dabei entwickelten mathematischen Modelle, zum Beispiel aus der Graphentheorie, beruhen dabei auf vereinfachenden Annahmen, die in der Praxis meist nicht erfüllbar sind [Bog94]. Die häufiger anzutreffenden Heuristiken basieren auf der Messung und Bewertung von Lastindizes zur Zuteilungsfindung [Bog94].

Der folgende Abschnitt beschäftigt sich nach einer Einführung in ökonomische Modelle und marktbasierter Ressourcenzuteilung in der Informatik mit dem Handel in elektronischen Märkten. Anschließend werden grundlegende Begriffe und Verfahren ökonomischer Modelle und konkrete Implementierungen elektronischer Märkte mit ihren Vor- und Nachteilen vorgestellt. Abgerundet wird der Abschnitt schließlich mit einer Übersicht und Klassifikation dieser Märkte im Hinblick auf die Ziele dieser Arbeit.

3.2.1 Ökonomische Modelle in der Informatik

Zur Lösung der Problematik der Inkompatibilität von Ressourcen kann den Systemressourcen ein homogenes Maß, die Kosten, zugeordnet werden, die vom augenblicklichen Vorrat dieser Ressource im System abhängig sind. Die Kosten einer Anwendung bestehen aus der Summe der Kosten all ihrer benötigten Ressourcen. Zur Entwicklung eines Gleichgewichtes zwischen Angebot und Nachfrage nach limitierten Systemressourcen bieten sich aufgrund der umfangreichen Forschung und vorhandenen Literatur ökonomische Modelle aus der freien Marktwirtschaft an. Diese sollten in der Lage sein, den für das Prozeßsystem günstigsten, da bezüglich der zur Verfügung stehenden knappen Ressourcen gewinnbringendsten Prozeß zur Zuteilung der Systemressourcen auszuwählen.

Bei der Definition der handelbaren Güter muß immer ein geeignetes Maß zwischen der Heterogenität der Güter und der Größe ihrer Märkte gefunden werden. Erlaubt man den Handel sehr heterogener Güter, erhöhen sich die Transaktionskosten und reduziert sich gleichzeitig die Marktbreite der individuellen Güter, so daß unter Umständen ein Gleichgewichtspreis zwischen Angebot und Nachfrage nicht erreicht wird. Werden im Markt dagegen eher wenige homogene Güter gehandelt, werden zwar die Transaktionskosten durch die höhere Marktbreite der Güter reduziert, allerdings kann Angebot oder Nachfrage nach individuelleren Gütern nicht befriedigt werden. Bei sehr homogenen Rechenressourcen ist beispielsweise die Befriedigung sehr spezieller Sicherheits- oder Architekturbedürfnisse nicht mehr möglich.

Bevor ein marktbasiertes Ressourcenzuteilungs-Modell angewendet werden kann, muß für die Systemressourcen, die sich zum Teil erheblich von realen Gütern unterscheiden, ein geeignetes Ressourcenmodell gefunden werden. Dazu muß zunächst eine Definition und Bewertung der innerhalb dieses Marktes handelbaren Güter gefunden werden. Die Begriffe Rechenzeit, Speicherbedarf, Ein- und Ausgabebedarf reichen zur Beschreibung dieser Güter nicht aus, da sie keine Details wie Einheiten, Gütereigenschaften, Termine und Garantien enthalten.

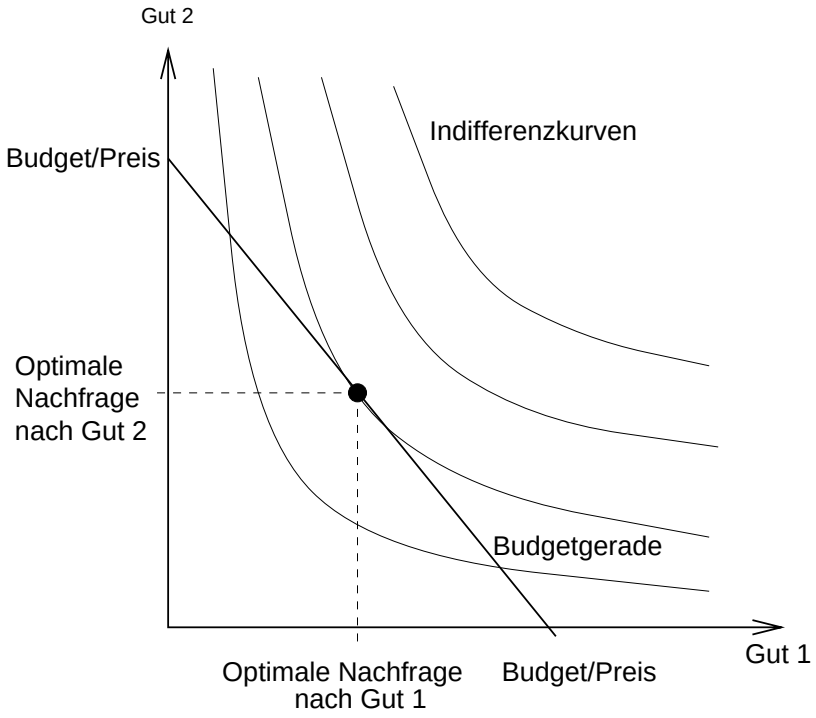

Abbildung 3.4 Isographenlinie mit Orten gleichen Nutzens [Bac98]

Marktbasierte Ressourcenzuteilung

Eine der Hauptforschungsbereiche der Ökonomie ist die Zuteilung von Ressourcen, insbesondere knapper Ressourcen.

Definition 3.14 (Knappheit) *Bei einem knappen Gut handelt es sich um eine Ressource, dessen Nachfrage das Angebot übersteigt, falls dessen Kosten Null betragen.*

Das Standard Beispiel für nicht knappe Ressourcen ist Luft, die nichts kostet und die im Überfluß vorhanden ist. In einem Prozeßsystem sind alle Ressourcen wie Prozessorzeit, Hauptspeicher oder Ein- und Ausgabeoperationen, knapp. Auch die Erhöhung des Vorrats um endliche Bestände innerhalb eine Prozeßsystems kann diese Knappheit nicht beheben [8]. Nur ein unendlich großer Vorrat kann ein knappes Gut zu einer nicht-knappen Ressource machen. Zur optimalen Zuteilung knapper Ressourcen wird in ökonomischen Systemen das Marktmodell eingesetzt, das in vielen Fällen optimale Ressourcenzuteilung ohne oder nur mit geringer zentraler Steuerung erzielen kann. Optimale Zuteilung heißt, daß die Ressourcenzuteilung durch einen allwissenden Zuteiler – ein Zuteiler, der die Zukunft kennt – nicht weiter verbessert werden könnte.

Definition 3.15 (Markt) *Ein Markt ist ein Koordinationsmechanismus, der Angebot und Nachfrage zusammenführt, um den Güter- und Dienstleistungsaustausch (in einer Volkswirtschaft) zu ermöglichen.*

[8] Diese Erfahrung haben bereits zahllose Programmierer gemacht, die einst dachten, *niemals* mehr als 256 Kilobyte Hauptspeicher zu benötigen, und nun mit 32 Megabyte nicht auskommen

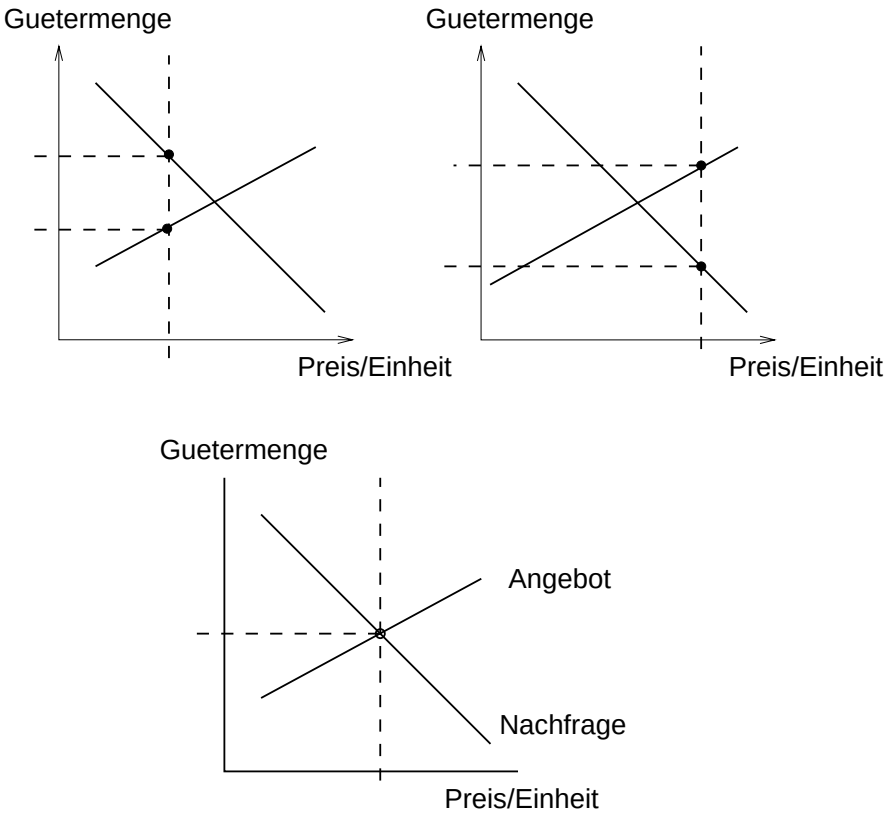

Abbildung 3.5 Equilibrium zwischen Angebot und Nachfrage [Bac98]

Definition 3.16 (Marktplatz) *Der Marktplatz ist der Ort, an dem der Austausch gehandelter Güter und Leistungen stattfindet.*

Wie bereits erwähnt sind marktbasierte Koordinationsformen geeignet mehrere gleichberechtigte Ressourcen effizient zuzuteilen. Dazu muß der Marktteilnehmer zwischen den gleichberechtigten Gütern (auch der Preis ist ein Gut) abwägen, da nicht alle Ressourcen zu den gewünschten Mengen oder Preisen verfügbar sind. Hierzu dient der Begriff des Nutzens, der den (subjektiven) Wert eines Marktteilnehmers für eine Ressource bezeichnet und die Abwägung zwischen ein oder mehrere Ressourcen und ihren Preisen ermöglicht.

Definition 3.17 (Nutzen) *Der Nutzen einer Ressource ist der (subjektive) Wert, den eine Ressourcen für einen Marktteilnehmer besitzt.*

Trägt man alle Orte gleichen Nutzens für zwei Güter in Isographenlinien, den sogenannten Indifferenz-Kurven ein, und bringt sie mit der marktgegebenen Preis-Budget-Funktion zum Schnitt, erhält man die für einen bestimmten Marktteilnehmer optimale Ressourcenverteilung zum günstigsten Preis. Dieser Sachverhalt entspricht der gleichzeitigen optimalen Zuteilung von beispielsweise Rechenzeit und Hauptspeicher und ist in Abbildung 3.4 illustriert.

Der Markt ist, neben anderen Eigenschaften, insbesondere geeignet, den sogenannten Gleichgewichtspreis zu ermitteln, der zwischen Angebot und Nachfrage eines Gutes ausgleicht, das heißt der Preis, der möglichst viele Marktteilnehmer – Verkäufer und Käufer – zufriedenstellt. Trägt man die Nutzen-Funktionen von Verkäufern und Käufern, die ihre Bereitschaft einen Handel abzuschließen in Abhängigkeit vom erzielten Preis bezeichnen, in einem Koordinatensystem auf, ergibt sich der Gleichgewichtspreis, wie in Abbildung 3.5 illustriert, aus dem Schnittpunkt dieser beiden Funktionen. Wählt man einen niedrigeren Preis, ergibt sich eine Differenz, da zu diesem Preis mehr Käufer als Verkäufer Handel treiben wollen, während bei einem höheren Preis sind mehr Verkäufer als Käufer bereit, Handel zu treiben. Die Auswirkungen beider Ungleichgewichte sind ebenfalls aus Abbildung 3.5 ersichtlich.

Definition 3.18 (Transaktion) *Unter einer Transaktion versteht man eine Interaktion zwischen zwei oder mehreren Marktteilnehmern zur Vorbereitung, Vereinbarung oder Abwicklung eines Güter- oder Dienstleistungsaustausches.*

Definition 3.19 (Transaktionskosten) *Unter Transaktionskosten werden sämtliche fixe und variable Kosten verstanden, die durch eine Transaktion – insbesondere des Güteraustausches – verursacht werden.*

Transaktionen gliedern sich in Informations-, Vereinbarungs- und Abwicklungsphase.

Informationsphase – Die Informationsphase ermöglicht den Marktteilnehmern, sich einen Überblick über die gesamtwirtschaftlichen und branchenspezifischen Rahmenbedingungen, über potentielle Geschäftspartner und deren Solvenz und Lieferfähigkeit sowie die angebotenen Güter und Dienstleistungen in diesem Markt zu verschaffen. Nach Abschluß dieser Phase verfügen die Teilnehmer über genügend Informationen, um ein Gebot abgeben zu können.

Vereinbarungsphase – In der Vereinbarungsphase werden die Details eines Vertrages ausgehandelt, also die Konditionen und Bedingungen der Transaktionen, die zur Aushandlung und zum Abschluß eines Vertrages über den Bezug von Waren und Dienstleistungen notwendig sind. Der Vertrag kann dabei als Folge eines einseitigen Angebots, einer bilateralen Aushandlung oder einer Auktion zustande kommen.

Abwicklungsphase – Die Abwicklungsphase stellt die Einhaltung und Überwachung des abgeschlossenen Vertrags sicher, zum Beispiel den Güteraustausch, die Zahlungsabwicklung sowie den Austausch von Rechnungen und Quittungen.

Besonderheiten von Marktmechanismen

Das Marktmodell besitzt eine Reihe von Vorteilen, die den Einsatz marktbasierte Zuteilungsstrategien in Prozeßsystem rechtfertigen und die im folgenden vorgestellt werden.

Effizienzbegriff – Ökonomische Theorie liefert in vielen Fällen einen geeigneteren Effizienzbegriff als der in der Informatik gebräuchliche, in der er überwiegend im Sinne von schneller im Vergleich zu anderen Verfahren verwendet wird. In der Informatik bezieht er sich oft nur auf die Ressource (Prozessor-) Zeit. Selbst wenn mehrere

andere Ressourcen wie zum Beispiel Speicher, Plattenplatz, Algorithmus zur Ermittlung der Effizienz herangezogen werden, ist dennoch die Ressource (Prozessor-) Zeit ausgezeichnet. Dagegen bezieht der ökonomische Effizienzbegriff sämtliche Ressourcen und ihre Wechselwirkungen aufeinander zur Erzielung maximalen Nutzens miteinbezieht. Ein Maß für die Effizienz einer Zuteilung mehrerer Ressourcen ist die Pareto-Optimalität [Bog94].

Definition 3.20 (Pareto-Optimalität) *Eine Zuteilung von Ressourcen ist pareto-optimal, falls jede Veränderung an der Zuteilung zur Verschlechterung einer der beteiligten Parteien führt, das heißt jede Partei hat bei der Zuteilung ihren Nutzen maximiert.*

Der Effizienzbegriff kann verwendet werden, um bei der Definition rationaler Ressourcenzuteilung behilflich zu sein. Die Pareto-Optimalität impliziert, daß die Zuteilung unter Berücksichtigung der Präferenzen aller beteiligten Parteien nicht verbessert werden kann. Daher können alle pareto-optimalen Zuteilungen als rationale Zuteilungen bezeichnet werden, da eine Abweichung von dieser Zuteilung zu keiner Verbesserung aller Parteien führt.

Dezentrale Entscheidungsfindung – Marktsysteme basieren auf dezentraler Entscheidungsfindung. Um ein optimales globales Gleichgewicht zu erzielen ist **kein** zentraler und allwissender Entscheidungsträger notwendig. Es genügt, wenn jeder Teilnehmer am Markt nur lokale Informationen auswertet, um Entscheidungen bezüglich seiner Präferenzen und Bewertung von gehandelten Ressourcen zu treffen.

Dies ist aus einer Reihe von Gründen erstrebenswert. Aufgrund der Menge von global vorliegenden Informationen kann es ineffizient oder sogar unmöglich sein, alle Informationen bei einer zentralen Entscheidungsfindung in genügend kurzer Zeit zu berücksichtigen. Im Fall unsicherer oder widersprüchlicher Informationen muß eine lokale Entscheidungsfindung abhängig von wenigen, potentiell unsicheren Informationen ihre Präferenzen setzen.

Dynamische Anpassung – Zu den größten Stärken von Marktsystemen gehört die Fähigkeit, auf plötzliche, unvorhergesehene Ereignisse sehr schnell reagieren zu können. Jeder Marktteilnehmer muß nur seine eigenen Einschätzungen und Präferenzen hinsichtlich dieser Ereignisse neu bewerten, um einen erneuten Gleichgewichtszustand zu erreichen, der die Veränderungen innerhalb des Marktes reflektiert. Eine zentrale Stelle ist zur Bestimmung des neuen Gleichgewichts nicht notwendig. Aus diesem Grund können Marktsysteme ebenfalls sehr schnell auf verändertes Angebot und Nachfrage reagieren und so die Produktion von Waren, die nicht mehr nachgefragt werden, einstellen oder mit der Produktion des entsprechenden Guts bei plötzlicher Nachfrage beginnen.

Ein unangenehmer Seiteneffekt dieser dynamischen Anpassung ist die Tatsache, daß Märkte oft chaotische Systeme sind, in denen sehr kleine Veränderungen zu großen Auswirkungen auf den zukünftigen Lauf des Marktes haben. Die Chaos-Eigenschaft macht eine Analyse oder Vorhersage des Verhaltens des System schwieriger.

Nachteile marktbasierter Ressourcenzuteilung

Der Einsatz von ökonomischen Strategien bei der Vergabe von Ressourcen in Prozeßsystemen ist jedoch nicht uneingeschränkt empfehlenswert. Neben der schon in Abschnitt 3.2.1 besprochenen Neigung zum chaotischen Verhalten existieren noch eine Reihe anderer Probleme marktbasierter Zuteilung wie eine geeignete Initialisierung, die Errechnung anstelle einer Erreichung von Gleichgewichtszuständen (*Equilibrii*) sowie die höheren Kosten von marktbasierten Entscheidungen.

Initialisierung – Ein noch weitgehend ungelöstes Problem ist die Festlegung der Kosten der gehandelten Güter, der Erträge und der Präferenzen der Teilnehmer, insbesondere der Konsumenten eines marktbasierten Prozeß-Systems. Die Teilnehmer – insbesondere die Konsumenten – bilden die Regisseure in einem Marktsystem. Die Nachfrage nach Gütern und Dienstleistungen legt den Marktwert fest und bestimmt so, welche Güter produziert und welche Dienstleistungen durchgeführt werden. Die Bestimmung der Präferenzen und die anfängliche Festlegung der Güter- und der Währungkurse ist ein noch sehr junges Gebiet der Forschung.

Equilibrii – Desweiteren unterscheiden sich reale Märkte von den in Prozeßsystemen implementierten dadurch, daß in ersteren Gleichgewichtszustände nicht errechnet werden, sondern sich entwickeln. Ob und wie sich diese Tatsache auf die Preise, Präferenzen und Gleichgewichtszustände von elektronischen Märkten und Simulationen auswirkt, ist ebenfalls noch Gegenstand aktueller Forschung. Desweiteren erhöhen sich bei marktbasierter Zuteilung die Transaktionskosten zur Entwicklung von effizienten Gleichgewichtszuständen, da diese im Gegensatz zu *Scheduling*-Algorithmen nicht auf geringe Komplexität hin entwickelt wurden.

Effizienz – Ein Punkt, dem besonderes Augenmerk gewidmet werden muß, ist das Verhältnis zwischen den Kosten, das heißt sämtliche aufgewendete Ressourcen insbesondere der Zeit, und dem Nutzen einer marktbasierten Zuteilung, das heißt der durch sie erzielten Verbesserung des Prozeßsystems. Dies ist notwendig, da marktbasierte Verfahren rechnerisch aufwendiger sind als prioritätenbasierte Strategien, die auch besonders im Hinblick auf ihre Effizienz entwickelt wurden. Betriebswirtschaftlich gesprochen erhöhen marktbasierte Verfahren die Transaktionskosten und reduzieren dadurch den durch sie erzielbaren Nutzen.

3.2.2 Bewertung von Gütern und Dienstleistungen

Um mit Gütern und Ressourcen einer Rechen-Infrastruktur Handel betreiben zu können, müssen ihnen zunächst relative Werte zugeordnet werden. In einer Ökonomie von Rechenressourcen könnten diese Güter bestimmte Aufgaben sein, die berechnet werden oder Dienste, die durchgeführt werden sollen. Auch andere Ressourcen wie Hauptspeicher und Plattenkapazität können wie Güter behandelt werden, erfordern aber besondere Aufmerksamkeit, weil sie über einen festgelegten *Zeitraum* gemietet anstatt gekauft werden.

Zu einer Ökonomie von Rechenressourcen gehören ebenfalls Anbieter und Nachfrager mit individuellen Vorlieben und Produktionsfähigkeiten, aus denen sich relative Bewertungen oder Nutzen dieser Ressourcen ergeben. Es besteht jedoch keine Notwendigkeit, diese relativen Werte zentral festzulegen. Sie entstehen selbständig aus Angebot

und Nachfrage innerhalb des Marktmechanismus. Die Nachfrage nach Diensten, die zu ihrer Durchführung Rechenzeit benötigen, bestimmt den relativen Marktwert von Rechenzeit. Analog bestimmen die Kosten zur Bereitstellung von Rechenzeit die Kosten zur Durchführung dieser Dienste, und damit ihren Marktpreis. Analog beeinflussen die Kosten zur Bereitstellung von Rechenzeit den Marktpreis von darauf aufbauenden Dienstleistungen. Der Marktmechanismus sorgt für ein Gleichgewicht all dieser stark voneinander abhängigen Verhältnisse. Daher ist ein Markt für Rechenzeit alleine ohne Berücksichtigung anderer Güter und Dienstleistungen in einer Ökonomie nicht sinnvoll.

Zur Beschreibung der Tauschverhältnisse und zur Vereinfachung der Notation ist die Einführung einer Ressource sinnvoll, die stets den Wert 1 besitzt. Diese Ressource wird nur als Währung zum Ausdruck relativer Werte oder Preise verwendet und besitzt eigentlich keine eigene Kaufkraft. Sie wird nur zur Vereinfachung bei der Beschreibung von Preisen verwendet.

Definition 3.21 (Preis) *Unter dem Preis eines Gutes oder einer Dienstleistung versteht man das Tauschverhältnis zu einem ausgezeichneten Gut, das den Wert 1 besitzt.*

3.2.3 Besonderheiten eines Ressourcenmarktes

Obwohl ein Markt für Rechenressourcen bisher einem realen Markt für mietbare Güter und Dienstleistungen relativ ähnelte, existieren jedoch Unterschiede zu materiellen Gütermärkten.

Einzigartigkeit – Die Ressource Prozessor ist (zumindest in Einprozessor-Systemen) ausgezeichnet und unteilbar. Es handelt sich demnach eigentlich um ein Monopol mit einem einzigen Exemplar des Produkts. In traditionellen Märkten können Güter aufgeteilt und gelagert werden. Die Ressource Prozessor ist nicht teilbar und ihr augenblicklicher Wert verfällt bei Nichtgebrauch mit sofortiger Wirkung. Dies bedeutet, daß in einem solchen geschlossenen Markt keine Möglichkeit besteht, ausgleichend mit Rechenleistung zu wirtschaften. Erst eine Öffnung eines solchen monopolistischen Marktes durch Koppelung mit anderen "Monopolen" ermöglicht ein freies Handeln mit überschüssiger Rechenleistung.

Zeitabhängigkeit – Die Ressource Prozessor ist extrem zeitkritisch. Es können kaum Voraussagen über zukünftige Ereignisse wie Reservierungen gemacht werden, da gegenwärtige Aktionen die zukünftige Verfügbarkeit des Prozessors bestimmen. Außerdem ist die Überlassung der Kontrolle über den Prozessor nur zeitweise, die Ressource wird also *vermietet* anstatt verkauft. Aus diesem Grund sollten auch alle Preise und Gebote zeitbezogen sein.

Diskrete Verfügbarkeit – Der Prozessor ist nur zu diskreten Zeitpunkten und für bestimmte Intervalle verfügbar. Erstere ergeben sich aus den Umschaltpunkten des Schedulers und letztere sind durch Randbedingungen des Marktes bedingt. Zu kurze Intervalle können beispielsweise unrentabel sein, weil sie den Aufwand zur Zuteilung eines neuen Jobs nicht rechtfertigen oder sich kein Nachfrager findet, der für sehr kurze Rechenintervalle Nutzen besitzt. Bei der Zuteilung zu langer Intervalle reagiert ein Marktsystem unter Umständen zu langsam auf sich verändernde Ressourcenpreise.

Aus den genannten Besonderheiten folgt, daß es sich bei Rechenleistung um eine extrem zeitkritische Ressource mit unendlich hoher Abschreibung handelt. Ressourcen, die nicht gelagert werden können und auf die nicht unmittelbar zugegriffen werden kann, machen die Garantierung von Dienstgüten (*Quality of Service*) unmöglich und schränken auch die Vermarktbarkeit stark ein. Rechenressourcen erlauben demzufolge keinen Handel von Optionen auf Ressourcen, da über die zukünftige Verfügbarkeit keine Aussagen gemacht werden kann. Beim herkömmlichen Handel von Ressourcen können ebenfalls keine Garantien gemacht werden, ob nach der Vertragsaushandlung über den Bezug von Ressourcen diese auch sofort zur Nutzung bereitstehen. Es ist nicht unwahrscheinlich, daß zu einem bestimmten Zeitpunkt t der Nachfrage nach Ressourcen *kein* Angebot gegenübersteht. Ressourcen lassen sich aufgrund der diskreten Verfügbarkeit nur unmittelbar bei ihrer Entstehung handeln. Es folgt, daß die notwendige Transaktionszeit und Transaktionskosten zur Etablierung eines Kaufvertrages auf ein absolutes Minimum reduziert werden müssen.

3.2.4 Ideale und effiziente Zuteilung

Der ideale Rechenmarkt sollte die Ressourcen des Prozeßsystems effizient und rational im Sinne von Pareto-Optimalität zuteilen.

1. Implementierung einer rationalen Zuteilung, die stets den produktivsten, das heißt gewinnbringendsten, Prozeß zuteilt

2. Rechnerisch effizienter Mechanismus zur Verwaltung und Ermittlung von Geboten und Auktionen. Falls eine effizient errechnete Zuteilung nicht mehr möglich ist, die Kosten zur Entscheidungsfindung zu teuer werden, sollen Standardprioritäten für Prozesse als Notfallplan Verwendung finden.

3. Günstiges Verhältnis zwischen Berechnung des Marktgleichgewichtes und Nutzungszeit des Prozeßsystems

Ein marktbasiertes Verfahren sollte die Systemressourcen ideal zuteilen, das heißt, daß es den erzielten Mehrwert des Systems durch die Produktion von Gütern durch den Einsatz von Ressourcen maximieren sollte.

Definition 3.22 (Ideale Zuteilung) *Unter idealer Zuteilung versteht man eine Zuteilung von Ressourcen, die zu jedem Zeitpunkt den Einsatz von Ressourcen erzielbare Mehrwert des Prozeßsystems maximiert.*

Eine ideale Zuteilung ist jedoch aus zwei Gründen praktisch unmöglich. Zunächst müßten zur Bestimmung der idealen Zuteilung alle möglichen Zuteilungen im Marktsystem bestimmt werden, da jede Zuteilung die relative Bewertung zukünftiger Berechnungen ändern kann. Dazu werden die zu jeder Zuteilungszeit vorhandenen Nachfrager nach Ressourcen als Knoten in einen Baum eingetragen, deren Knoten mit dem jeweiligen Nutzen und dessen Kanten mit den individuellen Kosten bezeichnet werden. In diesem Baum ist nun der Weg mit der höchsten Kantensumme gesucht, also die Zuteilung, die den Gewinn des Systems maximiert. Diese Suche ist von der Ordnung $O(nlogn)$, wobei

n die Anzahl der Prozeße im System darstellt. Die Zeit zur Suche selbst, die keine ideale Zuteilung der Ressourcen nach obiger Definition darstellt, beeinflußt ebenfalls zukünftige Bewertungen und muß daher in der Berechnung aller möglichen Pfade mitberücksichtigt werden. Schließlich lassen sich in einem unendlich lange laufendem Prozeßsystem nicht alle möglichen Pfade in endlicher Zeit berechnen [Bog94]

Selbst wenn sämtliche Pfade in endlicher Zeit berechnet werden könnten, ist es möglich, daß ideale Pfade, die kürzere Zeitintervalle bewerten, keine Präfixe von idealen Pfaden unter Berücksichtigung längerer Zeitintervalle sind, das heißt, ein idealer Pfad zum Zeitpunkt n erweist sich im nächsten Zeitschritt $n + 1$ nicht mehr als Präfix eines idealen Pfades. Daher ist es sinnvoll, ein theoretisch erreichbares Ziel bei der Ressourcenzuteilung zu definieren, die effiziente Zuteilung.

Definition 3.23 (Effiziente Zuteilung) *Eine effiziente Zuteilung ist eine Zuteilung, in der die erwartete zukünftige Zuteilung ideal ist.*

Das bedeutet, daß zu jedem Zeitpunkt der Prozess zugeteilt wird, der den ersten Schritt einer idealen Zuteilung repräsentiert. Eine effiziente Zuteilung ist ideal, falls sämtliche zukünftigen Berechnungen bekannt sind.

Als letztes Problem verbleibt die Tatsache, daß die die Zeit, die zur *meta-greedy*-Suche aufgewendet wird nicht der Definition von effizienter Zuteilung entspricht. Dies kann dadurch gelöst werden, daß die Ressourcen, die für die Suche benötigt werden, der Rechenzeit des effizientesten Prozesses zugeordnet werden [Bog94].

3.2.5 Unterschiede zum Scheduling

Ein Scheduler eines Prozeßsystems garantiert bestimmte Eigenschaften, so zum Beispiel daß kein Prozeß ausgehungert werden kann. Selbst wenn die Priorisierung eines Prozesses im Augenblick für eine Zuteilung von Ressourcen nicht ausreichend ist, so wird seine Priorität solange erhöht, je länger er wartet, bis er schließlich zugeteilt wird. Diese Garantien können in einem marktbasierten Prozeßsystem unter Umständen nicht erfüllt werden, auch wenn dieser Fall über einen längeren Zeitraum unwahrscheinlich ist. In einem Rechenzeit-Markt werden zu jedem Zeitpunkt nur die profitabelsten Marktteilnehmer zugeteilt. Ein Teilnehmer, der keinen oder kaum Profit erzielt, kann über längere Zeit nicht zugeteilt werden, solange stets profitablere Marktteilnehmer mit ihm um Ressourcen konkurrieren. Dieses Phänomen ist allerdings genau das, was mit marktbasierten Verfahren erreicht werden soll. Das Marktmodell verlangt nicht, daß Dienstleistungen wertvoller werden, je länger sie warten. In der Praxis wird dies aber in der Regel der Fall sein, so daß die Eigenschaft der Nicht-Aushungerung in Rechenzeitmärkten zwar nicht garantiert ist, aber zumindestens meistens erfüllt ist.

3.2.6 Marktliche Koordinationsformen

Zur Etablierung eines fairen Marktes bzw. einer fairen Zuteilung, wie sie bei herkömmlichen Zuteilungsmechanismen üblich ist, sind geeignete Verfahren zum Ausgleich zwischen Angebot und Nachfrage von Systemressourcen zu bestimmen. Übliche Preisbildungsmechanismen lassen sich in vier Gruppen kategorisieren: Auktionen und Börsen, bilaterale Aushandlung, Ausschreibung und Einschreibung sowie die Angebot- und Nachfrage-

Fixierung [Zah99]. Letztere ist fast zum Allgemeinfall geworden – als Beispiel können sämtliche in Listen oder Katalogen festgelegte Preise angeführt werden – bildet jedoch keine direkte Preisermittlung im eigentlichen Sinne und wird daher im folgenden Abschnitt nicht weiter behandelt.

Auktionsmechanismen

Bei Auktionen handelt es sich um eine Koordinationsform, bei der Nachfrager mit der höchsten Kaufbereitschaft den Zuschlag auf das Gut erhalten, unabhängig davon, ob es sich um den momentanen markthöchsten Preis handelt oder nicht. Zur Ermittlung der Kaufbereitschaft und des Preises dienen Auktionsverfahren mit versiegelten Geboten (*fire-and-forget*), so daß nur eine Runde zur Kommunikation notwendig ist. Außerdem sind sie effizient – zum Beispiel mit einigen wenigen Hash-Funktionen – berechenbar, um den Gewinner der Auktionsrunde zu ermitteln. Es existieren eine Reihe von in der wirtschaftswissenschaftlichen Literatur und Praxis verbreiteten Mechanismen, die sich grob in offene und verdeckte Auktionen einteilen lassen.

Offene englische Auktion Bieter geben ein offenes Gebot für ein bestimmtes Angebot ab, das von jedem Bieter eingesehen werden kann, worauf die Bieter neue Gebote abgeben können. Falls (innerhalb einer Frist) keine neuen Gebote mehr eingehen, erhält das attraktivste Gebot, zum Beispiel das höchste, den Zuschlag.

Erstpreis-Submissionsauktion In dieser Auktionsform geben die Bieter ein einziges verdecktes Gebot ab, so daß nur dem Auktionator die Gebote bekannt sind. Nach Ablauf einer Gebotsfrist erhält das attraktivste Gebot den Zuschlag.

Zweitpreis-Submissionsauktion (wiederholte Vickrey-Auktion) Wiederum geben die Bieter ein einziges verdecktes Gebot ab. Die Besonderheit der Zweitpreis-Submissionsauktion ist die Bestimmung des Preises für ein Angebot. Er wird nicht vom Sieger der Auktion bestimmt, sondern ermittelt sich durch das Gebot des Zweiten.

Eine modifizierte Vickrey-Auktion findet beim *OpenIPO* Börsengang [Lam] Anwendung. Aus allen Geboten von Anbietern und Nachfragern nach Aktien wird ein Räumungspreis ermittelt, der zum Verkauf aller angebotenen Aktien führt. Die höchsten Bieter gewinnen den Zuschlag, aber sämtliche Gewinner zahlen nur den Preis des niedrigsten Gebotes, das noch in die Gewinnergruppe gefallen ist. Auf diese Weise gelangen viele zu einem niedrigeren Preis an ihre Aktien als sie Gebote hierfür angegeben haben. Jüngstes Beispiel eines *OpenIPO* Börsengangs war die Premiere des Web-Dienstleisters *Andover.net* am 8. Dezember 1999 an der amerikanischen Technologiebörse NASDAQ.

Doppelauktion (Offene holländische Auktion) Bei diesem Auktionstyp nennen sowohl Anbieter wie Nachfrager einen Minimal- und einen Maximalpreis, den sie zu zahlen bzw. zu nehmen bereit sind sowie eine Rate, mit der ihr Gebot fällt bzw. steigt, falls zum aktuellen Preis kein Vertrag zustandekommt. Die Anbieter beginnen mit ihrem maximalen Preis und erniedrigen diesen mit der festgelegten Rate, bis ein Käufer gefunden ist. Analog beginnen die Nachfrager mit ihrem minimalen Gebot

und erhöhen diesen mit der festgelegten Rate, bis ein Anbieter gefunden ist, der zum entsprechenden Preis liefert. In jeder Runde kommt nur ein Anbieter-Nachfrager-Vertrag zustande.

Clearinghouse-Doppelauktion Die Clearinghouse Doppelauktion entspricht der normalen Doppelauktion, außer das in jeder Runde mehr als ein Anbieter-Nachfrager-Paar bestimmt wird. Zu bestimmten Handelszeiten werden sämtliche Käufergebote und Verkäuferforderungen zur Bestimmung der aktuellen Angebots- und Nachfrage-Kurve verwendet. Der aus dieser Kurve bestimmte Gleichgewichtspreis wird für alle Transaktionen dieser Runde verwendet.

Strategische Gebote Voraussetzung für eine faire Zuteilung von Ressourcen ist, daß die Zuteilung von Ressourcen ökonomisch effizient ist. Das heißt daß die globale Nutzen-Funktion des (verteilten) Prozeßsystems maximiert wird, indem Ressourcen jederzeit an den Bieter mit dem höchsten Nutzen für die jeweilige Ressource vergeben werden, also den Nutzer, der mit dieser Ressource den höchsten Gewinn zu erzielen in der Lage ist. Um für die faire Vergabe von Rechenressourcen die passende Zuteilungsstrategie zu ermitteln, muß zunächst die Motivation der Nachfrager bzw. Bieter analysiert werden. Geht es den Bietern tatsächlich um eine faire Verteilung der Ressourcen auf alle Nachfrager oder wollen sie vielmehr ihren Gewinn maximieren? Eine gerechte Verteilung kann nur dann erfüllt werden, falls die Bieter dem Auktionator bzw. Zuteiler ihren individuellen Nutzen offenbaren. Falls die Bieter strategisch bieten, das heißt mit falschen Nutzenwerten an den Zuteiler herantreten, kann dieser die Ressourcen nicht optimal vergeben. Strategisches Verhalten der Bieter ist gleichbedeutend mit Geboten, die nicht dem wahren Nutzen der Nachfrager entsprechen, und mit der Angabe höherer Gewinne, als der wahre Nutzen eigentlich erlaubt [Bog94] einhergehen. Hierfür benötigt der Bieter zusätzliche Informationen über das (gegenwärtige und zukünftige) Marktverhalten der Mitbieter, die unter Umständen mit zusätzlichen Transaktionskosten erst noch vom Markt beschafft werden müssen und die Preise erhöhen.

Aus diesem Grund ist es wichtig im Auge zu behalten, welche Informationen über das Verhalten der Mitbieter die jeweiligen Auktionsverfahren zur Verfügung stellen, um seine Eignung für faire Ressourcenzuteilung einschätzen zu können. Bei den offenen Auktionen englischer und holländischer Art sind die Gebote offen und gewähren daher einen ausreichenden Überblick über die gegenwärtige Marktpreis-Situation. Dagegen erlauben die verschiedenen Submissionsauktionen durch ihre verdeckten Gebote eine preisliche Einordnung des eigenen Gebots erst nach dem Zuschlag. Nach [MD88] ist das einzige effiziente Allokationsverfahren die Zweitpreis-Submissionsauktion, da es strategisches Verhalten ausschließt. Die Vickrey-Auktion erteilt dem Bieter mit dem höchsten erzielbaren Gewinn – also dem höchsten Nutzen – den Zuschlag und ist daher eher innerhalb eines einzigen Subnetzes mit gemeinsamer administrativer Kontrolle geeigneter als die anderen Verfahren, die besser zur Zuteilung unter unabhängigen, konkurrierenden Gruppen geeignet sind.

Bilaterale Aushandlung

Wie der Name schon sagt, erfolgt bei diesem Verfahren die Bestimmung des Preises direkt oder indirekt (über einen Mittelsmann) zwischen den beteiligten Partnern am Markt vorbei, wobei die Preisbestimmung weder öffentlich noch transparent ist. Falls genügend Informationen über Angebot und Nachfrage im Markt verfügbar sind, ist der direkte Kontakt günstiger, und im Fall keiner oder zu komplexer Informationen erscheint die Aushandlung über einen Makler geeigneter, der Informationen zur Verfügung stellen oder filtern kann. Problematisch wird die Verhandlungsphase falls ein Nachfrager auf mehr als einen Anbieter zur Aufgabenerfülung angewiesen ist. In diesem Fall ergeben sich Abhängigkeiten, die sich auf die Preise auswirken.

Ausschreibung und Einschreibung

Eine öffentliche und transparente Preisbildung über eine einseitige Marktveranstaltung wird als Ausschreibung, falls nur ein Nachfrager und als Einschreibung, wenn nur ein Anbieter auf einer Marktseite vorhanden ist, bezeichnet.

3.2.7 Handel von Ressourcen

Der Einsatz marktbasierte Zuteilungsmechanismen stellt bereits innerhalb eines einzelnen Prozeßsystems einen Fortschritt bei der Zuteilung mehrerer gleichberechtigter Ressourcen gegenüber prioritätenbasierten Verfahren dar. Diese Vorteile lassen sich aber auch leicht auf verteilte Prozeßsysteme verallgemeinern, wenn das Marktmodell um marktbasierten Im- und Export von Ressourcen erweitert wird. Falls nämlich das Preisniveau in externen Märkten höher oder niedriger liegt, kann der Systemgewinn durch Export von teueren Ressourcen erhöht bzw. die Systemkosten durch Import preiswerter Ressourcen erniedrigt werden. Im Fall von gleichen Preisniveaus können Überbestände von Systemressourcen, die nicht angesammelt werden können und daher eine Haltbarkeit von Null besitzen, abgebaut und damit deren Fixkosten gesenkt werden. Die Erwirtschaftung eines Handelsüberschusses von Ressourcen bietet also trotz der fehlenden Haltbarkeit von Ressourcen die Möglichkeit, Vorratshaltung durch die Schaffung von Reserven zu betreiben. Ressourcenüberschuß kann also exportiert und bei akutem Bedarf wieder importiert werden.

3.3 Theoretische Analyse

Das Hauptaugenmerk bei marktbasierter Zuteilungsverfahren ist, ob sie sowohl im Hinblick auf die ökonomische Ressourcenverteilung als auch der hierfür benötigten Kosten effizient arbeiten, kurz ob sie ökonomisch und komplexitätsmäßig effizient sind, um die hohe Teilnehmerzahl und Dynamik elektronischer Märkte zu verkraften. Die Eigenschaften marktbasierter Verfahren sind relativ eng an Parameter der Marktteilnehmer gekoppelt, die im folgenden genauer untersucht werden.

3.3.1 Annahmen

Der Handel in einem Marktsystem wird durch einen *Marktmechanismus* gesteuert, der eine Menge von handelbaren Gütern, Währungen und Regeln zur Kommunikation von Käufern und Verkäufern bezeichnet. Ein Markt kann als Realisierung einer mehrründigen Simulation betrachtet werden, deren Entscheidungen innerhalb einer Runde, die unter Umständen auf unvollständigen Informationen beruhen, die nachfolgenden Runden beeinflußt. Bei den Teilnehmern handelt es sich um opportunistische Käufer und Verkäufer, die sich dynamisch am Markt an- und abmelden, um ihre individuelle Nutzen-Funktion zu maximieren. Da die Teilnehmer die Nutzen-Funktionen ihrer Gegenüber nicht kennen, basieren ihre Entscheidungen auf unvollständigen Informationen. Um die Simulation genauer formal zu analysieren, ist es sinnvoll ihm Folgenden von einigen vereinfachenden Annahmen auszugehen.

1. Käufer und Verkäufer haben keine Information über die Anzahl, die Nutzen-Funktionen und die Maschinen ihrer Wettbewerber.

2. Die Gebote bzw. Forderungen sind während einer Auktionsrunde fix, dazwischen frei beweglich

3. Die Auktionsrunden finden in äquidistanten Zeitintervallen von d Einheiten statt

Das Hauptinteresse in den folgenden theoretischen Analysen ist die soziale Effizienz der Simulation und ihrer Methoden, also ob sie geeignet sind, knappe Ressourcen fair und effizient zu verteilen. Zum Nachweis einer solchen Effizient verwenden wir das *Kompensationskriterium*, das auf das bereits vorgestellte Kriterium der *Pareto-Optimalität* zurückgreift. Dazu wird überprüft, ob eine Zuteilung den Gesamtwohlstand einer Ökonomie maximiert, indem der Nutzen, den jeder Marktteilnehmer von einer bestimmten Zuteilung A erwirtschaftet, aufsummiert wird. Danach wird eine andere Zuteilung A' gesucht, die von allen Teilnehmern der Zuteilung A vorgezogen wird. Existiert keine solche bessere – *pareto-bevorzugte* – Zuteilung, ist die Allokation A pareto-optimal [Bog94]. Das bedeutet gleichzeitig, daß die Zuteilung den Wohlstand der Ökonomie maximiert.

Die gehandelten Güter sind verschiedene Ressourcen, die in Grundeinheiten bzw. Grundwährungen eingeteilt sind. Zur Vereinfachung gehen wir im Folgenden davon aus, daß Käufer ihren Nutzen direkt aus diesen Grundeinheiten ableiten, obwohl diese Annahme in der Praxis in der Regel nicht erfüllt sind. Genauso sei angenommen, daß der Nutzen der Verkäufer nur von den erzielten Grundwährungen abhängt. Daher kann ebenfalls angenommen werden, daß der sich ergebende Gleichgewichtspreis sowohl die Produktionskosten der Verkäufer als auch den möglichen Gewinn der Käufer widerspiegelt.

3.3.2 Marktteilnehmer

Der Käufer Ein Käufer i besitze einen aus einer Wahrscheinlichkeit-Verteilung F gezogenen Typ V_i^0, der seine *Bewertung* einer Ressourceneinheit zum Zeitpunkt 0 bezeichnet. Die Bewertung eines Käufers sinkt mit der Zeit und sei mit $V_i^t = \alpha^t V_i^0$, wobei $0 \leq \alpha \leq 1$ eine allen Käufern gemeinsame Verfallskonstante sei. Weiterhin sei angenommen, daß ein Käufer, der zum Zeitpunkt t L Ressourceneinheiten zum Preis p erwirbt, einen Nutzen von $L(V_i^t - p)$ erzielt [Reg98]. Aus dieser Formulierung des Nutzen folgt, daß Käufer risikoneutral am Markt teilnehmen.

Der Verkäufer Ein Käufer i besitze analog einen aus einer Wahrscheinlichkeits-Verteilung G gezogenen Typ e_i, der seine *Kosten* für eine Ressourceneinheit bezeichnet. Ein weiterer Parameter des Verkäufers ist die Geschwindigkeit seiner Maschine in z_i Ressourceneinheiten pro Sekunde. Die Geschwindigkeit ist ebenfalls aus einer Wahrscheinlichkeits-Verteilung H gezogen. Darüberhinaus sei angenommen, daß ein Verkäufer, der zum Zeitpunkt t L Ressourceneinheiten zum Preis p verkauft, einen Gewinn von $L(\frac{e}{z-p})$ erzielt [Reg98]. Aus dieser Formulierung des Gewinns folgt, daß Verkäufer ebenfalls risikoneutral am Markt teilnehmen.

3.3.3 Auktionsverfahren

Die Zweitpreis-Submissionsauktion

Bei der Zweitpreis-Submissionsauktion (auch als wiederholte Vickrey Auktion bezeichnet) geht das gehandelte Gut an das höchste Gebot zum Preis des zweithöchsten Gebots. Bei einründigen Auktionen bilden sich im allgemeinen strategische Verhaltensweisen der Anbieter und Bieter heraus, die die Marktentwicklung dominieren. Um diese Tendenzen zu verhindern, werden mehrere Runden von Submissionsauktionen angewandt, bei denen wir von folgenden Annahmen ausgehen:

1. Die Verkäufer sind *symmetrisch*, das heißt sie haben alle gleich leistungsfähige Maschinen mit einer Geschwindigkeit von z Einheiten / Sekunden und die gleichen Kosten e zur Produktion von Ressourcen pro Zeiteinheit. Die Verkäufer bestehen demnach aus einer Menge von Kopien eines einzigen Verkäufers.

2. Die Käufer sind ebenfalls *symmetrisch*, das heißt jeder Käufer zieht seinen Nutzen zum Zeitpunkt 0 aus der gleichen Wahrscheinlichkeits-Verteilung F.

3. Obwohl Kaufgebote jederzeit eintreffen können, gehen wir dennoch davon aus, daß alle Käufer dem Markt zum gleichen Zeitpunkt 0 beigetreten.

Einrunden-Submissionsauktionen

Aus der Auktionstheorie [Vic61] ist bekannt, daß die für Käufer dominante Strategie in einer Zweitpreis-Auktion ist, den wahren Nutzen eines Gutes zu offenbaren. Da der zu zahlende Preis nicht vom eigenen Gebot abhängt, sind die Käufer sogenannte *Preisnehmer* und können durch die Angabe von falschen Nutzen keine Vorteile bei der Auktion erzielen. Die Submissionauktion erfüllt daher die Bedingungen der kompatiblen Anreize (*incentive compatibility*) und des strategischen Gleichgewichts (*dominant strategy equilibrium* [Reg98]).

Wenn die Käufer ihre Gleichgewichtsstrategie einsetzen, ist eine einzige Runde einer Submissionsauktion sozial effizient, das heißt der Käufer mit dem höchsten Nutzen, mit Käufer i zum Zeitpunkt t bezeichnet, erhält den Zuschlag für das angebotene Gut. Das durch den Handel erzielte Gewinn hängt vom zweithöchsten Gebotes zV_j^t ab, lautet also für den Verkäufer $zV_j^t - e$ und für den Käufer $z(V_i^t - V_j^t)$ [Reg98]. Daher ist der Gewinn der gesamten Ökonomie $zV_i^t - e$ der maximal mögliche Gewinn aller potentiellen Abschlüsse einer Auktionsrunde.

Mehrrunden-Submissionsauktionen

Da in einer wiederholte Vickrey Auktion mehrere Auktionsrunden veranstaltet werden, gelten die Eigenschaften einer Einrunden-Auktion unter Umständen nicht mehr. Im Folgenden zeigen wir, daß auch eine mehrründige Submissionsauktion effizient ist.

Angenommen, daß Verkäufer und Käufer ihren wahren Nutzen offenbaren, also nichtstrategische Marktteilnehmer sind. Es sei $x = (x_o, ..., x_r)$ die Folge von Allokationen, die vom Marktmechanismus während der $r*d$ Sekunden ihrer Laufzeit vergeben wurde, wobei

$$x_i = \begin{cases} i & : \textit{falls dem Kaufer } i \textit{ zur Zeit } k*d \textit{ nicht zugeteilt wurde,} \\ 0 & : \textit{falls kein Kaufer zur Zeit } k*d \textit{ zugeteilt wurde.} \end{cases} \tag{3.3.1}$$

Die obige Formel enthält keine Angaben darüber, wer die jeweiligen Verkäufer der Auktionsrunden waren, da die Verkäufer per Definition identisch sind.

Um zu prüfen, ob diese Allokation x den Gesamtwohlstand der Ökonomie maximiert, müssen wir nach einer anderen Zuteilung x' suchen, die von allen Beteiligten x vorgezogen wird, also *pareto-bevorzugt* ist. Dabei sind nur legale Zuteilungen interessant, also eine Allokation die folgende Bedingungen erfüllen:

1. Verkäufe erfolgen zu diskreten Zeitpunkten, die Vielfache von d sind
2. Die Regeln des Marktmechanismus sind erfüllt
3. Die Bezahlung erfolgt nach Anzahl der verkauften Gütereinheiten, also $L * p$

Unter Beachtung der gemachten Annahmen existiert keine andere legale Allokation $x' \neq x$ für die gegebene Eingabe, die *pareto-bevorzugt* gegenüber x ist.

Der Beweis dieser Behauptung ist intuitiv nachvollziehbar und in [Reg98] zu finden. Es folgt unmittelbar, daß die vom Marktmechanimus gewählte Zuteilung pareto-optimal ist und den Wohlstand der Ökonomie maximiert.

Doppelauktionen

Aus dem vorigen Abschnitt ging hervor, daß Zweitpreis-Submissionsauktionen zwar im Sinne von Pareto-Optimalität effizient sind, aber theoretische und praktische Experimente [PD93][R.W83] haben gezeigt, das sie unter Umständen sozial ineffizient sind, da sie aufgrund der geschlossenen Gebote keinen Wettbewerb zwischen den Marktteilnehmern fördern. Eine Möglichkeit, Wettbewerb in den Marktmechanismus einzuführen, sind Doppelauktionen.

Einfache Doppelauktionen – Bei diesem Auktionstyp werden sowohl Anbieter als auch Nachfrager nach der Höhe ihrer abgegebenen Gebote in zwei Prioritätenschlangen eingeordnet. Nachfrager mit höheren Geboten werden weiter oben in der Käuferliste eingereiht, während Anbieter weiter oben in der Verkäuferliste eingereiht werde, je niedriger ihre geforderter Preis ist. In jeder Runde der Doppelauktion wird stets der höchste Bieter mit dem niedrigsten Anbieter zusammengebracht. Im zweiten Teil der Doppelauktion einigen sich die beiden Beteiligten zum Verkauf zum Durchschnittspreis ihrer beiden Gebote. Dieses Verfahren ist einfach und rechnerisch unaufwendig

zu implementieren, führt aber pro Runde nur ein einziges Anbieter-Nachfrager-Paar zusammen.

Die einfache Doppelauktion erinnert an die gleichzeitige Ausführung von zwei Erstpreis-Submissionsauktionen. Wie bei diesem Auktionstyp existiert auch hier kein dominantes strategisches Gleichgewicht. Da es sich außerdem um kein Verfahren mit sofortiger Offenbarung der Absichten handelt, ist es unter Umständen ineffizient. Da Marktteilnehmer nun Einfluß auf den erzielbaren Endpreis haben, kann ihr Handeln von strategischen Überlegungen geprägt sein, das heißt sie geben ihre wahre Bewertung der Güter nicht preis. So kann es passieren, daß alle Forderungen über sämtlichen Geboten liegen, obwohl es einen Käufer geben würde, der das Gut höher bewertet als der Verkäufer. Außerdem ist das Verfahren von relativ hoher Komplexität ($O(n)$), da in jeder Runde nur eine Transaktion stattfindet [Reg98].

Clearinghouse-Doppelauktionen – Die Clearinghouse Doppelauktion (auch als k-DA bezeichnet) entspricht der normalen Doppelauktion, außer daß in jeder Runde mehr als ein Anbieter-Nachfrager-Paar bestimmt wird. Zu bestimmten Handelszeiten werden sämtliche Käufergebote und Verkäuferforderungen zur Bestimmung der Angebots- und Nachfragekurven verwendet. Der aus dieser Kurve bestimmte Gleichgewichtspreis p_0 wird für alle Transaktionen dieser Runde verwendet. Käufer, die mehr als p_0 bieten werden mit Verkäufern zusammengebracht, die weniger als p_0 verlangen und damit der Markt bereinigt.

Der Bereinigungspreis p_0 wird in jeder Runde wie folgt bestimmt. Die Kaufgebote werden in absteigender Folge angeordnet, $V_1 \geq V_2 \geq ... \geq V_n$, und die Verkaufsangebote in ansteigender Reihenfolge $A_1 \leq A_2 \leq ... \leq A_m$. Die Reihen werden durch fiktive Gebote $V_{n+1} = 0$ und $A_{m+1} = \infty$ zur Vollständigkeit ergänzt. Der Marktmechanismus bestimmt nun das erste Paar (V_{j+1}, A_{j+1}), das die Bedingung $A_{j+1} > V_{j+1}$ erfüllt. Die gehandelte Gütermenge ist dann j und der Bereinigungspreis beträgt

$$p_0 = \frac{1}{2(V_j + A_j)} \tag{3.3.2}$$

daß heißt es liegt eine k-DA mit dem Faktor $k = \frac{1}{2}$ vor [Reg98].

Clearinghouse Doppelauktionen werden in einigen realen Märkten wie Aktien- und Warenmärkten eingesetzt. Im Folgenden gehen wir wie im Fall von einfachen Doppelauktionen von einigen Annahmen aus. So seien alle Käufer und Verkäufer symmetrisch und ihre Nutzen-Funktionen konstant. Außerdem seien alle Verkäufermaschinen identisch.

Theoretische Eigenschaften der einründigen k-DA Wie wir gesehen haben, besteht jede Runde des Marktmechanismus aus einer k-DA mit einen $k = \frac{1}{2}$. Da jedoch die Marktbeteiligten Einfluß auf den Bereinigungspreis p_0 haben und aus diesem Grund nicht ihre wahren Nutzen offenbaren, ist die Nachfragekurve niedriger als gewöhnlich und die Angebotskurve entsprechend höher als eine Kurve, die bei nicht-strategischen Bieten entsteht. Aus diesen Überlegungen folgt, daß ein Einrunden-k-DA kein direkter Offenbarungsmechanismus ist und demnach die Zuteilung nach einer Auktionsrunde nicht unbedingt effizient ist.

Trotz dieser Nachteile besitzt die k-DA jedoch eine bedeutende Eigenschaft, die ihren Einsatz rechtfertigt. So sinkt bei zunehmende Anzahl von Käufern und Verkäufern der Anreiz, einen falschen Nutzen zu berichten, und damit strebt der Gesamtgewinn zum maximal möglichen Wert. Diese Eigenschaft wurde von Rustichini, Satterthwaite und Williams [RSW90] nachgewiesen. In ihrem Nachweis gehen die Autoren davon aus, daß die Nutzen von Käufer i, v_i, und Verkäufer j, a_j Wahrscheinlichkeits-Verteilungen F und G entnommen sind. Das Kaufgebot V und das Verkaufsgebot A hängen jeweils von v_i und a_j ab. Der Doppelauktions-Mechanismus bewegt nun Käufer dazu, niedrigere Gebote als ihr wahrer Nutzen abzugeben und Verkäufer höhere Preise als ihr Nutzen zu verlangen. Die Autoren können jedoch zeigen, daß die Differenzen $v_i - V_i$ und $A_j - a_j$ von der Ordnung $O(\frac{1}{(m+n)})$ sind, wobei m und n die Anzahl der Käufer und Verkäufer bezeichnet.

Rustichini, Satterthwaite und Williams definieren den Begriff des *erwarteten Gewinns* aus dem Verkauf als Erwartungswert des realisierten Gewinns wenn die Nutzen der Marktteilnehmer den Verteilungen F und G folgen. Als *potentieller Gewinn* wird dagegen als Erwartungswert des Gewinn definiert, falls die Teilnehmer anstelle ihres wahren Nutzens eine anderen Wert offenbaren. Die *relative Effizienz* eines ökonomischen Gleichgewichts ergibt sich dann aus dem erwarteten Gewinn im Verhältniss zum potentiellen Gewinn. Die Autoren zeigen, daß eine Konstante k existiert, die nur von F und G abhängt, so daß die relative Effizienz des Gleichgewichts mindestens

$$1 - \frac{k}{(m-n)}^2 \tag{3.3.3}$$

beträgt. Damit ist gezeigt, daß innerhalb der Annahmen das Ergebnis einer k-DA nahe an einer optimalen Allokation liegt.

Theoretische Eigenschaften mehrründiger k-DA Komplikationen ergeben sich bei Doppelauktionen, falls mehrere Runden der Vickrey Auktion durchgeführt oder mehrere Güter gehandelt werden, die die Ergebnisse des vorhergehenden Abschnittes trüben. Äußert ein Käufer den Wunsch, mehrere Einheiten eines Gutes zu erwerben, wird dies durch mehrere identische Kauforders über eine Gütereinheit modelliert, die die theoretischen Aspekte der k-DA beeinflußt und unter Umständen zu ineffizienter Allokation führt. Als Illustration sei folgendes Beispiel herangezogen [Reg98].

Es werden $N > 2$ Kauforder und N Verkaufsorder während der Laufzeit $N * T$ Sekunden dieser Szenerie abgegeben. Der wahre Wert für eine Gütereinheit beträgt $(N+2) - i$ für den Käufer und $(N+1) - i$ für den Verkäufer und diese Bewertungen bleiben für die Handelsdauer konstant. Alle T Sekunden wird eine Doppelauktion abgehalten. Käufer i und Verkäufer i registrieren ihre Orders zum Zeitpunkt $i + (i-1)T$, so daß in jeder Runde eine Transaktion zwischen Käufer und Verkäufer i zustande kommt. Der gesamte Gewinn des Betrachtungszeitraumes ist daher $N * (Wert - Kosten)$, also N. Der maximale, von einem allwissenden Mechanismus erzielbare Gewinn beträgt jedoch

$$2 * \sum_{i=0}^{N*T} (i - \frac{N}{2}) \tag{3.3.4}$$

Mendelson [Men82] zeigt, daß das Verhalten von Marktteilnehmern in einem Clearinghouse Marktmechanismus nicht strategisch ist. Dazu geht er von gleich-verteilten Geboten

innerhalb eines Intervalls $[0, m]$ aus und modelliert die Zeitpunkte des Markteintritts von Teilnehmern als identische Poisson-Verteilungen mit Raten von αm, wobei α ein konstanter Faktor ist. Er nimmt weiter an, daß Angebot und Nachfrage unabhängige Poisson-Verteilungen mit der Rate $\theta = \alpha T$ sind. Diese Rate bezeichnet den Erwartungswert der Kauf- und Verkaufsgebote pro Preiseinheit. θ wird dabei als *Gebotsintensität* bezeichnet. Er kann nachweisen, daß die Varianz des Räumungspreises p_0 durch

$$var(p_0) = \frac{m}{8\theta} - \frac{1}{32\theta^2}(1 - e^{-2\theta m})(3 - e^{-2\theta m}) \tag{3.3.5}$$

gegeben ist.

Aus diesem Ergebnis folgen zwei Eigenschaften:

1. Der Räumungspreis stellt eine gute Abschätzung für den "wahren" Preis, also den Schnittpunkt der Angebots- und Nachfragekurve dar. Denn für den Erwartungswert des Räumungspreises gilt $E(p_0) = \frac{m}{2}$.

2. Zweitens ist die Varianz von p_0 eine mit θ fallende Funktion. Geht θ gegen Unendlich, bewegt sich die Varianz relativ schnell gegen Null. Auch eine Erhöhung de Handelsintervalls T sorgt für eine Verringerung der Varianz, da $\theta = \alpha T$ gilt.

Aus den beiden Eigenschaften folgt, daß der Räumungspreis dem "wahren" Gleichgewichtspreis sehr nahe kommt und insbesondere dynamischen, kurzfristigen Veränderungen der Angebots- und Nachfragekurven (zum Beispiel bei einer Veränderung der anzahle der Marktteilnehmer m) mit steigendem θ oder T schnell folgen kann.

3.4 Zusammenfassung

Dieser Abschnitt beschäftigte sich mit den Besonderheiten und Unterschieden marktbasierte Ressourcenzuteilung im Vergleich zu traditionellen prioritätenbasierten Zuteilungsverfahren. Es zeigte sich, daß diese Verfahren geeignet sind, mehrere konkurrierende Ressourcen – wenn eine ideale Zuteilung unmöglich ist – zumindest effizient (pareto-optimal) und gleichberechtigt zuzuteilen. Dieser Vorteil wird mit einer rechnerisch aufwendigeren Entscheidungsfindung erkauft, die bei der Initialisierung eines Marktsystems, bei der Gleichgewichtsbildung von Angebot und Nachfrage und bei der Ressourcenzuteilung selbst zu Tage tritt.

Es wurde zudem gezeigt, daß die vorgestellten Auktionsverfahren unterschiedliche Eigenschaften im Hinblick auf strategische Gebote aufweisen, die eine effiziente Zuteilung behindern und daher möglichst ausgeschlossen werden sollten. Das einzige effiziente Zuteilungsverfahren, das strategische Gebote, die nicht den wahren Nutzen des Bieters offenbaren, aussschließt, ist die Zweitpreis-Submisionsauktion, die sich allerdings eher für untereinander bekannte Parteien als für anonyme Teilnehmer eignet. Offene Auktionen englischer und holländischer Art gewähren dagegen einander nicht vertrauten Vertragspartnern einen ausreichenden Überblick über die gegenwärtige Marktpreis-Situation. Durch die Offenlegung der Gebote ist allerdings strategisches Bieten leicht möglich, dafür wird durch die Offenlegung auch der Wettbewerb zwischen den Parteien gefördert, der wiederum Vorraussetzung für eine sozial effiziente Zuteilung ist. Anderenfalls, falls also kein

transparenter Markt in Form offener Gebote vorhanden ist, kommt unter Umständen der Käufer mit dem höchsten Nutzen für eine Ressource nicht zum Zug.

Besonders problematisch ist jedoch die Tatsache, daß die Simulation von intelligentem Teilnehmerverhalten mit Hilfe von selbständigen Agenten komplex und daher im Hinblick auf die Transaktionskosten sehr kostspielig ist. Simple fallbasierte Algorithmen sind jedoch nicht in der Lage, die differenzierten Entscheidungen nachzubilden, wie sie beim Abwägen der persönlichen Kosten-Nutzen Relationen auftreten,. Zudem besteht bei agentenbasierten elektronischem Handeln stets die Gefahr, daß es aufgrund der fallsteuerten Agenten zu Rückkoppelungen und zu chaotischem Verhalten kommt und der Markt wegen des fehlenden regelnden Eingriffs eines menschlichen Überwachers zusammenbricht. Allein der zur Realisierung der Agenten notwendige Rechenaufwand – von Entwicklung und Sicherheitsaspekten abgesehen – verschlingt einen Großteil des durch seinen Einsatz erzielbaren Nutzens. Aus diesen Gründen wurde bei dem im Rahmen dieser Arbeit entwickelten elektronischen Marktes auf eine automatische agentenbasierte Steuerung verzichtet. Stattdessen wird den Marktteilnehmern eine Rechnerunterstützung bei allen Transaktionsphasen zuteil. Die Unterstützung läßt sich mit wenig rechenintensiven Algorithmen implementieren, da die Teilnehmer selbst über ihre Päferenzen entscheiden und für einen koordinierten Geschäftsverkehr sorgen.

3.5 Elektronische Märkte

Wegen der Ähnlichkeit zu realen Systemen wurden bereits in der Vergangenheit Anstrengungen unternommen, um marktbasierte Verfahren auf verteilte Rechensysteme zu übertragen und deren Eignung hierfür abzuschätzen. Da Marktmechanismen wie Auktionen und Gleichgewichtspreise ihre Eignung zur effizienten Zuteilung knapper Ressourcen in menschlichen Gesellschaften nachgewiesen haben, erscheint ein solcher Transfer durchaus sinnvoll [MD88]. Im folgenden Abschnitt finden sich, nach einer kurzen Begriffsfindung der für diesen Abschnitt relevanten Definitionen, einige dieser Informationsökosysteme näher vorgestellt und bewertet.

3.5.1 Begriffsfindung

Definition 3.24 (Telematik) *Unter Telematik versteht man eine Infrastruktur zur entfernten Ausübung von Kontrolle über räumliche Entfernungen.*

Definition 3.25 (Elektronischer Markt) *Bei einem elektronischen Markt handelt es sich um einen mit Hilfe eines Telematik-Systems realisierten Markt, das alle Phasen der Handelstransaktionen unterstützt.*

Aus der Realisierung über ein Telematik-System ergibt sich eine grundlegende Eigenschaft elektronischer Märkte, die Unabhängigkeit von Ort und Zeit bzw. die Globalität. Über das Telematik-System wie zum Beispiel das WWW oder das Telefonnetz hat ein Marktteilnehmer von jedem Ort und jederzeit Zugriff auf den Markt, der damit zu einem globalen Markt wird. Reale Märkte, auch elektronisch unterstützte, benötigen stets einen realen lokalen Marktplatz.

Definition 3.26 (Elektronischer Marktplatz) *Bei einem elektronischen Marktplatz handelt es sich um einen mit Hilfe eines Telematik-Systems realisierten Marktplatz, auf dem über das Telematik-System immaterielle Güter und Dienstleistungen ausgetauscht werden.*

Ein elektronischer Marktplatz umfaßt einen elektronischen Markt und wird im Verlauf dieser Arbeit als Begriff für einen vollständigen Markt im engeren Sinne verwendet, das heißt ein Telematik-System wird nicht nur unterstützend, sondern für sämtlich Aspekte aller marktlichen Transaktionsphasen – also Informations-, Vereinbarungs- **und** Abwicklungsphase – verwendet.

3.5.2 Implementierungen elektronischer Märkte

Drexler und Miller's *Escalator* Algorithmus

Drexler und Miller stellen in [MD88] ein Modell zur Prozessorzuteilung mit Hilfe von Auktionen vor, der als *Escalator*-Algorithmus (Rolltreppe) bezeichnet wird. Der Algorithmus erlaubt die Priorisierung von Prozessen bzw. Jobs und garantiert gleichzeitig, daß jeder Job irgendwann zugeteilt wird, eine Aushungerung also nicht möglich ist.

Das Rolltreppen-Verfahren teilt den Prozessor stets an den höchsten Bieter zu und überläßt diesem die Kontrolle, das heißt der Prozess rechnet ohne Präemption bis zur Vervollständigung der gewünschten Berechnung. Dies entspricht den üblichen, bereits vorgestellten anderen Auktionsverfahren.

Der Auktionsmechanismus ist jedoch vollständig von anderen Verfahren verschieden. Jeder Job wird auf eine Stufe der Rolltreppe "gesetzt", das heißt sein Gebot wird linear mit der Zeit erhöht, bis er als meistbietender Job zugeteilt wird. Die Autoren weisen in ihrer Arbeit nach, daß in diesem Modell jeder Job, der auf die Rolltreppe gesetzt wird, auch schließlich ausgeführt wird.

Der Startwert des Aufzugs und seine Steigung bestimmen in gewisser Weise die Dringlichkeit oder Priorisierung eines Jobs. Da diese Parameter jedoch nicht durch Marktkräfte bestimmt wurden, tragen sie auch keine Bedeutung für den Nutzen dieses Jobs in der Gesamtökonomie. Die Steigerung der Gebote erfolgt automatisch und reflektiert keine tatsächliche Erhöhung der Wichtigkeit bei der Erfüllung dieses spezifischen Jobs. Da die Bedingung der Nicht-Aushungerung in einer Ökonomie keine Rolle spielt, tragen die automatisch generierten Gebote nicht die beabsichtigte Bedeutung, insbesondere existiert keine Information über den erwarteten Beitrags eines Jobs zum Gesamtwohlstand. Aus diesem Grund ist der *Escalator* Algorithmus zur effizienten Ressourcen-Allokation nicht geeignet.

Enterprise

Enterprise [MFGH88] ist ein dezentrales, marktbasiertes Scheduling-Verfahren zur Lastverteilung in verteilten Rechenumgebungen, das auf einer Reihe von Ankündigungen, Geboten und Zuschlägen (*announcement, bid, award*) beruht.

1. Ankündigung – In der Ankündigungsphase teilt ein Client allen teilnehmenden Anbietern – in Enterprise als *contractors* bezeichnet – den Wunsch nach Bearbeitung

einer Aufgabe samt einer Beschreibung der Aufgabe, einer Rechenzeitabschätzung sowie einer Priorität mit.

2. Gebot – Untätige Kontraktoren antworten auf diese Anfrage mit einem Gebot, das das voraussichtliche Ende der Berechnung dieser Aufgabe enthält.

3. Zuschlag – Nach einer festgelegten Auswahlzeit bestimmt der Client aus allen eingegangenen Geboten das für ihn günstigste, das üblicherweise dasjenige mit der geringsten Bearbeitungszeit ist.

In einer Reihe von Simulationen konnte Malone [MFGH88] nachweisen, daß die Verwendung des Enterprise Systems auf verteilten Netzen von 5 bis 8 Rechnern zu signifikanten Laufzeitverbesserungen gegenüber der *lokalen* Bearbeitung auf einem einzelnen Arbeitsplatz-Rechner führte, die auch bei sehr unzuverlässigen Informationen über die notwendige Rechenzeit und Prioriät der Jobs aufrecht erhalten wurde. Bei Verwendung größeren Maschinenparks verflachte der Gewinn des Enterprise Systems jedoch wieder.

In der Praxis litt Enterprise an den geringen Schutzmechanismen damaliger Arbeitsplatz-Rechner und einiger grundlegender Designmängel. Das System stellte unter anderem keine Marktpreise zur Verfügung . Der Anwender konnte über den Preis nicht zwischen teueren, dafür schnelleren und preiswerten, aber langsameren Maschinen abwägen, sondern mußte seine Wünsche über künstliche Prioritäten ausdrücken, die über eine Reihe von Heuristiken – unter anderem die Ausführungszeit – bestimmt waren.

Popcorn

Das Popcorn System [RN98] besitzt nach den obigen Definitionen sowohl einen Markt als auch einen Marktplatz. Der elektronische Marktplatz folgt dem üblichen dreistufigen Modell aus Nachfragern (*buyer*), Anbietern (*seller*) und dem Markt selber (*market*).

Güter Die auf diesem Markt gehandelte Rechenleistung wird zur Verarbeitung von parallelen Java Applets verwendet, die ihren Parallelismus durch die Abspaltung nebenläufiger Handlungsfäden, sogenannter *Computelets*, erzielen. Dieses als Popcorn Programmier-Paradigma bezeichnete Verfahren unterstützt grobgranulare parallele Anwendungen mit einem günstigen Verhältnis zwischen Rechen- und Kommunikationsaufwand. Als Einheit der gehandelten Rechenzeit werden sogenannte *Java Operations* (JOPs) verwendet, die als ein Mix verschiedener Java Operationen definiert sind und deren Laufzeiten über einen Mikro-Benchmark ermittelt werden, das im Java Applet enthalten ist. Da dieser zusätzliche Benchmark wertvolle Rechenzeit kostet, ist allerdings auch die Abrechnung über Computelets vorgesehen.

Währungen Als Währungseinheit im Popcorn Markt ist das sogenannte *Popcoin* definiert, mit der sämtliche Transaktionen abgerechnet werden. Jeder am Popcorn System teilnehmende Benutzer besitzt ein Popcoin-Konto, mit dem der Kauf oder Verkauf von Rechenzeit vom oder an das System abgerechnet wird. Es existieren jedoch keine Konvertierungen in andere, insbesondere reale, Währungen.
Jede Bearbeitung eines Computelets stellt nun eine Transaktion im Markt dar. Der Programmierer eines Popcorn Applets kauft also Rechenzeit (Popcoins), um sein Problem

zu lösen. Dazu enthält jedes Computelet ein Vertragsobjekt, das alle für die Transaktion benötigten Parameter wie Preise, Bezahlung per JOP oder Computelet und den Ort des Markts selbst, enthält.
Der Verkauf von Rechenzeit gegen Popcoins erfolgt durch den Besuch einer Web-Seite mit anschließender Authentifizierung. Sobald die persönlichen Kontodaten eingegeben wurden, beginnt das eingebettete Applet seine Arbeit und die Entlohnung mit Popcoins.

Der Markt Die Hauptaufgabe des Popcorn-Marktes ist es, als zentrale Anlaufstelle und Vermittlungs-Instanz für Anbieter und Nachfrager von Rechenzeit zu dienen. Dies macht ihn allerdings zugleich zu einem Kommunikationsengpaß des Popcorn Systems, mit dem der Betrieb steht und fällt. Physikalisch besteht der Markt aus einem Server, den Nachfrager ansprechen können, und Web-Seiten, die Anwendungs-Applets für Anbieter enthalten.

Popcorn besitzt drei Verfahren zur Vermittlung zwischen Angebot und Nachfrage, die alle durch zwei Eigenschaften geprägt sind. Die Zuteilungsverfahren sind ökonomisch effizient, das heißt, sie maximieren zu jeder Zeit die globale Nutzen-Funktion des Systems durch die Zuteilung von Rechenzeit an den Teilnehmer mit dem höchsten Nutzen. Eine hierfür notwendige Bedingung ist zweitens, daß die Marktteilnehmer bei den Geboten ihre wahre Nutzen-Funktion offenbaren (*incentive compatibility* [MD88]). Dies verhindert taktische und strategische Gebote und ist für automatisches Bieten besser geeignet. Die in Popcorn eingesetzten Verfahren sind die Zweitpreis-Submissionauktion, die versiegelte Doppelauktion sowie die wiederholte Clearinghouse Doppelauktion (Vgl. 3.2.6)

3.5.3 ReGTime

ReGTime [Zah99] ist eine Implementierung eines elektronischen Marktes zum Handel mit Rechenleistung. Genauer handelt es sich um ein rudimentäres Maklersystem zur Unterstützung bei Handelstransaktionen, die bei der Miete und Vermietung freier oder dedizierter Rechenkapazitäten auf Clustern von Workstations anfallen, die über das Internet erreichbar sind. Potentielle Mieter und Vermieter wie Firmen und Institute nehmen an ReGTime zur Erweiterung ihrer Rechenkapazität und zur Reduzierung ihrer fixen Kosten teil und erhalten am Ende der im folgende beschriebenen Transaktionen ein "nacktes" UNIX Login, das mittels PVM, MPI, DCE oder DQS (*Distributed Queuing System*) genutzt werden kann. Eine weitere Unterstützung oder Hilfe des Mieters bei der Nutzung der erworbenen Rechenkapazität mit Hilfe zusätzlicher Software oder Hilfestellung finden nicht statt.

Markt Bei ReGTime handelt es sich um einen elektronischen Markt, der nur die Kommunikation, nicht aber den Güterfluß unterstützt. Ein elektronischer Marktplatz wird demnach nicht angeboten. Die Kommunikation gliedert sich in Informations-, Vereinbarungs- und Abwicklungsphase.

1. Informationsphase – Die Informationsphase stellt den Makler-Service eines elektronischen Marktes dar, das heißt die Information über und die Vermittlung zwischen Angebot und Nachfrage nach dem zu handelnden Gut. Diese Phase wird in ReGTime rudimentär durch eine Datenbank aller registrierten Anbieter von Rechenzeit

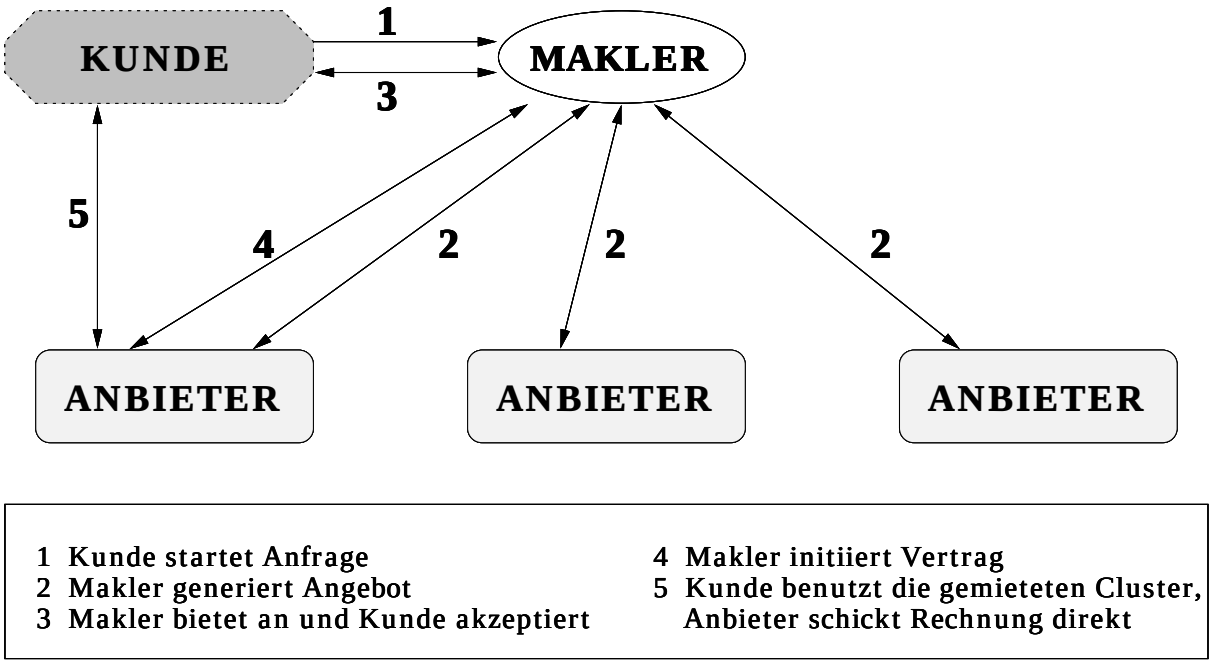

Abbildung 3.6 ReGTime Ablauf [HU97]

mit ihren individuellen zur Verfügung stehenden Rechenkapazitäten, ihren Nebenbedingungen sowie den jeweiligen Preisen unterstützt. Da diese Datenbank jedoch statisch ist, sind sämtliche Daten der Anbieter wie Kapazitäten, Preise und Nebenbedingungen fixiert, die Bildung von auftragsbezogenen Preisen ist damit also nicht möglich.

2. Vereinbarungsphase – Die Vereinbarungsphase umfaßt den Vertrags-Service, also die Aushandlung und den Abschluß eines Vertrages über den Bezug Dienstleistungen. Diese Phase wird in ReGTime durch einen oder mehrere Vertragsentwürfe unterstützt, die als Antwort auf eine Anfrage eines Nachfragers aus den vollständig übereinstimmenden Rechenzeit-Angeboten generiert werden. Konzeptionell entspricht dies einer Erstpreis-Submissionsauktion, das heißt die Vermieter geben pro Nachfrage (Auktion) ein verdecktes Gebot ab. Der Nachfrager kann sich im Vorab nicht über den Markt einen Überblick – beispielsweise über das allgemeine Preis-Niveau – verschaffen und erhält darüberhinaus kein passendes Angebot, falls keines der vorher fixierten Angebote hundertprozentig zu seiner Nachfrage paßt.

3. Abwicklungsphase – Die Abwicklungsphase stellt den Überwachungs-Service – die Einhaltung und Überwachung des abgeschlossenen Vertrags – zur Verfügung. ReGTime richtet dazu die erworbenen Logins ein und öffnet und sperrt die Accounts des Mieters während der Vertragslaufzeit. Anschließend erfolgt eine Erstellung der protokollierten Daten zu einer Rechnung, die per email versandt wird.

Architektur Prinzipiell ist ReGTime für drei mögliche Szenarien – das Fremdabteilungs-, das Partnerschafts- und das Fremdfirmenmodell (Vgl. [Zah99]) –

geeignet, wird aber aufgrund von Sicherheits-, Benutzungs- und sozialen Problemen von seinen Entwicklern nur für Mietverhältnisse zwischen einander vertrauten Partnern empfohlen. Es folgt eine kurze Übersicht der bei Prüfung der Praxistauglichkeit von ReGTime aufgetauchten Probleme.

Nachteile In [HU97] wird von einem möglichen elektronischen Rechenleistungs-Markt stets als einer Vision gesprochen. Die Evaluierung dieser "Vision" und ihrer Infrastruktur, nämlich ReGTime, erwies sich jedoch als problematisch, da keine Tester außerhalb der Entwicklergruppe gewonnen werden konnten. Dieser gravierende Nachteil wurde durch Umfragen innerhalb potentieller Anwender zum Teil abgeschwächt, um Informationen und Erfahrungen über bestehendes Interesse, Kapazitätsüberschüsse und mögliche Anwendungen zu sammeln.

Mangel an Applikationen – Parallelismus als Programmier-Paradigma hat seinen Weg aus den Universitäten und Forschungslabors in die breiten Anwenderschichten noch nicht vollzogen, so daß kaum Anwendungen zur Verfügung stehen, die von zugemieteter Rechenkapazität profitieren können. Jedoch ist zu erwartet, daß die Verfügbarkeit von preiswerter Rechenkapazität die Entwicklung und den Einsatz neuer Anwendungen initiiert.

Mangelnde Benutzerfreundlichkeit – Die Zurverfügungstellung eines bloßen Logins zur Nutzung mittels *remote shell* ist eine schwerwiegende Beeinträchtigung der Nutzungsfreundlichkeit und setzt darüberhinaus das Vorhandensein eines UNIX Betriebssystems auf Anbieterseite voraus. Neben einer komfortableren Schnittstelle zum Zugriff auf gemietete Rechenleistung ist weiterhin eine höhere Transparenz und Unterstützung des Marktes, insbesondere von der Mieterseite her, notwendig um eine Etablierung und Durchdringung eines elektronischen Rechenzeit-Marktes voranzutreiben.

Soziale Probleme – Die Entwickler von ReGTime haben bei der Evaluierung ihres elektronischen Marktes eine Reihe von nicht-technischen und irrationalen Problemen identifiziert, die in [HU97] als soziale Probleme bezeichnet wurden. Diese umfassen die mangelnde Bereitschaft, Rechenzeit auf der "eigenen" Workstation zur Verfügung zu stellen, aus Angst, die Kontrolle und den sofortigen Zugriff auf "seine" Maschine zu verlieren; außerdem das Unbehagen, "seinen" Computer nach außen und damit fremdem Zugriff mit unter Umständen bösen Absichten zu öffnen.

Mangelnde Sicherheit – Im Augenblick ermöglicht ReGTime nur die Authentifizierung der Vertragspartner mittels PGP Signatur. Weitere Sicherheitsmaßnahmen zum Schutz vor böswilligen Angriffen bestehen nicht. Mögliche Schritte zur Erhöhung der Sicherheitshürden sind der Einsatz von *restricted shells*, Java Byte Code sowie die Installation und Ausführung der ReGTime Software unter einer Benutzerkennung anstelle der Superuser Kennung.

Das Fazit der in [Zah99] beschriebenen Erfahrungen mit ReGTime fällt daher zwiespältig aus. Ein Rechenleistungs-Markt wird momentan als nicht realisierbar (mit ReGTime) bezeichnet, gleichwohl aber als Vision mit Zukunft betrachtet.

3.5.4 Java Market

Architektur Bei *Java Market* [AAB98b] handelt es sich um die Implementierung eines elektronischen Marktes und Marktplatzes. Das System besteht aus *Konsumenten*, die *Tasks* zum Verbrauch von Rechenzeit bereitstellen, *Produzenten*, die Rechenzeit zur Verfügung stellen und dem *Java Market*, der zwischen diesen beiden Teilnehmern vermittelt. Dazu verteilt er Tasks unter Berücksichtigung bestimmter Ressourcen-Zuteilungsstrategien auf Produzenten, um seinen Profit zu maximieren. Ein Task verfügt über folgende Eigenschaften: ein Java Applet mit Daten, einen Gewinn (für den Java Market) für die fristgerechte Erledigung, eine Abschätzung der hierfür notwendigen Rechenzeit sowie eine Frist, in der der Task zu erledigen ist.
Ein Produzent ist ein teilnehmender Rechner mit folgenden Eigenschaften: die mit der Überlassung anfallenden Kosten, eine Abschätzung der Leistungsfähigkeit, die Kommunikationkosten, das heißt die Latenz und Bandbreite dieses Produzenten.

Die Kosten-Nutzen-Strategie Java Market umgeht das Problem heterogene und damit nicht vergleichbare Ressourcen mit nicht vergleichbaren Prioritäten zuteilen zu müssen dadurch, daß homogene Kosten an die Ressourcen zugewiesen werden, die von ihrem augenblicklichen Gebrauch abhängen. Genauso werden Jobs unterschiedlicher Prioritäten einem homogenen Nutzen zugeordnet, der davon abhängt, in welcher Zeit diese Arbeit erledigt wird. Kosten und Nutzen werden durch Verwendung der gleichen Währung einheitlich. Die sogenannte Grenzkosten-Strategie zur Ressourcenzuteilung vermittelt einen Job an den Rechner, an dem der Ressourcenverbrauch die geringsten Grenzkosten verursacht. Das aus einer solchen Strategie resultierende Szenario besitzt die geringsten Gesamtkosten. Eine optimale Ressourcen-Zuteilungsstrategie ist nun eine solche, die den Gewinn des Systems – der Nutzen abzüglich der Kosten – maximiert. Neben der Einfachheit dieses Verfahrens is es darüberhinaus noch durch geringe Kosten gekennzeichnet. Der Ressourcenverbrauch der Kosten-Nutzen Strategie ist höchstens um $O(logn)$ höher als eine optimale Zuteilungsstrategie, die die Zukunft kennt.

Kosten – Die Kosten einer Ressource sind eine exponentielle Funktion ihrer gegenwärtigen Auslastung. Eine exponentielle Kosten-Funktion kann bereits als Kontrollmechanismus für den Marktzugang verwendet werden: falls der Nutzen eines Tasks höher als seine Kosten sind, wird der Task zugewiesen, sonst abgelehnt. Der eigentliche Vorteil von Kosten-Funktionen tritt vollständig hervor, falls verschiedene Ressourcen gleichzeitig zugeteilt werden müssen, zum Beispiel ein Job, der c CPU Sekunden **und** m Megabyte Hauptspeicher benötigt. In einem solchen Fall haben herkömmliche Zuteilungsverfahren Schwierigkeiten. Bei der Kosten-Nutzen Strategie werden die Kosten der verschiedenen Ressourcen addiert und der Job auf die Maschine plaziert, auf der die geringsten Grenzkosten anfallen.

Nutzen – Der Nutzen, den ein Job dem System liefert, kann mittels der Grenzkosten ermittelt werden. Kann ein Job auf keiner Maschine einen Gewinn erzielen, wird er abgelehnt oder verzögert. Im einfachsten Fall ist der Nutzen gleich der Priorität eines Jobs: niedrig priorisierte Jobs werden abgelehnt oder verzögert, Jobs mit hohen Nutzen werden eher zugeteilt und Jobs mit Nutzen, der höher als alle möglichen Kosten ist, werden stets zugewiesen. Diese einfachen Zuteilungsentscheidungen werden

durch Nutzen-Funktionen komplexer, da das System nun zwischen mehrerer Alternativen entscheiden muß. Nachstehende Nutzen-Funktion besitzt besonders schöne theoretische Eigenschaften für einen Pool von n Maschinen

$$n^{(Auslastung/Maximalauslastung)} \tag{3.5.1}$$

Mit dieser Nutzen-Funktion liegt die Maximalauslastung von Ressourcen höchstens $O(logn)$ über der Maximalauslastung eines optimalen Systems [AAB98b]. Angesichts der Tatsache, daß andere Zuteilungsstrategien überhaupt keine oberen Schranken besitzen, ist selbst diese schwache theoretische Obergrenze eine hervorzuhebende Eigenschaft, die in der Praxis meist deutlich unterschritten wird.

Der Markt Der Java Markt besteht aus drei Komponenten, dem *Ressourcen-Manager*, dem *Task-Manager* und dem *Markt-Manager*, der zwischen den beiden erstgenannten Komponenten vermittelt. Alle drei Manager laufen auf dem Server des Java Market Systems.

Ressourcen-Manager – Die Aufgabe des Ressourcen-Managers ist die Erstellung und Verwaltung von Maschinenprofilen, die die Eigenschaften (IP Adresse, CPU Geschwindigkeit, Bandbreite, Latenz) der teilnehmenden Produzenten enthalten. Die hierfür notwendigen Daten erhält der Ressource-Manager von dem *Launch Applet*, das auf dem Produzenten ausgeführt wird, wenn er den Market-Server kontaktiert.

Task-Manager – Der Task-Manager erledigt alle Maßnahmen, die zur Registrierung, Modifikation und Verwaltung der Konsumenten Tasks notwendig sind. Die dafür nötigen Informationen wie sämtliche Java und Eingabe Dateien, liefert das *Request Applet*, das ausgeführt wird, wenn der Konsument einen Task beim Markt-Server registriert. Der Task-Manager lädt alle Dateien über das WWW, modifiziert und übersetzt den Sourcecode und übergibt das Applet dann an den Markt-Manager.

Markt-Manager – Die Aufgabe des Markt-Managers ist die Zugangskontrolle zum Markt und die Ressourcenzuteilung an Konsumenten. Zwei Probleme treten bei diesen Aufgaben auf, nämlich die Unsicherheit über die zukünftige Verfügbarkeit eines Produzenten und die Notwendigkeit, den Gewinn des Markts zu optimieren. Um beide Aufgaben zu erfüllen wendet der Markt-Manager hierzu eine Kombination der beiden bereits besprochenen Verfahren an, des *Cost-Benefit*- und des *Winner-Picking*-Verfahrens.
Zunächst erfolgt eine Zugangsprüfung von Tasks anhand der *Cost-Benefit*-Strategie, deren zentrales Konzept die Grenzkosten sind. Grenzkosten sind eine Kosten-Funktionen, die exponentiell mit dem zunehmenden Gebrauch einer Ressource steigen. Ein Task wird akzeptiert oder abgelehnt, je nachdem, ob der durch seine Ausführung erzielbare Gewinn die Grenzkosten, diesen Task auszuführen, übersteigt oder nicht. Ist ein Task erst einmal zugelassen worden, können ihm Ressourcen zugewiesen werden. Dies geschieht unter Zuhilfenahme des *Winner-Picking*-Verfahrens, deren Zuteilungskosten nach oben begrenzt sind.

Eigenschaften Die Entwickler von Java Market haben eine Evaluierung ihres Systems anhand von Eigenschaften wie Dienstgarantien, Fehlertoleranz, Schutz vor Mißbrauch, Ressourcentransparenz, optimale Ressourcenzuteilung, Skalierbarkeit, Verteilungstransparenz und Leistungsbewertung vorgenommen. Im folgenden findet sich eine Zusammenfassung dieser Evaluierung.

Dienstgarantien – Um Voraussagen und Garantien über Rechenleistung in der Zukunft abgeben zu können, extrapoliert Java Market die zukünftige Verfügbarkeit eines teilnehmenden Anbieters aus der Vergangenen. Um die Korrelation zwischen vergangener und zukünftiger Verfügbarkeit zu verbessern und die Verschwendung von Ressourcen zu vermeiden, wird die *Winner-Picking*-Strategie eingesetzt. Dieses Verfahren verteilt Tasks an Produzenten mit einer Wahrscheinlichkeit, die exponentiell zu der Zeit ansteigt, die der Produzent ununterbrochen am Markt teilnimmt. Dadurch werden langfristige Tasks bevorzugt an zuverlässige und kurzfristige Tasks an eher unzuverlässige Produzenten vergeben. Darüberhinaus bevorzugt Java Market Produzenten, die eine gewisse Verfügbarkeit garantieren, mit großzügigerer Entlohnung.

Fehlertoleranz – Da Java Market fortlaufend in Kontakt mit den Marktteilnehmern steht, können Produzenten, die nicht mehr teilnehmen oder technische Schwierigkeiten haben, sofort aus dem System entfernt werden.

Schutz vor Mißbrauch – Das Sicherheitsmodell von Java Market basiert auf dem Java Sand-box Modell und erweitert dies noch um notwendige Punkte. Beispielsweise schützt die JVM den Host nicht davor, daß ein Applet sämtliche Systemressourcen verschlingt. Um zu verhindern, daß ein Applet die Kontrolle der CPU eines Providers nicht mehr abgibt, modifiziert Java Market den Applet-Code um Routinen, die den Markt selbst sowie seine Produzenten schützen.

Ressourcentransparenz – Heterogene Ressourcen wie Architektur, Betriebssystem und Browser werden vollständig durch die JVM verdeckt.

Optimale Ressourcenzuteilung – Das in Java Market verwendetet *Cost-Benefit* Schedulingverfahren bezahlt Produzenten für die Bereitstellung und belastet Konsumenten abhängig von der zur Verfügung gestellten oder verbrauchten Rechenleistung. Im Gegensatz zu traditionellen Zuteilungsverfahren, die die Systemauslastung zu optimieren versuchen, versucht die *Cost-Benefit*-Strategie den Systemprofit – der gesamte durch Ressourcen erzielte Gewinn abzüglich der für diese Ressourcen aufgewendete Preis – zu maximieren.

Nachteile In einigen der im letzten Abschnitt geforderten Eigenschaften einer Meta-Computing Umgebung weist Java Market jedoch auch deutliche Schwächen auf. So ist das System nicht skalierbar, nicht verteilungstransparent und kann nur eine eingeschränkte Leistungsbewertung vornehmen.

Skalierbarkeit – Java Market ist durch einen zentralen Server implementiert, der einen Schwachpunkt des Systems darstellt und nicht skaliert.

Verteilungstransparenz – Java Market tritt laut seinen Entwicklern nicht in Konkurrenz zu transparenten Meta-Computing Systemen. Java Market Teilnehmer müssen explizit ein verteiltes System ansprechen. Insofern ist keine Transparenz bezüglich Heterogenität und Verteiltheit der Ressourcen erkennbar.

Leistungsbewertung – Aufgrund der Sicherheitsbeschränkungen der JVM kann Java Market nur zwei Eigenschaften des Produzenten empirisch bestimmen, die CPU Geschwindigkeit und die Latenz des Netzanschlusses.

3.5.5 Spawn

Spawn [Wal92] erlaubt eine weitaus bessere Abwägung zwischen Ressourcen verschiedener Qualität als das im älteren Enterprise System der Fall war. Durch die Berücksichigung des Preises kann eine Anwendung die für sie günstigsten – teure, hochwertige oder preiswerte, minderwertige – Ressourcen zugeteilt bekommen.

Spawn ist ein elektronischer Markt und Marktplatz, der aus interagierenden Käufern und Verkäufern besteht. Käufer sind Marktteilnehmer, die zur Lösung ihres Problems Rechenressourcen kaufen wollen, während Verkäufer versuchen brachliegende Rechenressourcen ihres Rechners am Spawn Markt abzusetzen. Hierfür startet der Verkäufer einen Auktionsprozeß auf seiner Maschine, der den Verkauf seiner ungenutzten Ressourcen übernimmt, und der Käufer leitet einen Bietprozess ein, der Gebote für Rechenzeit auf benachbarten Auktionen abgibt. In der Spawn-Ökonomie sind daher Rechte an Ressourcen über Budgets und das Gleichgewicht zwischen Angebot und Nachfrage über Preise geregelt.

Architektur Wie bereits erwähnt wird Kauf und Verkauf von Rechenressourcen über eine Reihe von Systemprozessen realisiert. Diese sind Auktions-, Biet-, Ressourcen- und Anwendungs-Manager.

Auktions-Manager – Der Auktions-Manager verwaltet den Ressourcenverkauf untätiger Maschinen und nimmt dazu fortlaufend Kaufangebote von Nachfragern nach der nächsten frei werdenden Zeitscheibe entgegen. Ressourcen werden exklusiv zur Verfügung gestellt, das heißt, kein (rechenbereiter) Prozeß eines anderen Bieters oder des Eigentümers konkurriert mit dem Käuferprozeß um die Rechenressourcen. Die Auktionsstrategie ist durch den Ressourceneigentümer durch Angabe von minimalen und maximalen Zeitscheibenlängen parametrisierbar sowie einer Auktions-Funktion, die die Vergabestrategie bezüglich der Scheibenlänge, des Preises oder anderer Marktwerte beschreibt. Dadurch kann ein Anbieter zum Beispiel abhängig von der Marktsituation Nachlässe oder Aufschläge für längere Zeitscheiben verlangen. In der Praxis müssen die Zeitscheiben lang genug sein, um den Auktions- und Startaufwand zu rechtfertigen, aber kurz genug, um dem Ressourceneigentümer ausreichend schnell seine Maschine wieder zur Verfügung zu stellen.

Biet-Manager – Gebote bestehen aus der gewünschten Länge der Zeitscheibe, dem Preis und einer kurzen Aufgabenbeschreibung. In Spawn werden zur Ressourcenzuteilung Zweitpreis-Submissionsauktionen verwendet, eine Strategie, die Bieter dazu veranlaßt, durch ihr Gebot ihren wahren Nutzen für die nachgefragte

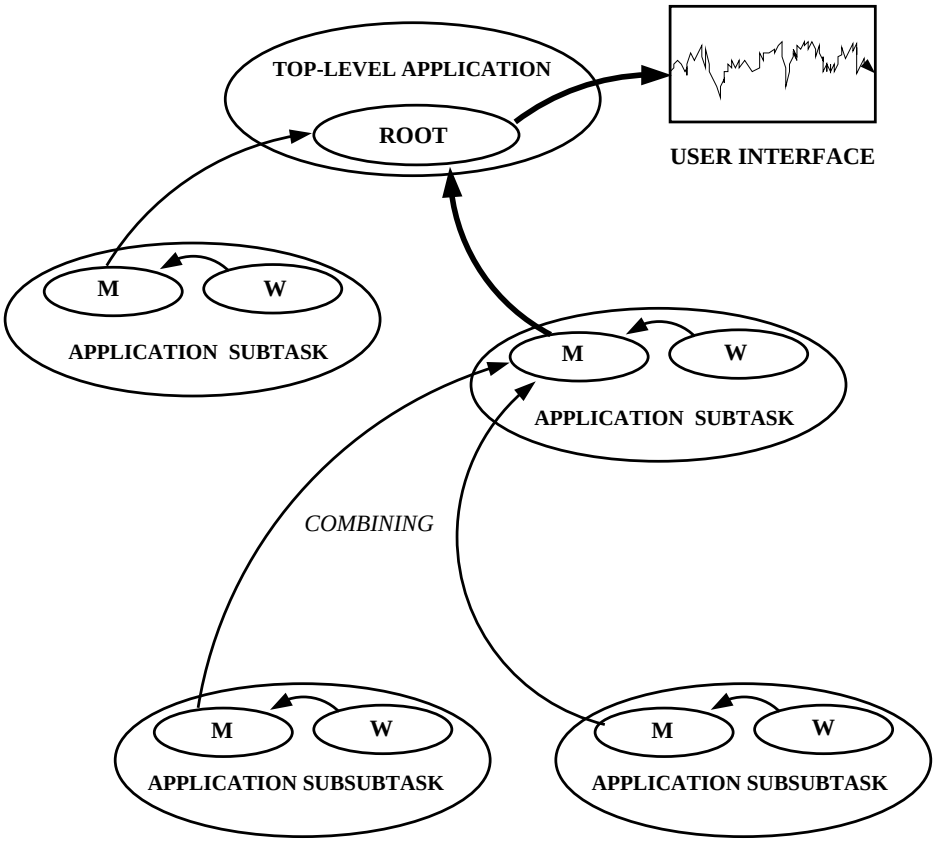

Abbildung 3.7 Spawn Anwendungs-Manager [Wal92]

Ressource zu offenbaren und das seine Effektivität in wirklichen Märkten nachgewiesen hat. Empirische Studien und theoretische Analysen haben gezeigt, daß Zweitpreis-Submissionsauktionen zu ähnlichen Preisen führen wie offene englische oder holländische Erstpreis-Auktionen, allerdings mit geringerem Kommunikations- und damit Rechenaufwand [Reg98]. Falls kein zweites Gebot vorliegt, ist der Marktpreis Null, das heißt ohne Nachfragewettbewerb sind Ressourcen frei.

Ressourcen-Manager – Mit jeder Auktion ist ein Ressourcen-Manager assoziiert, der die Ausführung und Abrechnung der Anwendung übernimmt, die die aktuelle Zeitscheibe ersteigert hat. Verbraucht eine Anwendung mehr Ressourcen als sie erworben hat, wird sie vom Ressourcen-Manager beendet. Da Spawn keine Sicherungspunkterzeugung und keine Prozeßmigration unterstützt, soll eine vorzeitige Prozeßterminierung nach Möglichkeit vermieden werden. Dazu besitzt jede Anwendung einmal die Option, die Präemption durch den Ressourcen-Manager zu verschieben (*right of first refusal*), falls die Anwendung in der Lage ist, den Marktpreis für die nächste Zeitscheibe zu bezahlen, um die Berechnung zu Ende zu führen. Es ist daher Verantwortung der Anwendungen sicherzustellen, daß genügend Mittel zur Berechnung der Aufgaben zur Verfügung stehen.
Der Ressourcen-Manager abstrahiert die Anwendungen von Marktmechanismen wie

die Lokalisierung, Belegung und Abrechnung von Ressourcen auf unterster Ebene. Dennoch erlaubt er den Anwendungen in einem gewissen Rahmen die Kontrolle über die Ressourcen-Allokation durch die Bereitstellung eines Budgets für Sub-Tasks einer Anwendung. Der Ressource-Manager verbirgt die Details der Ressourcenzuteilung vor der Anwendung.

Anwendungs-Manager – Spawn Anwendungen bestehen aus Manager- und Arbeitermodulen. Das Arbeitermodul enthält den Steuerfluß der Anwendung, für die Ressourcen bereitgestellt werden sollten, und besitzt einen zugehörigen Anwendungs-Manager, der für die Ressourcenzuteilung des Arbeiters und seiner Sub-Tasks verantwortlich ist. Dazu steht der Anwendungs-Manager mit dem Ressourcen-Manager in Kontakt und sorgt für eine Abspaltung von Arbeitermodulen und Anwendungs-Managern zur Bearbeitung von Sub-Tasks. Die Anwendungs-Manager stellt somit die Schnittstelle für Anwendungen zum Spawn System dar, wobei der oberste Anwendungs-Manager die Schnittstelle zu einer verteilten Anwendung darstellt.
Im einfachsten Fall fordert der Haupt-Anwendungsmanager die Ausführung einer *Blackbox* Anwendung an und stellt hierfür ein Budget zur Verfügung. Sobald der lokale Anwendungsmanager eine Auktion um Rechenzeit gewonnen hat, wird der Task gestartet. Sein Anwendungsmanager fängt die Ausgabe der Anwendung auf und leitet sie an den Haupt-Anwendungsmanager zurück. Falls die Anwendung aus einer Baumstruktur von Sub-Tasks besteht, leiten die Sub-Task die Ergebnisse ihrer nebenläufigen Teilberechnungen an ihre Anwendungsmanager, die die einkommenden Ergebnisse zusammenfassen und an die nächsthöheren Manager weiterleiten. Anwendungsmanager zerlegbarer Anwendungen können weitere Kinder zur Bearbeitung von Sub-Tasks erzeugen wie in Abbildung 3.7 illustriert.

Budgets Das Hauptziel von Spawn ist die Anwendung gesponserten Rechnens (*sponsored computing*) im Rahmen elektronischer Ökosysteme zur fairen Ressourcenzuteilung an nebenläufige, baumbasierte Programme. Die verschiedenen Manager dienen dabei als Sponsoren ihrer Arbeitermodule und kontrollieren die Verteilung des Budgets auf die von ihnen erzeugten Sub-Tasks. Durch die Beschränkung auf die Verteilungsschlüssel relativer Budgets wird die Schnittstelle zur Spawn Infrastruktur von Details wie absoluten Budgetgrössen oder komplexe Kommunikationsprotokolle zur Anforderung und Auswertung von Budgets befreit. Einfache Budgetierungs-Strategien brauchen daher ihre Geldmittel nur gleichmässig auf ihre Sub-Tasks zu verteilen während komplexere Strategien Lastwerte, Programm-Fortschritt oder Effektivität von Sub-Tasks dynamisch mit einbeziehen könnten. Spawn bietet Anwendungen damit ein relativ abstraktes Rahmenwerk zur Ressourcenzuteilung auf höherer Ebene an.

Die Spawn Sponsorenschaft kann als baumartiges Röhrensystem angesehen werden, durch das (abstrakte) Währungen von der Wurzel zu den Managern fließen. An jeder Verzweigung der Röhren sitzt, wie in Abbildung 3.8 veranschaulicht, ein Ressource-Manager, der über eine Einleitungsrohr, ein Währungsreservoir sowie einen im Durchmesser beschränkbaren Auslaß verfügt. Durch die Anpassung des Durchmessers und der Größe des Auslasses und des Reservoirs hat der Manager Kontrolle über sein eigenes Budget und die Finanzierung seiner Sub-Tasks. Auf oberster Ebene wird die Vergabe von Mitteln für

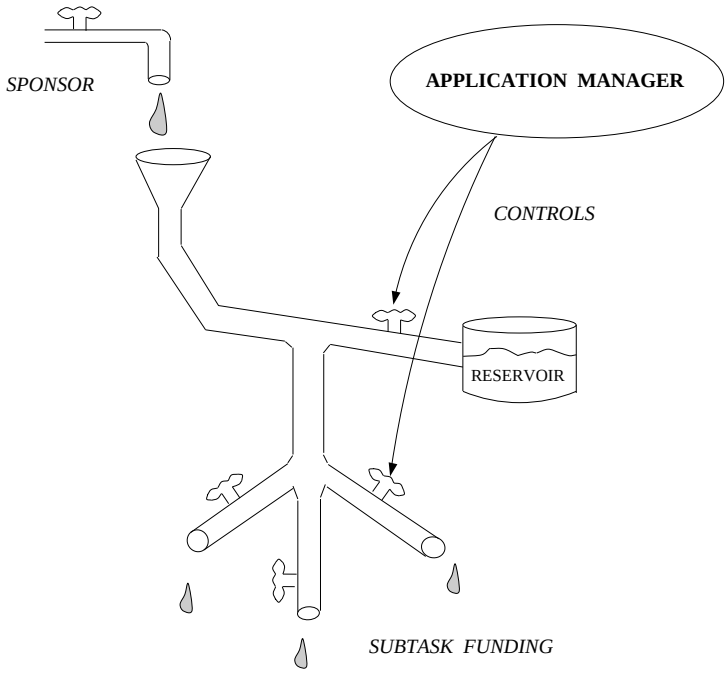

Abbildung 3.8 Spawn Währungsfluß [Wal92]

Ressourcen von menschlichen Administratoren bestimmt. Manche Benutzer werden in der Lage sein, ihr Ressourcen-Budget durch Verkauf freier Rechenzeit auf ihrer Workstation zu erzielen, während der hohe Ressourcenbedarf anderer Benutzer von der Administration durch ein hohes Budget manifestiert werden muß, das die relative Bedeutung dieser Benutzer bzw. ihren besonderen Ressourcenhunger unterstreicht.

Nachteile Spawn ist für Netze heterogener Arbeitsplatz-Rechner implementiert und profitiert dadurch von einer sicheren Ausführungsumgebung, deren Fehlen beispielsweise das *Enterprise* System stark einschränkte. Diese Tatsache bedingt allerdings auch einige Unzulänglichkeiten des Systems. So ist *Spawn* in erster Linie ein Job-Scheduling System für Netze von Arbeitsplatz-Rechnern und bewertet den individuellen Nutzen oder Betrag eines Jobs zur Ökonomie des Gesamtsystems nicht. Jobs werden über Sponsoren finanziert, und zwar solange, bis sie auch tatsächlich ausgeführt werden. Würde *Spawn* zur Zuteilung den wahren ökonomischen Betrag eines Jobs berücksichtigen, könnte es zur Aushungerung einzelner Jobs kommen, falls stets wichtigere Jobs rechenbereit sind, eine Situation, die für einen Scheduler nicht akzeptabel ist.

Außerdem ist die Prozeßerzeugung für eine im Benutzermodus ablaufende Infrastruktur zu teuer um andere als sehr grobgranulare nebenläufige Anwendungen zu unterstützen. Auch können Prozesse nicht zwischen Maschinen migriert oder Fehler- und Abbruchbedingungen abgefangen werden, sondern die Anwendungen müssen vielmehr selbst für eine ausreichende Finanzierung und zeitgerechte Terminierung sorgen. Außerdem bietet Spawn keinerlei Sicherheitsmechanismen, die Anwendungen und Manager vor böswilliger Manipulation von Budgets schützt.

System	Transaktionskosten	Skalierbarkeit	Fehlertoleranz	Lokalität	Legacy
Enterprise	gering	nein	nein	nein	nein
JavaMarket	mittel	nein	nein	nein	ja
Popcorn	mittel	nein	nein	nein	nein
ReGTime	hoch	nein	nein	nein	ja
Spawn	gering	ja	nein	nein	nein

Tabelle 3.1 Klassifikation elektronischer Märkte

3.6 Klassifikation und Bewertung der Systeme

Die Vorstellung der verschiedenen elektronischen Märkte und Marktplätze in den vorausgegangenen Abschnitten hat gezeigt, daß sich die Systeme wesentlich in der Höhe ihrer Transaktionskosten sowie der Unterstützung von Standardsoftware, sogenannten *Legacy*-Systemen unterscheiden. Die im Anschluß vorgenommene Betrachtung elektronischer Märkte erfolgt daher sowohl unter allgemeinen Aspekten, aber insbesondere auch unter den Gesichtspunkten der Transaktionskosten und der Software Unterstützung. Die Ergebnisse der Betrachtungen der vorangegangenen Abschnitte finden sich zusammengefaßt in Tabelle 3.1 und bilden Grundlage des folgenden näheren Vergleichs der Systeme.

Vom Gesichtspunkt der Transaktionskosten betrachtet, zeichnet sich eine sichtbare Trennung zwischen "echten" elektronischen Märkten wie Enterprise und Spawn und hybriden Web-Infrastrukturen wie JavaMarket und Popcorn ab. Elektronische Märkte zeichnen sich als spezialisiertes und effizienteres System durch niedrigere Transaktionskosten beim Güterhandel als Web-Computing Systeme aus. Dies ist auch Folge der Tatsache, daß Enterprise und Spawn zwar dezentrale, aber geschlossene Systeme mit statische Ressourcen sind. Dagegen sind Web-Computing Infrastrukturen offene Systeme, die auch einen sehr dynamischen Pool von Ressourcen verwalten können und diesen Vorteil durch höhere Transaktionskosten erkaufen. ReGTime nimmt eine Sonderstellung ein, da es nur einen Marktplatz für Rechenressourcen zur Verfügung stellt, ein Großteil der Transaktionen einen Marktes müssen von den Teilnehmern ohne Unterstützung des ReGTime Systems selbst vorgenommen werden. Hierdurch sind die Transaktionskosten von ReGTime höher als bei den beiden vorhergehenden Markttypen.

Im Hinblick auf *Legacy* Software läßt sich allgemein ein Mangel an Unterstützung für Standardsoftware erkennen. Elektronische Märkte bieten als spezialisierte geschlossene Systeme keine Rückwärtskompatibilität für traditionelle Anwendungen, aber auch hybride Web-Computing Infrastrukturen tun sich mit *Legacy*-Unterstützung schwer und bieten mit Ausnahme von JavaMarket keine Unterstützung hierfür an. ReGTime biete Rechenressourcen auf Grundlage von Standard-Benutzerkennungen und unterstützt daher sämtliche Anwendungen, die auf einer UNIX Workstation installierbar und lauffähig sind.

Mit Ausnahme von Spawn ist keines der Systeme hoch skalierbar und allen Markttypen gemeinsam ist das Fehlen von Mechanismen zur Fehlertoleranz und die Berücksichtigung der Lokalität zur Reduzierung der Verwaltungs- und damit der Transaktionskosten, wie zum Beispiel anonyme und automatische Markteilnahme.

Kapitel 4

Modellierung eines elektronischen Ressourcenmarkts

4.1 Einführung

Werden die im Zusammenhang mit der Nutzung von NOWs eingesetzten Programmierumgebungen und gewonnenen Erfahrungen auf globale Netze übertragen, wird das Weitverkehrsnetz (*Wide Area Network*, WAN) nur unzureichend ausgenutzt. Bei einem solchen Vorgehen wird nur der infrastrukturelle Aspekt eines WANs verwendet, der "soziale" Aspekt, wie zum Beispiel die Vielzahl und Heterogenität der teilnehmenden Institutionen und Benutzer, die sich in unterschiedlicher Administration und unterschiedlichen Nutzungsgewohnheiten offenbaren, bleibt unberücksichtigt. In einem lokalen Netzwerk (*Local Area Network*, LAN) sind die beteiligten Nutzer homogen, zahlenmäßig gleichstark und betreiben einen Ausgleich oder Austausch von freien Ressourcen. Darüberhinaus unterliegen sie der gleichen oder zumindestens kooperierender administrativer Kontrolle. Diese Struktur ähnelt einer zentralen Planwirtschaft: reichen Ressourcen zur Erwirtschaftung der von einer zentralen Instanz vorgegebenen Aufgaben nicht aus, können zusätzliche Ressourcen von anderen Betrieben und Institutionen entliehen oder "beschafft" werden, ohne daß diese Ressourcen bezahlt und bilanziert werden müssen.

Die Benutzer eines Weitverkehrsnetzes unterscheiden sich dagegen sehr stark von denen eines LANs, sie sind äußerst heterogen, zahlen- und ortsmäßig sehr stark gestreut, und unterliegen verschiedener administrativer Kontrolle. Verfahren und Programmiermodelle, die Eigenschaften eines LANs voraussetzen und auf diesen aufbauen, sind aus den genannten Gründen zur Nutzung global verteilter Ressourcen völlig ungeeignet. Es werden vielmehr Methoden und Modelle benötigt, die die Strukturen eines WANs wie Heterogenität, Asymmetrie und lokale Administration berücksichtigen und widerspiegeln und sich damit an einer freien Marktwirtschaft anlehnen. Es ergeben sich demnach eine Reihe von Anforderungen an Modelle und Verfahren zur Sammlung und Nutzung immaterieller Güter wie Rechenleistung und Speicherkapazität in Weitverkehrsnetzen.

4.2 Modellvoraussetzungen

Asymmetrische Modellierung – Um das volle Potential des World Wide Web, das heißt seiner Millionen von Benutzern, zu erreichen – im Gegensatz zur bloßen Verwendung seiner Mittel, wie HTTP, Browser und Java – muß eine asymmetrische Sichtweise eines Ressourcenmarktes angenommen werden. Wegen des geringen Effizienz dieser Mittel sind zur

praktischen Durchsetzung eines Ressourcenmarktes eine bis mehrere Größenordnungen mehr Anbieter als Nachfrager von Ressourcen notwendig. Eine Ausrichtung auf Ingenieure und Informatiker als Anbieter und Nachfrager von Rechenressourcen allein ist daher nicht ausreichend. Vielmehr ist die Beteiligung von Laien – der Durchschnittsbenutzer des WWW – als Anbieter überschüssiger Ressourcen Voraussetzung für einen Erfolg.

Transparente Teilnahme – Durch die Einbindung einer großen Anzahl von Laien in das System entstehen jedoch eine Reihe weiterer Probleme bzw. Voraussetzungen. Aus mehreren Gründen kann nicht jeder kleine Anbieter von Ressourcen durch eine zentrale Instanz verwaltet werden. Zum einen ist die Verwaltung und Buchung hunderttausender von Mitgliedern nicht effizient und in vielen Fällen in hinreichender Zeit nicht möglich. Zum anderen kann von Laien, die nicht mit den Begriffen und Funktionsweisen von Ressourcen und Märkten vertraut sind, nicht erwartet werden, daß sie sich mit technischen Details wie Angebot, Nachfrage oder Buchung von Ressourcen beschäftigen. Solche Details müssen vor diesem Anbieterkreis so weit wie möglich verborgen werden und der An- und Verkauf von Ressourcen transparent – das heißt für die Anbieter automatisch, ohne Notwendigkeit der Interaktion – gestaltet werden.

Marktbasierte, hierarchische Verwaltung – Lösungen, die die vorausgehenden Forderungen erfüllen, sind marktbasierte Ressourcenzuteilungs-Strategien, die auf dezentraler Entscheidungsfindung beruhen. Dezentrale Mechanismen ermöglichen die gleichzeitige Erfüllung sehr vieler Nebenbedingungen und Präferenzen von Marktteilnehmern und fördern die Entwicklung lokaler Handelspolitiken, die örtliche Besonderheiten hinsichtlich Transaktionskosten, Sicherheitsanforderungen und Teilnahmeanreizen von Teilmärkten widerspiegeln. Normalerweise sind die Anforderungen an Transaktionskosten, Sicherheit und funktionierende Anreize unter einander fremdem Teilnehmern in einem globalen Markt ungleich höher als in einem lokalen Markt mit einander befreundeten oder zumindest bekannten Partnern. Mit Hilfe eines hierarchischen Netzes von Zwischenhändlern, die jeweils einen Teilmarkt des Gesamtsystems verwalten, sowie unter Ausnutzung der Lokalität dieser Teilmärkten ist es jedoch möglich, Transaktionskosten, Sicherheitsanforderungen und Anreize an die Bedingungen des Teilmarkts anzupassen und so weit wie möglich zu reduzieren.

Lokale Teilmärkte und Politiken – Die Marktteilnahme wird durch die regionalen Zwischenhändler verwaltet, die durch lokale Politiken für die Sammlung und Nutzung von Ressourcen dafür sorgen (können), daß die Voraussetzungen für eine einfache Teilnahme wie Anonymität, transparente Teilnahme, geringe Transaktionskosten, funktionierende, lokalisierte Teilnahmeanreize und -bedingungen usw. erfüllt sind. Als Gegenleistung für eine Marktteilnahme werden Mikrowährungen oder Austauschgüter ausgeschüttet. Es bleibt den dezentralen Zwischenhändlern überlassen, welche Strategien sie zur Sammlung von Ressourcen einsetzen – ob interaktiv Mitglieder-basiert nach dem Beispiel von regelmäßigen Abbuchungen oder automatisch und anonym, ähnlich der bis in die achtziger Jahre praktizierten anonymen Blutspende. Je nach lokaler Umgebung bietet die eine oder die andere Strategie Vorteile. In einer Gemeinschaft mit einem festen, überschaubarem Teilnehmerkreis wie einem unternehmensweiten Intranet besitzt eine automatische Sammlung von Ressourcen Vorteile, während in einem anonymen und dynamischen Benutzerverbund die erhöhten Sicherheits- und Buchführungsanforderungen durch eine Mitglieder-basierte

Sammlung und Verteilung berücksichtigt werden sollten.

Funktionierende, lokalisierte Anreize – Bei der automatischen Marktteilnahme muß der Anonymität und Dynamik der Teilnehmer des WWW auch dadurch Rechnung getragen werden, daß die Ausschüttung von Gegenleistungen vereinfacht wird. Die Bezahlung gelegentlicher und kurzfristiger Anbieter von Ressourcen mit Geld beinhaltet zuviel Aufwand. Solche Anbieter sollten stattdessen mit Mikro-Währungen entlohnt werden und diese nur akkumuliert in Geld umgetauscht werden. Dies wird in der Regel bei Zwischenhändlern, die größere Mengen von Ressourcen anhäufen, der Fall sein.

Zusammenfassung Aus den genannten Voraussetzungen kristallisieren sich drei Säulen heraus, die einen Markt für überschüssige Rechenressourcen tragen können, Multiplikation, Lokalität und Rekursivität.

Ausnutzung multiplikativer Effekte – Es werden deutlich mehr Anbieter als Verbraucher benötigt, um höhere akkumulierte Rechenleistung zu erzielen. Als Zielgruppen müssen daher neben technisch fachkundigem Publikum (als Abnehmer der Rechenleistung) auch Durchschnittsbenutzer des WWW als Anbieter in Betracht genommen werden. Die enorme Größe dieser Zielgruppe und ihre effiziente Verwaltung kann nur durch eine dezentrale Organisation über eine Reihe von Zwischenhändlern, die jeweils einen Teilmarkt verwalten, als Multiplikatoren zur Sammlung und Abrechnung von Ressourcen kleiner Mengen bewältigt werden. Insgesamt ergibt sich eine asymmetrische Modellierung der Teilnehmer.

Ausnutzung der Lokalität – Die Verwaltung der Zwischenhändler bzw. Multiplikatoren mit ihren Teilmärkten sowie einer sehr großen Zahl von Teilnehmern ist von einer zentralen Steuerkomponente nicht möglich, selbst wenn sie allwissend ist. Stattdessen ist eine dezentrale Entscheidungsfindung auf Basis marktbasierter Verfahren, die zum größten Teil auf lokalen Informationen beruhen, besser geeignet, Ressourcen effizient zuzuteilen. Weiterhin besitzen verschiedene Teilmärkte auch sehr unterschiedliche Anforderungen an Transaktionskosten, Sicherheitsanforderungen sowie Teilnahmebedingungen und -anreize. Es muß den Multiplikatoren daher überlassen bleiben, auf ihre Teilmärkte abgestimmte Politiken zur Sammlung, zum Handel und zur Abrechnung von Ressourcen zu entwickeln und damit lokale Besonderheiten zu berücksichtigen und gezielt auszunutzen. Ziel lokaler Politiken ist stets die weitestmögliche Reduzierung der Kosten zur Verwaltung und Aufrechterhaltung des Ressourcenhandels.

Anwendung von Rekursivität – Zum Aufbau und zur Verwaltung hierarchisch organisierter Zwischenhändler und Teilmärkte bietet sich das Prinzip der Rekursivität an. Zu große, überlastete oder inhomogene Märkte[1] werden rekursiv in kleinere und homogenere Einheiten aufgeteilt, hierarchisch gekoppelt und dezentral von den untergeordneten Server-Diensten verwaltet. Diese Rekursivität terminiert, falls eine weitere Aufteilung nicht mehr möglich ist. Dies ist dann gleichbedeutend mit der Tatsache, daß die entstandenen Teilmärkte homogen geworden sind und ihre Server-Dienste sich in einem Gleichgewichtszustand befinden.

[1] und zugehörige Server-Dienste

4.2.1 Multiplikative Effekte

Strukturelle Asymmetrie – Traditionelle Modelle und Methoden zur gemeinsamen Nutzung von Ressourcen in Nahverkehrsnetzen (LAN) basieren auf einem symmetrischen Modell (*peer-to-peer*) der beteiligten Parteien und Ressourcen. Die Teilnehmer und Ressourcen in einem LAN sind meist homogen und unterliegen der gleichen oder zumindest kooperierender administrativer Kontrolle. Bei der Übertragung solcher Methoden auf Weitverkehrsnetze (WAN) wird nur der infrastrukturelle Aspekt eines WANs verwendet, der "soziale" Aspekt, wie zum Beispiel die Vielzahl und Heterogenität der teilnehmenden Institutionen und Benutzer, die sich in unterschiedlicher Administration und unterschiedlichen Nutzungsgewohnheiten offenbaren, bleibt unberücksichtigt. Das WAN dient lediglich zur Überbrückung größerer Entfernungen als sie in einem LAN möglich sind. Um dieser strukturellen Asymmetrie Rechnung zu tragen, ist ein asymmetrischer Ansatz bei der Sammlung und Nutzung frei verfügbarer Ressourcen notwendig.
Die Teilnehmer sind jedoch nicht nur strukturell verschieden sondern erfüllen zudem noch sehr unterschiedliche Rollen, wie wir im nächsten Abschnitt sehen werden.

Numerische Asymmetrie – Die strukturelle Asymmetrie von WANs erfordert zur Überwindung von Heterogenität und Verteilung spezielle *Middleware*-Protokolle und -Sprachen, die sich jedoch zum heutigen Zeitpunkt allesamt durch relativ geringe Effizienz auszeichnen. Diese Ineffizienz kann in einem skalierenden System durch eine sehr hohe Zahl von Teilnehmern ausgeglichen werden. Anwendungen, deren Ressourcenbedarf die Ressourcen aktueller Rechensysteme um ein Vielfaches übertreffen, sind jedoch rar. In einem Ressourcenmarkt werden sich aus diesem Grund nur sehr wenige Nachfrager, jedoch abhängig vom gebotenen Preis unter Umständen sehr viele Anbieter von Ressourcen finden. Daher ist zur Modellierung eines solchen Marktes ein asymmetrischer Ansatz naheliegender als ein symmetrischer. Die größere Bedeutung der Anbieter von Ressourcen in einem solchen Markt muß sich daher in einem asymmetrischen Modell widerspiegeln. Diese Tatsache muß weiterhin auch zur Ausrichtung auf den Anbieter führen. Sämtliche mit Anbietern in Verbindung stehende Modellentitäten und Transaktionen in einem Ressourcenmarkt sollten aus deren Sichtweise modelliert werden und auf deren Bedürfnisse optimiert werden.

Beide Asymmetrien, sowohl die strukturelle als auch die numerische, führen zu einem Gefälle, das zur Erhöhung der Leistung eines Ressourcensammlungs-System verwendet werden kann. Aufgrund der strukturellen und numerischen Heterogenität befinden sich zu jedem Zeitpunkt sehr verschieden leistungsfähige, aber auch verschieden ausgelastete Knoten (bzw. ihre Benutzer) und diese auch in sehr unterschiedlicher Anzahl in einem Weitverkehrsnetz. Man stellt fest, daß sehr wenige sehr leistungsstarke Rechensysteme mit extrem leistungshungrigen Anwendern auf der einen Seite (fortan als Nachfrager bezeichnet) und extrem viele durchschnittliche, eventuell nur temporär angebundene, Rechner mit sehr moderaten Anwendern auf der anderen Seite (als *Web*-Terminals bezeichnet) existieren. Werden diese heterogenen Teilnehmer über einen Multiplikator zusammengeführt, sind die Voraussetzungen für ein verteiltes Hochleistungssystem gegeben.

Zielgruppen Die jeweiligen Zielgruppen sind von der implementierten Politik der verantwortlichen Zwischenhändler der Teilmärkte abhängig. Der kleinste mögliche Teilmarkt

ist ein anonymer WWW Benutzer mit einem *Web*-Terminal, der unter Umständen nur zeitweise angeschlossen ist, als Anbieter von Rechenzeit. Ein solcher minimaler Teilmarkt besteht aus einer monopolistischen CPU als Anbieter von Ressourcen und mehreren, um diese CPU konkurrierende Benutzer-Prozesse.

Definition 4.1 (*Web*-Terminal) *Unter einem* Web-*Terminal wird ein temporärer oder statischer Endknoten (Blatt) eines Weitverkehrsnetzes verstanden, der auf beliebiger Hardware und einem beliebigen Betriebssystem basiert. Einzige Anforderung an die Software-Infrastruktur ist ein Java-fähiger Browser.*

Web-Terminals sind zum Beispiel Marktteilnehmer, die geringe Mengen von Rechenleistung transparent und anonym zur Verfügung stellen und hierfür mit einer Mikrowährung entschädigt werden.

Definition 4.2 (Multiplikator) *Multiplikatoren sind Marktteilnehmer, die über eine große Menge an Nutzern oder Rechnern verfügen und die Ressourcen-sammelnde Wirkung des Locust-Clients vervielfachen, indem sie ihn an ihre Nutzer oder Rechner verteilen.*

Multiplikatoren wie *Content-Provider*, Firmen, Organisation und Rechenzentren nehmen als lokaler Markt mit einem Ressourcenüberschuß am Ressourcenmarkt teil.

Definition 4.3 (Nachfrager) *Nachfrager sind Marktteilnehmer, die die Locust-Infrastruktur bzw. deren Rechenleistung für verteilte und parallele Anwendungen nutzen.*

Nachfrager nach Ressourcen, wie beispielsweise eine Trickfilm-Firma, nehmen als lokaler Markt mit einem Ressourcenmangel am Ressourcenmarkt teil.
Das Modell unterscheidet demnach drei verschiedene Arten von Teilnehmern:

1. WWW Terminals – Der *Web*-Terminal (bzw. der jeweilige Benutzer) führt z.B. eine Recherche mittels einer Suchmaschine durch während auf seinem Computer als Gegenleistung der Locust Client ausgeführt wird.

2. Multiplikatoren – Ein Multiplikator ist beispielsweise ein Suchmaschinen-Betreiber, der den Locust-Client in seine Suchergebnisse einbettet, um sein Online-Angebot zu refinanzieren. Ein weiteres Beispiel für einen Multiplikator wäre ein Rechenzentrumsbetreiber, der den Locust-Client als Bildschirmschoner betreibt und dadurch Rechenzeit erwirtschaftet.

3. Nachfrager – Ein Nachfrager kann zum Beispiel eine Trickfilm-Firma sein, die die Locust Infrastruktur und die von ihr zur Verfügung gestellte Rechenzeit zum rechenintensiven *Rendering* eines animierten Filmes nutzt.

4.2.2 Rekursivität

Hierarchische Verwaltung Aufgrund der Größe eines Marktes mit Größenordnungen mehr Anbietern als Verbrauchern sowie einer großen Heterogenität der Teilnehmer stellt eine effiziente Verwaltung eine nicht zu unterschätzende Problematik dar. Eine zentrale Administration und Steuerung eines Ressourcenmarktes, die zumindest über einen

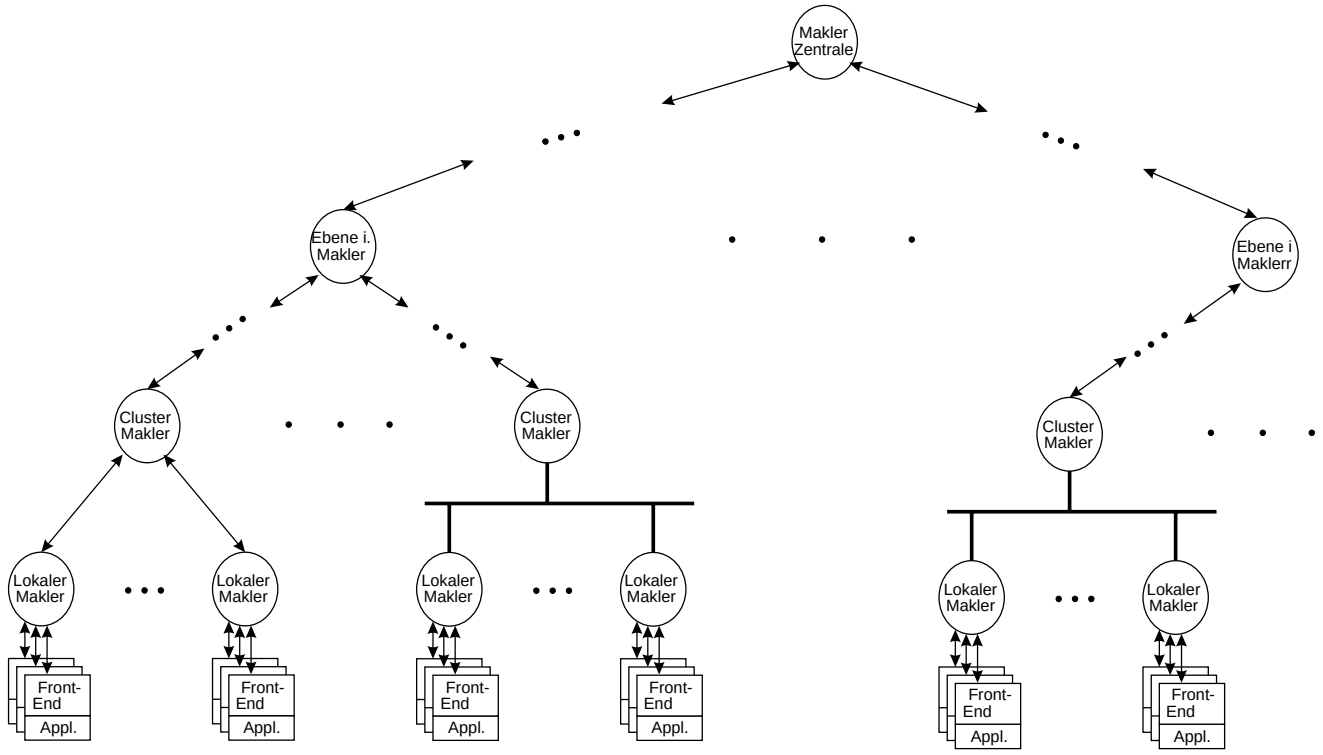

Abbildung 4.1 Hierarchie von Zwischenhändlern im Locust-Modell [Bac98]

Großteil aller Informationen verfügen müßte, ist daher allein aufgrund des anfallenden Kommunikations- und Berechnungsaufwands nicht sinnvoll. Stattdessen wird der – für die Verwaltung mittels eines einzelnen Maklers zu große – Markt bezüglich einer Eigenschaft in mindestens zwei Teilmärkte (Cluster) aufgeteilt, die jeweils von einen eigenen Makler administriert werden. Die bei der Teilung entstehenden Makler werden dem zentralen Makler unterstellt und können über diesen Informationen austauschen. Dieser Teilungsprozeß wird solange rekursiv fortgesetzt, bis die verbleibenden Teilmärkte/Cluster nicht mehr bezüglich einer Eigenschaft teilbar sind. Eine Unteilbarkeit bedeutet dann, daß der verbliebene lokale Teilmarkt bezüglich seiner Teilnehmer homogen geworden ist und von ihrem jeweiligen Makler zentral verwaltet werden kann (Vergleiche Abbildung 4.1).

Definition 4.4 (Teilmarkt) *Eine Teilmarkt ist ein elektronischer Markt, der bezüglich einer oder mehrerer bestimmten Eigenschaft homogen ist, daß heißt, er ist bzgl. dieser Eigenschaft(en) nicht weiter aufteilbar.*

Durch die Unterteilung des Marktes entsteht eine Struktur, die auch zur Sammlung und Verteilung von Ressourcen geeignet ist. In diesem Fall wirken die Verwalter auch als Händler und Zwischenhändler von Ressourcen.

4.2.3 Lokalität

Lokale Märkte Die Zwischenhändler übernehmen Management, Kontenverwaltung und Abrechnung für eine Vielzahl von kleinen anonymen Lieferanten. Durch die Einrichtung

einer Hierarchie von Zwischenhändlern kann das Management des Ressourcenmarktes lokalen Händlern und ihren Teilnehmern überlassen werden, die jeweils ihre eigenen Präferenzen für Sammlung und Nutzung von Ressourcen verfolgen.

In einem homogenen Teilmarkt, der unter der gleichen Administration oder sogar unter dem gleichen Besitzer steht, sind die Anforderungen an Sicherheit, Buchführung oder (finanzielle) Anreize ungleich niedriger als in einem Markt untereinander fremder Teilnehmer. Durch die Aufteilung des Marktes wird dabei eine sehr große Flexibilität erreicht. Jeder Zwischenhändler kann eine andere Strategie zur Sammlung oder Verteilung von Ressourcen anwenden und dabei Besonderheiten des lokalen Teilmarktes berücksichtigen, die stets mit einer Veränderung des Transaktionskosten einhergehen. Diese Präferenzen können durch Zielgruppen, Transaktionskosten, administrative Bereiche oder andere Parameter bestimmt sein. Ressourcen werden innerhalb eines Teilmarkts auch ohne strenge Anforderungen an die Sicherheit oder die Aufzeichnung der hierzu notwendigen Transaktionen ausgetauscht. Im einem solchen Teilmarkt ist in den meisten Fällen auch eine Entlohnung innerhalb einer Gruppe überflüssig. Die Führung von Konten für Ressourcen und Geld ist in einem Teilmarkt meist genausowenig notwendig wie die Abrechnung und Bezahlung von Ressourcen. All diese Vereinfachungen bei dem Austausch führen wie bereits weiter oben erwähnt zu einer Reduktion der Transaktionskosten, so daß in einem solchen Szenario ein automatischer Handel von Ressourcen mit anonymen Laufkunden effizienter ist als andere Modellierungen.

Marktteilnahme Darüberhinaus sollte die Marktteilnahme um so einfacher und unkomplizierter sein, je niedriger das Niveau eines Teilnehmers in Bezug auf Ressourcenqualität und -quantität sowie Vorwissen ist. Auch diese Forderung ist über eine dezentrale und hierarchische Verwaltung erreichbar. Die Marktteilnahme in unteren Ebenen wird durch lokale Händler verwaltet, die durch lokale Politiken für die Sammlung und Nutzung von Ressourcen dafür sorgen (können), daß die Voraussetzungen für eine einfache Teilnahme wie Anonymität, automatische Teilnahme, geringe Transaktionskosten, funktionierende, lokalisierte Anreize und Bedingungen usw. erfüllt sind. Das bedeutet, daß die Teilnahme am Ressourcenmarkt um so leichter sein sollte, je niedriger die Hierarchieebene des Teilmarktes innerhalb des gesamten Marktes ist. Da die meisten Teilnehmer in den unteren Hierarchie-Ebenen am Markt teilnehmen, führt diese Eigenschaft des Modells zu einer Vereinfachung der Verwaltung und Abwicklung des Ressourcenmarkts.

Definition 4.5 (Blutspende-Modell) *Im Folgenden wird unter einem Blutspende-Modell der automatische und transparente Handel von Ressourcen mit anonymen Laufkunden verstanden.*

In einem inhomogenen (Teil-) Markt mit unter Umständen nicht vertrauenswürdigen Partnern steigen die Transaktionskosten durch die erhöhten Anforderungen an Sicherheit, Teilnahmebedingungen und -anreizen an. Voraussetzung für einen Handel mit fremden Partnern ist ein angemessener Preis sowie genaue Aufzeichnung der Leistungen und Ressourcen, die Gegenstand des abgeschlossenen Vertrags sind. Außerdem sind erhöhte Sicherheitsanforderungen zu beachten, um die geringe Vertrauenswürdigkeit der beteiligten Vertragspartner zu berücksichtigen. Im Fall gegenseitig fremder Marktteilnehmer ist daher die Modellierung eines interaktiven Handels mit Abonnenten vorzuziehen.

Definition 4.6 (Abonnement-Modell) *Im Abonnement-Modell wird interaktiver Handel von Ressourcen mit identifizierbaren Mitgliedern mit Hilfe buchgeführter Konten betrieben.*

Teilnahmeanreize Anreize sind Voraussetzung für die Nutzung eines Marktsystems. Diese fundamentale Tatsache ist bei einer Reihe universitärer Projekte zwangsläufig unberücksichtigt geblieben, da sie keine langfristig funktionierenden Anreize zur Nutzung ihrer Infrastrukturen bieten konnten und wollten. Eine Buchführung und Abrechnung aller Teilnehmer, die unter Umständen nur winzige Ressourcenmengen ein- oder ausführen, ist aber schon allein aus Gründen der Organisation und Verwaltung nicht skalierbar möglich. Aus diesem Grund wird zur weitmöglichsten Reduktion der Transaktionskosten der Einsatz von Mikrowährungen[2] zur Entschädigung von Marktteilnehmern niedriger hierarchischer Ebenen des Systems vorgeschlagen. Ein Austausch und eine Auszahlung in realen Währungen erfolgt nur an Zwischenhändler in übergeordneten Hierarchien des Händlersystems für größere Mengen angehäufter Ressourcen. Zur Reduzierung der Transaktionskosten erhalten Anbieter niedriger Hierarchieebenen Mikrowährungen anstelle realer Währungen für die Bereitstellung minimale Ressourcenmengen.

Einführung – Die Suche und der Bezug von Informationen macht gegenwärtig einen hohen Anteil des Verkehrs des World Wide Webs aus, zumal ein Großteil dieser Informationen kostenlos zu erhalten ist. Deren Quantität und Qualität könnte jedoch weiter erhöht werden, wenn es geeignete Aufwandsentschädigungen für Informations-Anbieter im WWW gäbe. Viele Informationen sind wegen fehlender sicherer und unkomplizierter Zahlungsverfahren speziell für kleine Beträge unter einer Mark gar nicht im Netz verfügbar oder aber nur in einer minderwertigen kostenlosen Form. Bei den meisten dieser Informations-Angebote handelt es sich um Dienstleistungen, deren Einzelwert den Aufwand einer normalen finanziellen Transaktion nicht rechtfertigt. Für solche Leistungen wird daher eine *Mikrowährung* mit extrem geringen Transaktionskosten benötigt, so daß eine Abrechnung rentabel wird. Diese sogenannten Mikrowährungsverfahren sollen sich durch sehr geringe Transaktionskosten, eine vernachlässigbar geringe Latenz, eine extrem einfache Handhabung sowie eine möglichst universelle Akzeptanz auszeichnen, allesamt Punkte, die nur schwer gleichzeitig zu erreichen sind.

Es hat sich deshalb noch kein geeignetes Zahlungsverfahren im WWW zur Abrechnung geringwertiger Güter und Dienstleistungen wie Suchergebnisse, Nachrichten, Audio-, Video- oder Grafik-Dateien durchsetzen können. Die meisten *Mikrowährungen* scheitern an mangelnder kritischer Masse, fehlender Anonymität, komplizierter Handhabung und kostspieliger Markteinführung oder einer Kombination dieser Ursachen. Das einzig funktionierende Verfahren zur Finanzierung geringwertiger Inhalte im WWW stellt derzeit *Online* Werbung dar, eine Form der indirekten Verrechnung, die man als primitive Form einer *Mikrowährung* bezeichnen könnte.

Der folgende Abschnitt führt grundlegende digitale Zahlungsmittel und -technologien vor, erläutert wichtige Eigenschaften und Alternativen von Mikrowärungssystemen und stellt schließlich notwendige Eigenschaften von *Micro Payment* Systemen vor.

Mikrowährungssysteme – Wie die Gegenwart zeigt haben sich elektronische Zahlungsmittel

[2] wie Online Inhalten oder Informationen

nicht so weit verbreitet, wie es noch vor einigen Jahren euphorisch prognostiziert worden war. Hindernisse bei der Verbreitung digitaler Währungen waren vor allem Probleme bei der Akzeptanz und der Errichtung der notwendigen Infrastruktur für solche Zahlungssysteme. Die Infrastruktur ist insofern besonders problematisch, da sie im Gegensatz zu anderen Voraussetzungen sowohl für Endkunden als auch Händler einen bedeutenden Initialaufwand bedeutet, der nur aufgewendet wird, wenn der Erlös die Kosten rechtfertigt.

Das Interesse der Händler richtet sich in erster Linie auf die Zahl der erreichbaren Kunden und den eigenen Installationsaufwand. Die Kunden hingegen, deren Installationsaufwand meist von den Betreibern übernommen wird, berücksichtigen vor einer Entscheidung zugunsten eines neuen Zahlungssystems dessen Verbreitung, um bei einer möglichst großen Zahl von Händlern einkaufen zu können. Sie werden erst auf eine neue Zahlungsinfrastruktur umsteigen, wenn genügend Händler an das System angeschlossen sind. Die Händler wiederum werden das System erst dann unterstützen, wenn es von einer ausreichenden Zahl von Kunden genutzt wird.

Wegen dieser reziproken Abhängigkeit sind die meisten Feldversuche im Micro-Payment Bereich der letzten Jahre erfolglos abgebrochen worden. Es existieren zwar erfolgreiche Gegenbeispiele wie das Inkasso-System von T-Online, dem Nachfolger des BTX Systems der Deutschen Telekom, das jedoch einen Sonderfall darstellt, da es das geschlossene T-Online Netzwerk als Infrastruktur verwendet und auf offene Internet Technologien nicht übertragbar ist. Betreiber stehen nun vor einem Dilemma: Es sind neue Währungstechnologien und -systeme erforderlich. Diese benötigen aber offensichtlich neue Infrastrukturen, die jedoch aufgrund mangelnder Verbreitung nicht akzeptiert werden. Entweder müssen neue Technologien und Infrastrukturen mit aller strategischer Macht von Allianzen und Bündnissen durchgesetzt werden oder es werden neue Zahlungsverfahren entwickelt, die auf den bestehenden und akzeptierten Infrastrukturen aufbauen. Von letzteren Punkt handelt dieser Abschnitt.

Zahlungsgrößen – Laut IBM [Fra99] unterscheidet ihre *e-Business* Abteilung sechs verschiedene Kategorien elektronischen Zahlungsverkehrs:

1. Garantierte Schecks mit Größenordnungen über DM 5000,-
2. Kreditkarten mit Größenordnungen über DM 100,- [3]
3. Geldkarten-Zahlungen unter DM 100,-
4. Digitales Geld für Beträge unter DM 10,-
5. *Micro Payments* unter DM 1,-
6. *Pico Payments* unter 1 Pfennig

Zur augenblicklichen Verfügbarkeit von Zahlungsmitteln für die genannten Kategorien ist zu sagen, daß es für (1) keine existierenden Systeme gibt. Kategorie (2) wird durch das SET System abgedeckt. die Kategorien (3) und (4) werden wie der Name bereits sagt von Geld-Karte und Digitalem Geld übernommen. Für die Kategorie (5) stehen diverse Micro-Payment Verfahren zur Verfügung, die sich für die Zwecke eines Ressourcen-Handels am

[3] Allerdings ist eine Kreditkarten-Zahlung bereits ab DM 2,50 kostendeckend

besten eignen und nachfolgend genauer untersucht werden. Pico-Payment-System für (6) wird es laut IBM nicht geben, da die Transaktionskosten zu keinem (sinnvollem) Verhältnis zu den transferierten Beträgen stehen.

Zahlungsarten Kreditkarten-Zahlungen beinhalten nicht zu vernachlässigende Minimalgebühren pro Transaktion und sind daher für Zahlungen mit einem Betrag unter zehn DM ungeeignet. Als Alternative zu Kreditkarten-Zahlungen existieren bereits eine Reihe von traditionellen Zahlungsverfahren, die kleine und kleinste Beträge unterstützen, das Subskriptions-, das Werbe- und Mikrowährungen.

Subskriptions-Modell – Sogenannte *Bundlings* oder Abonnements fassen eine Reihe von Inhalten oder Angeboten geringen Werts zu einem höherwertigen Bündel zusammen, dessen Preis den Aufwand für den Einsatz aufwendigerer Zahlungsverfahren rechtfertigt. Der Kunde wird dabei nur ein einziges Mal für eine Reihe von Leistungen belastet.
Nachteile für den Nachfrager beim Subskriptions-Modells sind die zusätzliche Indirektion und die langfristige Bindung an den Anbieter. Abonnementen sind durch eine Subskription nicht mehr in der Lage, die für sie interessantesten Angebote im Markt zusammenzutragen und einzeln zu erwerben. Stattdessen müssen sie ein Paket von Leistungen kaufen, das ein Mittelsmann – also eine zusätzliche Indirektion zwischen Anbieter und Nachfrager – zusammenstellt. Zweitens ist ein Kunde durch eine Subskription längerfristig an einen Anbieter gebunden, auch wenn seine Qualität nachgelassen hat. Zudem bietet im WWW das Subskriptions-Modell, bei dem zahlende Abonnenten mit einem Paßwort Zugang zu geschützten Daten erhalten, in der Praxis zu wenig Sicherheit. Es hat sich herausgestellt, daß Paßwörter "geteilt" und von bis zu einem Dutzend Benutzern verwendet werden. Aus den genannten Gründen zeigen sich Kunden bei der Akzeptanz des Subskriptions-Modell relativ zögerlich.

Werbe-Finanzierung – Die Finanzierung über Werbung oder Sponsoring erfüllt praktisch alle Anforderungen an ein Zahlungssystem für Kleinbeträge. Da ihr Einsatz keine zusätzliche Infrastruktur benötigt, sind mit einem Schlag die Forderungen nach niedrigen Transaktionskosten, geringer Latenz und universeller Verwendbarkeit erfüllt. Darüberhinaus ist die Anwendung denkbar einfach. Dennoch sind Einkünfte aus der Werbe-Finanzierung beschränkt, so daß ihre Anwendungen nur für Angebote sinnvoll ist, die einen sehr geringen Wert – typischerweise im Pfennig-Bereich – haben. Gewöhnlich berechnen sich Werbeeinnahmen nach der Anzahl der Sichtkontakte (*Impressions*) oder der Clicks auf eine Werbefläche (*click-throughs*). Die Preise für Tausend Kontakte (*Cost per (roman) M*, CPM) reichen dabei von DM 10,- bis DM 100,-. Geringe Preise werden dabei für relativ heterogene zufällige Sichtkontakte bezahlt, während identifizierbare, sehr diversifizierte Zielgruppen höhere Preise erzielen. Die genannten Preise ergeben einen Wert von ungefähr 1 bis 10 Pfennigen pro Sichtkontakt mit der Werbefläche und damit dem angebotenen Inhalt. Inhalte und Angebote mit einem höherem Wert über zehn Pfennigen haben demnach keine Aussichten, sich über Werbung zu finanzieren. Es besteht also eine klaffenden Lücke zwischen dem durch Werbe-Finanzierung abdeckbaren Kosten und Zahlungen, die den Einsatz von Kreditkarten-Transaktionen rechtfertigen.

Mikrowährungssysteme – Micro-Payment-Systeme bieten sowohl dem Anbieter als auch dem Nachfrager von Angeboten und Inhalten Vorteile. Doch obwohl diese Systeme bereits seit einigen Jahren diskutiert werden, haben sie trotz des enormen Bedarfs den Durchbruch nicht geschafft. In erster Linie sollte sich eine Mikrowährung durch sehr geringe Transaktionskosten und -latenzen und eine extrem einfache Anwendung auszeichnen. Insbesondere an einer universellen Verfügbarkeit und Anwendbarkeit der Währung scheitern immer wieder Modellversuche. Benutzer gehen zu Recht davon aus, mit einem Micro-Payment-Verfahren bei praktisch allen Händlern einkaufen zu können, während Händler höchstens ein oder zwei verschiedene Systeme unterstützen können. Daraus ergibt sich, daß langfristig nur für sehr wenige Mikrowährungen ein Markt vorhanden ist. Diese Tatsache macht es selbst für technisch ausgereifte Systeme sehr schwer, sich im Markt zu etablieren und eine kritische Masse – die Basis für die Entscheidungen der Käufer und Händler ist, ein bestimmtes System zu unterstützen oder nicht – zu erreichen.

4.3 Das Locust Modell

Es existieren Millionen von Computern im Internet, die eine mehr oder weniger leistungsfähige temporäre oder feste Anbindung ans WWW besitzen (Modem, ISDN, DSL, Standleitung), jedoch sehr unregelmässig ausgelastet sind. Moderne PCs von WWW Nutzern besitzen während des Surfens noch große Leistungsreserven, Rechner in Rechenzentren oder Firmen werden gewöhnlicherweise weniger als 80 Stunden wöchentlich genutzt und liegen während der Nacht und am Wochenende meist vollständig brach. Zur Lösung der genannten Probleme wird ein System benötigt, das die ungenutzte Rechenleistung solcher Rechner Nachfragern, die preiswert zusätzliche, skalierbare Rechenleistung benötigen, zur Verfügung stellt.

4.3.1 Konzept

Zentraler Gedanke des Locust (LOwcost Computing Utilizing Skimmed idle Time) Modells ist eine Infrastruktur zur Nutzung brachliegender Rechenleistung ans Internet angeschlossener Rechner für darauf aufsetzende, verteilte und parallele *Tasks*. Die Infrastruktur besteht aus dem Locust Server, sowie einem hierarchischen Netzwerk von Multiplikatoren (wie zum Beispiel Rechenzentren und *Content Providern*). Die *Tasks* werden von einer zweiten Software, dem Locust Client, realisiert.

Der Client wird mit Hilfe der Multiplikatoren auf die *Web*-Terminals oder Multiplikator-Rechner verteilt und von diesen ausgeführt. Durch die Ausführung des Clients wird die lokale, überschüssige Rechenleistung abgeschöpft und der verteilte und parallele *Task* ausgeführt. Als Gegenleistung für ihre Rechenleistung erhalten die *Web*-Terminals Zugang zu einfachem, kostenpflichtigen *Online*-Inhalten und -Dienstleistungen. Die Multiplikatoren werden für ihre Leistungen von Locust mit einer anderen Gegenleistung – zum Beispiel einem Vorrat an Rechenleistung – entlohnt. Die abgegebene Rechenleistung erfüllt bei der Abrechnung damit die Funktion einer Mikrowährung. Ziel des Locust-Clients ist die Abschöpfung opportunistischer Ressourcen der, unter Umständen nur sehr kurzfristig, teilnehmenden *Web*-Terminals.

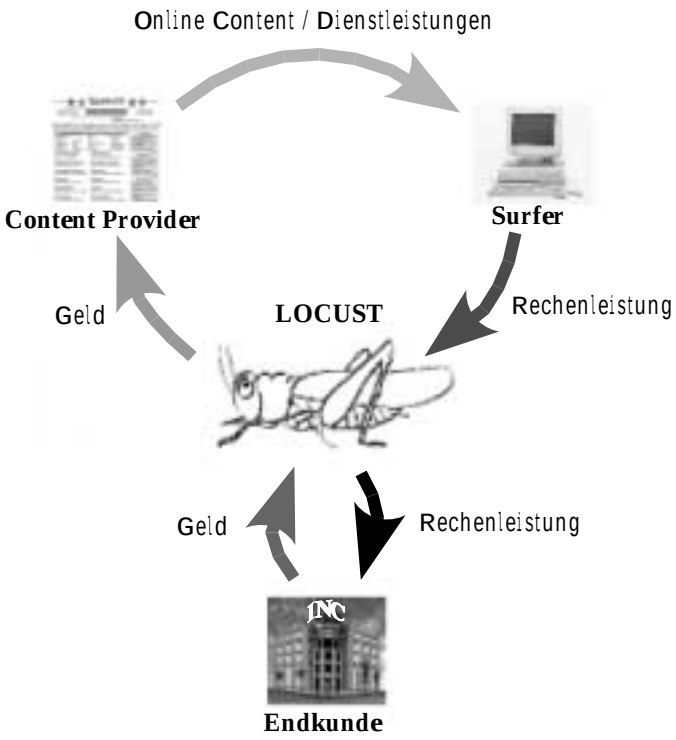

Abbildung 4.2 Dienstleistungs- und Geldfluß im Locust Modell

Neben opportunistischen Ressourcen werden darüberhinaus zur Realisierung zuverlässiger Dienste und zur Aufrechterhaltung der Infrastruktur selbst dedizierte und zuverlässige Ressourcen benötigt. Diese werden durch den sogenannten *Locust-Markt* zur Verfügung gestellt, der temporäre und stationäre Knoten eines Weitverkehrsnetzes stunden- bis tageweise anmietet. Die (dedizierten) Ressourcen dieser Knoten sind hauptsächlich zur Implementierung der Locust-Infrastruktur wie zum Beispiel der Locust-Server vorgesehen, können aber auch bei entsprechender Nachfrage zur Realisierung komplexerer paralleler und verteilter Anwendungen verwendet werden, die höhere Anforderungen an Rechenleistung, temporäre Plattenkapazität oder Sicherheit besitzen. Die längere Verfügbarkeit dieser Ressourcen rechtfertigt überdies auch den initialen Transport großer Code- und Datenmengen an den Einsatzort.

4.3.2 Teilnehmer

Die logischen Beziehungen zwischen WWW Terminals, Multiplikatoren, Nachfragern und der Locust Infrastruktur sind im folgenden genauer erläutert(vgl. Abbildung 4.2):

1. Locust ist Abnehmer überschüssiger, ungenutzter Rechenleistung von *Web*-Terminals. Dies erfolgt über eine spezielle Software, dem Locust-Client, der zusam-

men mit dem *Online-Content* auf den Terminal des Surfers geladen und ausgeführt wird.

2. Um eine sehr große Anzahl von *Web*-Terminals zu erreichen, setzt Locust Rechenzentren und *Content-Provider* als Multiplikatoren ein, die die Rechenleistungen ihrer Terminals sammeln, abrechnen und gebündelt an das Locust System weiterreichen. Die Multiplikatoren vervielfachen die Reichweite des Clients indem sie ihn in ihre Inhalte oder Rechner mit hohen Zugriffszahlen oder hoher Reichweite einbetten.

3. Die gewonnene Rechenleistung verwendet Locust zur Realisierung von verteilten und parallelen Aufgaben, die zu einem Bruchteil der sonst üblichen Kosten erbracht werden können.

4. Einem Teil der gesammelten Rechenleistung gibt Locust an seine Multiplikatoren weiter, der Rest wird als Kommission einbehalten und an Nachfrager nach Rechenzeit vermarktet.

5. Die *Web*-Terminals werden für die Abgabe von (ungenutzter) Rechenleistung von den Multiplikatoren mit (eigentlich kostenpflichtigen) Inhalten und Dienstleistungen entlohnt.

Aufgrund dieses Kreislaufes kann das Locust Modell auch als indirektes *Micro-Payment-System* mit Rechenleistung als *Mikrowährung* bezeichnet werden[May99b].

4.3.3 Teilnehmernutzen

Nachdem das Locust Modell auf der Akzeptanz von drei Gruppen von Teilnehmern basiert, ist eine genaue Analyse der Bedürfnisse dieser Nutzergruppen notwendig. Die gleichzeitige Erfüllung der individuellen Gruppenwünsche ist nicht einfach, gelingt dieser Spagat jedoch, ist das mit großer Sicherheit gleichbedeutend mit dem Durchbruch und der Profitabilität des Locust Modells.

1. Web-Terminals wünschen sich kostenlosen Zugang zu qualitativ hochwertigem *Online* Inhalten, wie sie sonst nur kostenpflichtig oder unter Tolerierung von massiver Werbung abrufbar wären. Falls Inhalte nur kostenpflichtig zu erlangen wären, wünschen sich in einer Umfrage [Res] aus dem Jahr 1998 30,5 % der WWW Teilnehmer ein anonymes Zahlungsmittel, während 49 % dazu keine Meinung haben, d.h. nur 20,5 % der Nutzer legen auf Anonymität keinen Wert. In der gleichen Umfrage äusserten sogar 88,1 % der Surfer ihre besondere Wertschätzung der anonymen Navigation im WWW [Res].

   Durch Verwendung von Rechenleistung als Mikrowährung erlangen WWW Terminals kostenlos und völlig anonym Zugang zu den von ihnen gewünschten Inhalten – jedoch nur falls der Ersteller der Inhalte ein Locust Multiplikator ist.

2. Multiplikatoren (Rechenzentren, *Content-Provider*) suchen nach funktionierenden Möglichkeiten zur leistungsgerechten, zuverlässigen Refinanzierung ihrer Inhalte oder Dienstleistung als Alternative zu schwerfälligen und unsicheren Methoden wie Abonnements und Werbung.

Als Locust Multiplikator erreichen *Content-* und *Service-Provider* leistungsgerechte, zuverlässige Refinanzierung ihrer Inhalte und Dienste. Einkommensausfälle aufgrund der abschreckenden Wirkung von Abonnementgebühren oder der schlechten Zahlungsmoral von Vermarktern von Werbung sind ausgeschlossen.

3. Nachfrager wünschen sich preiswerte, skalierbare und hochwertige zusätzliche Rechenleistung, auf die direkt und sehr dynamisch zugegriffen werden kann. Derartige Dienstleistungen sind bisher nur über Neuanschaffungen von Hardware in großen diskreten Schritten und sehr kostenintensiv zukaufbar. Darüberhinaus erfordert das *Outsourcing* von IT Dienstleistungen einen verwaltungstechnischen Aufwand, der zu zeitlichen und inhaltlichen Reibungsverlusten bei der Nutzung führt.

   Locust Endkunden erhalten die gewünschten Rechenleistungen in Form von stufenlosen Erweiterungen der eigenen Rechenkapazität zum Beispiel zum parallelen *Rendering*.

4.3.4 Wettbewerb

Das Locust Modell steht in Konkurrenz zu folgenden Infrastrukturen:

Mikrowährungen – Mikrowährungen zeichnen sich durch ein günstiges Verhältnis zwischen dem zu zahlenden Betrag und den erforderlichen Transaktionskosten aus. Es existieren unzählige, ehrgeizige Versuche kleiner, aber auch großer Firmen [Ser98], [Cyb98], [Com97], [Net98], eine Mikrowährung im WWW zu positionieren. Alle diese Versuche sind an mangelnder kritischer Masse, fehlender Anonymität, komplexer Handhabung und kostspieliger Markteinführung oder einer Kombination dieser Ursachen gescheitert. Das einzig funktionierende Verfahren zur Finanzierung geringwertiger Inhalte im WWW stellt derzeit *Online* Werbung mit den bereits skizzierten Nachteilen dar.

Rechenleistung als Mikrowährung besitzt dagegen sämtliche Eigenschaften, die anderen Verfahren zum Durchbruch fehlen: Die Bezahlung mit Rechenleistung ist für das *Web*-Terminal anonym sowie technisch und logisch transparent, d.h. es ist keine zusätzliche Hard- oder Software[4] zur Abrechnung notwendig. Die Transparenz geht sogar soweit, daß das Terminal unter Umständen die Transaktion selbst gar nicht bewußt wahrnimmt. Eine bedeutende kritische Masse zur Etablierung als "branchenübliches" Zahlungsmittel ist durch Einsatz der Locust Multiplikatoren – die ebenfalls ohne besondere Voraussetzungen am Zahlungssystem teilnehmen können – erreichbar. Ein weiterer Vorteil des Konzepts ist, daß Rechenleistung bei den meisten Benutzern im Überfluß vorhanden ist und dieser "Rohstoff" bei Nichtgebrauch verfällt. Aus dieser Tatsache folgt eine erhöhte Bereitschaft der Nutzer, Rechenleistung für *Online-Content* "auszugeben" [5], die wiederum zu erhöhtem Konsumverhalten im Web führen kann.

[4] wie zum Beispiel ein Browser *Plug-In*

[5] im Gegensatz zu realem Geld, das bei Nichtgebrauch nicht verfällt, sondern nach Möglichkeit gespart wird

Rechen-Infrastrukturen und -zentren (z.B. `SETI@home`, `distributed.net`)
: Rechenzentren waren die ersten Dienstleister, die spezielle IT Leistungen für Dritte angeboten haben. Aufgrund ihrer notwendigen Größe sind sie meist in öffentlicher Hand und nicht auf die Erfordernisse des freien Wettbewerbs vorbereitet, das heißt die Inanspruchnahme ihrer Dienste ist bürokratisch, ineffizient und langsam.

  Die einzigen verteilten Rechen-Infrastrukturen im WWW, `SETI@home` [SET] und `distributed.net` [dis], sind Forschungs- und Nutzergemeinschaften, die sich ausschließlich ganz bestimmten, eng umrissenen Zielen widmen. `SETI@home` hat seine über 1,5 Millionen freiwilligen Teilnehmer zur Suche nach außerirdischen Spuren in kosmischer Strahlung mobilisiert. Dazu werden die Daten eines Radioteleskops in kleine Pakete zerlegt, an die Teilnehmer verschickt, die diese Daten mittels eines speziellen Programms analysieren und die Ergebnisse zurückschicken. Mit Hilfe von auf die jeweilige Computer-Plattform optimierten Testprogrammen und einer hohen Teilnehmerresonanz erreicht `SETI@home` auf diese Weise nach eigenen Angaben eine Gesamtrechenleistung von zehn Teraflops.
  `distributed.net` dagegen hat sich dem wettbewerbsmäßigen Brechen kryptographischer Verfahren (wie DES-II, DES-III und RC-5 Schlüssel) gewidmet und setzt auf hoch-optimierte Clients sowie eine hohe Teilnehmerzahl. Die auf die jeweilige Computerarchitektur optimierten Testprogramme von `distributed.net` und der intensive Wettbewerb seiner Teilnehmer untereinander bei der Suche nach dem richtigen Schlüssel ermöglicht ebenfalls eine sehr hohe akkumulierte Rechenleistung von über 100 Giga-Keys (Schlüssel) pro Sekunde. Zur Realisierung anderer rechenintensiver Aufgaben sind beide Systeme aufgrund des hohen Optimierungsgrades und fehlender Teilnehmer bei Aufgaben, die nicht ideeller Natur sind, jedoch völlig ungeeignet.

  Das Locust Modell dagegen bietet seinen Endkunden über die bei Multiplikatoren und *Web*-Terminals installierte Basis eine multifunktionale, generische Plattform für das Lösen rechenintensiver Aufgaben über das WWW an. Durch innovativen Einsatz von Rechenleistung als Mikrowährung und dem Multiplikator-Effekt erschließt das Locust Modell die Rechenleistung von Millionen von *Web*-Terminals, ohne jedoch mit aufwendiger und kostenintensiver Verwaltung, Abrechnung und Support belastet zu werden. Diese Dienstleistungen werden von den fachkundigen Multiplikatoren übernommen. Eine direkte Kommunikation zum Aufbau und Wartung der Plattform ist nur noch mit ihnen notwendig. Auf diese Weise kann das Locust Modell die gleiche oder höhere akkumulierte Leistung wie die genannten Rechen-Infrastrukturen erbringen, ohne auf eine einzige, unter Umständen unprofitable, Dienstleistung festgelegt zu sein.

4.4 Zusammenfassung

Aufbauend auf den Modellvoraussetzungen wurde das Ressourcen-Handelsmodell Locust entworfen und vorgestellt. Als notwendige Voraussetzungen für eine erfolgreiche Etablierung eines Ressourcen-Handels wurden folgende Prinzipien identifiziert:

Ausnutzung multiplikativer Effekte – Die enorme Größe der anvisierten Zielgruppe laienhafter WWW Benutzer und ihre effiziente Verwaltung wird durch Ausnutzung multiplikativer Effekte beherrschbar gemacht. Als Multiplikatoren werden Zwischenhändler eingesetzt, die viele Rechner oder Benutzer verwalten und dadurch die Reichweite des ressourcen-sammelnden Locust-Clients vervielfachen. Sie sind für die Ressosurcensammlung und Abrechnung der ihnen unterstellten Rechner und Nutzer verantwortlich. Der durch mehrere Ebenen solcher hierarchisch organisierten Zwischenhändler erreichbare exponentielle Multiplikationsfaktor reduziert den hierfür notwendigen Verwaltungs- und Serveraufwand um ein Vielfaches. Voraussetzung zur Nutzung multiplikativer Effekte ist ein asymmetrische Modellierung der beteiligten Marktteilnehmer.

Ausnutzung der Lokalität – Die hierarchisch organisierten Zwischenhändler und ihre Teilmärkte werden dezentral mit Hilfe marktbasierter Zuteilungsverfahren verwaltet, die zum größten Teil auf lokalen Informationen beruhen. Dabei bleibt es den Multiplikatoren überlassen, lokale Besonderheiten ihrer Teilmärkte zu berücksichtigen. Verschiedene Teilmärkte besitzen sehr unterschiedliche Anforderungen an Transaktionskosten, Sicherheitsanforderungen sowie Teilnahmebedingungen und -anreize. Lokale Politiken zur Sammlung, zum Handel und zur Abrechnung von Ressourcen haben dabei stets zum Ziel, die Kosten zur Verwaltung und Aufrechterhaltung des Ressourcenhandels so weit wie möglich zu reduzieren.

Anwendung von Rekursivität – Zum Aufbau und zur Verwaltung hierarchisch organisierter Zwischenhändler und Teilmärkte findet das Rekursivitätsprinzip Anwendung. Inhomogene Märkte oder überlastete Serverdienste werden rekursiv in kleinere und homogenere Einheiten aufgeteilt bis eine weitere Aufteilung nicht sinnvoll oder nicht mehr notwendig ist. Dieser Zustand ist gleichbedeutend mit der Tatsache, daß die entstandenen Teilmärkte homogen geworden sind und ihre Serverdienste sich in einem Gleichgewichtszustand befinden. Die alten und neuen Märkte werden hierarchisch gekoppelt und dezentral von den untergeordneten Teilmärkten und Serverdiensten aus unter Ausnutzung der Lokalität verwaltet.

Das prototypische Ressourcenhandels-System Locust erfüllt die angegebenen Anforderungen an eine geeignetes Modell zur Sammlung und zum Handel überschüssiger Rechenressourcen [May99c]. Das ihm zugrundeliegende Modell ist inhärent asymmetrisch und in allen Teilaspekten zugunsten der zahlenmässig gegenüber Nachfragern im Vorteil liegenden Anbietern von brachliegender Rechenleistung ausgerichtet. Sämtliche mit Anbietern in Verbindung stehende Entitäten und Transaktionen sind aus deren Sichtweise modelliert und auf deren Bedürfnisse optimiert.
Die Ausnutzung multiplikativer Effekte in diesem Modell manifestiert sich ebenfalls im hierarchisch organisiertem Netz von Händlern und Zwischenhändlern, die jeweils einen Teilmarkt verwalten, der bezüglich einer oder mehreren Eigenschaften homogen ist. Diese Homogenität wird von den Zwischenhändlern ausgenutzt, um die Transaktionskosten und die Zutrittsbarrieren zur Marktteilnahme an lokale Gegebenheiten anzupassen und auf ein möglichst niedriges Niveau zu senken. In homogenen Teilmärkten, die unter der gleichen Administration oder sogar unter dem gleichen Besitzer stehen, sind die Anforderungen an Sicherheit, Buchführung oder (finanzielle) Anreize ungleich niedriger als in einem Markt

untereinander fremder Teilnehmer. Auf diese Weise können die Hindernisse zur Marktteilnahme und die Kosten hierfür umso niedriger gehalten werden, je größer die Lokalität und damit die Homogenität des Teilmarktes ist.

Haupt-Zielgruppe des Locust Systems sind gelegentliche Nutzer des WWW, sogenannte *Web*-Terminals, die durch den Besuch einer Web-Seite oder *Web-Site* automatisch und anonym ungenutzte Rechenressourcen zur Verfügung stellen. Eine Anmeldung oder Registrierung, Installation oder selbst der bewußte Start einer Anwendung ist hierzu nicht notwendig. Verwaltet wird die große Zahl anonymer Anbieter von Rechenressourcen über ein hierarchisch organisiertes Netz von Haupt- und Zwischenhändlern, die in mehreren hierarchischen Ebenen die Sammlung, Abrechnung und den Handel der ihnen untergeordneten Händler und, auf unterster Ebene, WWW Teilnehmer verwalten.
Die Zwischenhändler von Locust sind Marktteilnehmer, die über viele Rechner oder Benutzer verfügen wie Rechenzentren-Betreiber und Inhalte-Ersteller (*Content-Provider*) und als Teilmarkt mit Ressourcenüberschuß an der Locust-Infrastruktur teilnehmen. Diese binden den sogenannten Locust Client zur Abschöpfung der lokalen brachliegenden Rechenleistung in ihre Rechner und Web Seiten ein und erhöhen dadurch die Reichweite des Clients um ein Vielfaches (Multiplikatoreffekt). Als Gegenleistung erhalten die Multiplikatoren eine entsprechende Gegenleistung in Form von aggregierter freier Rechenzeit oder Geld.
Am Locust System teilnehmende *Web*-Terminals werden durch den Inhalt der besuchten Seiten für die Bereitstellung von Rechenressourcen entschädigt. Damit erfüllt die abgegebene Rechenleistung die Funktion einer Mikrowährung, wie sie bereits seit langem zur Eindämmung der ständig wachsenden Werbe-Finanzierung im WWW gefordert wird [Mic99a]. Ein Großteil der *Web*-Terminals wird dabei gar nicht wahrnehmen, daß sie einen Teil ihrer Ressourcen für den Zugriff auf *Online*-Inhalte wie Börsenkurse, Suchanfragen oder Nachrichten eintauschen.
Locust Endkunden schließlich nutzen die Locust-Infrastruktur bzw. die von ihr zur Verfügung gestellte Rechenzeit zur preiswerten, stufenlosen und dynamischen Erweiterung ihrer begrenzten Rechenkapazitäten für rechenintensive verteilte und parallel Anwendungen wie beispielsweise dem *Rendering* eines animierten Filmes.

Locust tritt in einer solchen Ressourcenhandels-Szenerie als Vermittler zwischen Angebot und Nachfrage freier Rechenleistung im World Wide Web auf und erzielt durch die Schaffung von Mehrwert auf Basis dieses Ausgangs-Rohstoffs Gewinn. Da es sich bei Locust jedoch um ein akademisches System handelt, ist ein Handel von Ressourcen mit realen Währungen nicht möglich. Als Ersatzwährung wird stattdessen die angesammelte Rechenzeit selbst verwendet, die in einem bestimmten Verhältnis[6] vergütet wird. Das bedeutet, Multiplikatoren erhalten die Hälfte ihrer gesammelten Ressourcen als Guthaben zur spätere Verwendung. Der von Locust als Kommission für die Vermittlung und Organisation des Handels einbehaltene Anteil an Ressourcen ist zur internen Verwendung – parallele Beispielanwendungen oder Teile der Locust Funktionalität selbst – vorgesehen.

In einem Real-Betrieb jedoch – außerhalb eines wissenschaftlichen Umfelds – ist Locust als Kommissionär und Weiterverarbeiter in der Lage, durch die Schaffung von Mehrwert auf den rohen Eingangsressourcen Gewinn zu erwirtschaften. Der erzielte Mehrwert besteht aus der Schaffung einer homogenen Ressource Rechenzeit, die direkt für verteilte

[6] zum Beispiel zwei zu eins

und parallele Aufgaben konsumiert werden kann. Ebenfalls denkbar ist die Weiterverarbeitung zu verschiedenen Basisdiensten – "elektrischer Strom" – zum Bau und Unterhalt von Meta-Computing-, Web-Computing- und Rechennetz-Infrastrukturen.

Kapitel 5

Locust Architektur und Implementierung

Locust hat das Ziel, möglichst viel Rechenleistung zu sammeln und gebündelt zur Lösung verteilter und paralleler Aufgaben einzusetzen. Während andere verteilte Web-Computing Infrastrukturen [dis], [SET] dabei auf spezielle, hoch-optimierte Client-Software und einen fachkundigen Teilnehmerkreis setzen, verfolgt Locust eine andere Strategie.

Um sehr viele Teilnehmer, die Rechenleistung zur Verfügung stellen, – im folgenden als Leistungserbringer bezeichnet – zu gewinnen, ist Locust in *Web*-Seiten und -Dienstleistungen integriert und nutzt damit die enorme Popularität des WWW aus. Locust erreicht damit einen weitaus größeren Multiplikationsfaktor als herkömmliche Web-Computing-Systeme.

Voraussetzung für eine derart große Zahl von Teilnehmern ist jedoch, daß die Zutrittsbarrieren und Transaktionskosten dieser Benutzergruppen viel niedriger als bei vergleichbaren Systemen sonst liegen, und zwar um so niedriger, je höher der Multiplikationsfaktor im Locust Modell ist (Vgl. Abschnitt 4). Mit einem hohen Multiplikationsfaktor geht stets eine niedrige Ebene des Teilnehmers im hierarchischen Locust Modell einher. In anderen Worten bedeutet dies, daß die Anforderungen an technisches Verständnis der *Web*-Terminals extrem gering sein müssen, um ihnen die Teilnahme zu ermöglichen, während die Erfordernisse, die Multiplikatoren abverlangt werden, deutlich höher sein können und zur Erfüllung ihrer Funktion im Locust Modell auch sein müssen.
Insbesondere für *Web*-Terminals auf der niedrigsten Hierarchieebene des Locust Modells sollte die Bezahlung mit Rechenleistung anonym sowie technisch und logisch transparent sein, d.h. es darf keine zusätzliche Hard- oder Software[1] zur Abrechnung notwendig sein. Die Transparenz geht sogar soweit, daß das *Web*-Terminal unter Umständen die Transaktion selbst gar nicht bewußt wahrnimmt.

5.1 Entwurfsziele

Dieser Abschnitt behandelt allgemeine Überlegungen zur technischen Umsetzung des Locust Modells. Um das Locust Modell zu realisieren, sind eine Reihe von Voraussetzungen notwendig. Der hohe Multiplikationsfaktor setzt eine extreme Skalierbarkeit der Gesamtarchitektur voraus, weil es sich potentiell um Hunderttausende bis Millionen von Teilnehmern handelt. Die hohe Zahl, unter Umständen sehr laienhafter, Teilnehmer bedingt wiederum eine weitestgehende Vereinfachung des Anmelde- und Teilnahmeprozesses. Da

[1] wie zum Beispiel ein Browser *Plug-In*

eine solch hohe Zahl von Teilnehmern kaum mit individuellen Benutzerkonten abgewickelt werden kann, sind zur Beherrschung hoher Teilnehmerzahlen der Verzicht auf Benutzerkonten sowie Installation und Interaktion zur Teilnahme, zugunsten anonymer und automatischer Teilnahme, notwendig.

Traditionelle Hochleistungssysteme erreichen ihre Leistungsfähigkeit durch Spezialisierung auf eng umrissene Aufgabengebiete mit Hilfe hoch-optimierter Architektur (MPP, Vektorrechner), Sprachen (OpenMP, HPF) oder Software (Maschinensprache-Optimierungen). Locust ist als Plattform für verteilte und parallele Problemstellungen konzipiert und sollte daher über wiederverwendbare und portable Software implementiert werden, die die Anpassung an neue Algorithmen unterstützt und vereinfacht.

Aufgrund der hierarchischen Struktur und der unterschiedlichen Motivation der verschiedenen Teilnehmer basiert der Erfolg von Locust auf der Akzeptanz von drei Gruppen von Teilnehmern, *Web*-Terminals, Multiplikatoren und Endnutzern. Daher müssen alle individuellen Zielgruppenwünsche gleichzeitig erfüllt sein. Die Benutzerfreundlichkeit und Nützlichkeit des Systems ist also ebenfalls wichtiges Entwurfsmerkmal.

Nicht zuletzt sollte eine Implementierung ausreichend schnell sein, um parallele und verteilte Berechnungen in vergleichbarer Zeit – jedoch zu einem Bruchteil der üblichen Kosten – bewältigen zu können als ähnliche Systeme. Zusammengefaßt soll die Locust Implementierung die folgenden Entwurfsziele erfüllen:

- Hohe Multiplikation und Skalierbarkeit – Die Software sollte eine automatische, transparente Teilnahme an der Infrastruktur nach dem Blutspende-Modell (vgl. Definition 4.5) ermöglichen. Die Ausrichtung auf *Web*-Terminals führt zu Teilnehmerzahlen, die eine redundante und extrem hoch skalierende Architektur notwendig macht.

- Wiederverwendbarkeit – Die Infrastruktur sollte als Plattform für eine Vielzahl verteilter und paralleler Probleme dienen können. Die Entwicklung und Pflege eines auf eine einzige hoch-optimierte Anwendung ausgerichteten Rahmenwerks macht angesichts der Dynamik und Schnellebigkeit des WWW und ihrer Technologien keinen Sinn.

- Portabilität – Der clientseitige Anteil der Infrastruktur sollte ohne zusätzlichen Aufwand auf vielen verschiedenen Plattformen lauffähig sind. Diese Anforderung ist sowohl im Hinblick auf die weitere Wartung als auch auf eine einfache und intuitive API (*Application Programmer Interface*) der Locust Infrastruktur für Entwickler bedeutsam.

- Teilnehmernutzen und Benutzerfreundlichkeit – Nachdem das Locust Modell auf der Akzeptanz von drei Gruppen von Teilnehmern basiert – *Web*-Terminals, Multiplikatoren und Endnutzer –, ist die gleichzeitige Erfüllung dreier individueller Gruppenwünsche notwendig. Erst dann ist ein stabiler Geld- und Dienstleistungsfluß etabliert.

- Akzeptable Geschwindigkeit – Trotz der bereits genannten Nebenbedingungen sollte die gesammelte Rechenleistung auch moderate zeitliche Anforderungen der Endnutzer erfüllen. Angesichts des geringen zu erwartenden Preises der mit Locust erzielten Rechenleistung sollte ein eventueller Geschwindigkeitsverlust nicht den preislichen Vorteil zunichte machen.

5.1.1 Skalierbarkeit

Multiplikation Um die Transaktionskosten bei einer sehr großen Zahl von Teilnehmern möglichst niedrig zu halten, sieht das Locust Modell anonyme, automatische Teilnahme sowie die transparente Entlohnung von *Web*-Terminals mit *Online* Inhalten und Dienstleistungen vor. Die Führung von Konten für Ressourcen und Geld ist dadurch nicht notwendig, was zu einer Reduktion der Teilnahmevoraussetzungen und Transaktionskosten führt. In einer solchen Szenerie ist ein automatischer Handel von Ressourcen mit anonymen Laufkunden einem mitgliederbasierten Teilnahmeverfahren mit Kontoführung und -abrechnung vorzuziehen.
Aus den genannten Gründen sollte die Software eine automatische Installation und anonyme Teilnahme an der Infrastruktur nach dem Blutspende-Modell (vgl. Definition 4.5) ermöglichen.

Skalierbarkeit Um die geforderte sehr hohe Skalierbarkeit zu unterstützen, sollte bei der Kommunikation zwischen Client und Server sowie bei der Benutzerverwaltung soweit wie möglich auf einen umfangreichen und teuren Status[2] verzichtet werden. Das bedeutet, nach Möglichkeit auf teure, hochschichtige Protokolle wie TCP zugunsten leichtgewichtiger Protokolle wie UDP (*User Datagramm Protocol*) zu verzichten. UDP bietet folgende Vorteile:

- UDP ist verbindungslos und daher sehr leichtgewichtig.
- UDP benötigt zur Übertragung der gleichen Informationsmenge weniger Bandbreite als TCP.
- UDP Kommunikation ist dadurch schneller.

Der einzige Nachteil von UDP – es bietet keine garantierte Auslieferung der Daten– ist angesichts der erwarteten hohen Teilnehmerzahl leicht zu verschmerzen und kann mit Hilfe der ab Java 1.1 verfügbaren UDP Bibliothek mit *Time-outs* leicht behoben werden. Bei einer ausbleibenden Antwort kann der Client oder der Server nach einer gewissen Zeit die Pakete erneut verschicken.

5.1.2 Wiederverwendbarkeit

Nachfolgend wird zwischen der Wiederverwendbarkeit des Codes und der Wiederverwendbarkeit der Infrastruktur für verschiedene Aufgaben unterschieden. Allgemein erreicht man Wiederverwendbarkeit durch Modularität, also die Trennung aufgabenspezifischer Code-Teile in eigene Module.

Wiederverwendbarkeit der Infrastruktur Um die Infrastruktur wiederverwendbar zu machen, muß sie von der speziellen Anwendung getrennt werden. Der Locust-Client (das Applet, das vom *Web*-Terminal ausgeführt wird) ist unabhängig von einer speziellen Aufgabe lauffähig. Er stellt lediglich die Kommunikation mit dem Server und eine minimale grafische Benutzeroberfläche (*Graphical User Interface*, GUI) zu Verfügung. Auch auf

[2] im Sinne von Kontext

Serverseite existiert diese Trennung. Der Server übernimmt die Kommunikation mit dem Client, bietet eine einfache Benutzeroberfläche für den Locust Administrator und verwaltet Module für Locust Anwendungen, sogenannte *Jobs*.

Wiederverwendbarkeit der Anwendungen Der geschaffenen Code sollte soweit möglich eine Wiederverwendung erlauben und damit die Portierung neuer Anwendungen auf Locust erleichtern. Dies geschieht am besten mit Hilfe von semi-abstrakten[3] Klassen, die als Schablone für neue Anwendungen dienen und vom Anwendungsprogrammierer nur noch ergänzt werden müssen. Im günstigen Fall ist dabei nur noch die `run()` Methode zu ergänzen, um das Standard Java Interface `Runnable` zu implementieren. Diese Vorgehensweise garantiert, daß die Anwendung als *Thread* im Browser des Clients – gleichberechtigt neben anderen, bereits laufenden Applets – ausgeführt wird.

5.1.3 Portabilität

Hinsichtlich der Portabilität läßt sich zwischen der Portabilität des Clients und der Portabilität des Servers unterscheiden.

Portabilität des Clients Angesichts der Vielfalt unterschiedlicher Rechnerarchitekturen und Betriebssysteme im Internet ist ein portabler Client unbedingte Voraussetzung für Web-Computing-Systeme. Derzeit existieren jedoch nur wenige leistungsstarke Technologien, die standardmäßig in vielen Browsern integriert sind und damit eine weite Verbreitung im Internet haben. Anforderungen an eine solche Technologie sind eine umfangreichen API bzw. eine vollständige Programmiersprache, eine plattformunabhängige und preiswerte Verfügbarkeit im Sinn einer in 2.4 definierten Infrastruktur sowie das Vorhandensein von bedeutenden Legacy-Bibliotheken und -Anwendungen. Von den clientseitig verfügbaren Technologien Java [AG96], ActiveX [Mic00b], VBScript [Mic99b], JavaScript [Fla98], Macromedia Flash [Mac99] kommt nur Java in Betracht. Diese Technologie vereint Funktionalität, Leistungsstärke und weite Verbreitung. Java bietet eine "vollständige" Programmiersprache, die kostenlos auf jeder Plattform verfügbar ist und es existieren weltweit mehrere Tausend Anwendungen, die zum größten Teil ebenfalls kostenfrei erhältlich sind.

Portabilität des Servers Die Portabilität des Servers ist weniger wichtig als die Portabilität des Clients. Die Wahl der Implementierungssprache ist davon abhängig, welche Architekturen zur Verfügung stehen, wieviele Server man einsetzen möchte und welche Gesamtleistung erreicht werden soll. Da Locust nicht auf eine bestimmte Zielarchitektur festgelegt sein soll, wurde der Server ebenfalls in Java implementiert und ist somit leicht portabel. Eine gemeinsame Implementierungssprache verringert darüberhinaus den Aufwand für die Kommunikation zwischen Client und Server.

Zeigt sich, daß der Server aufgrund hoher Last zum Engpaß des Systems wird, ist ein schneller, in nativem Code vorliegender Server zur Problemlösung naheliegender, jedoch läßt sich mehr Leistung auch mittels *Clustering*, also durch mehrere parallel laufende

[3] bzw. semi-konkreten Klassen

Java Server, erreichen. Dieser Ansatz wird durch einen portablen, in Java implementierten Server erleichtert, da dann dabei keine Rücksicht mehr auf die Zielarchitektur mehr genommen werden muß.

5.1.4 Teilnehmernutzen und Benutzerfreundlichkeit

Die Erfüllung individueller Teilnehmerwünsche sowie der Benutzerfreundlichkeit ist für den Erfolg jedes Systems notwendig. Bei Locust betreffen diese Punkte drei wichtige Teilnehmergruppen des Locust Modells:

Administratoren – Der Administrator legt viel Wert auf geringen Ressourcenverbrauch, hohe Zuverlässigkeit und Sicherheit des Locust Servers. Darüberhinaus werden automatisierbare Abläufe und eine offene, transluzente Installation gern gesehen. Die Benutzerfreundlichkeit des Locust Servers dient sowohl der initialen Inbetriebnahme von Locust, unterstützt aber auch den fortlaufenden Betrieb.

Multiplikatoren – Multiplikatoren wägen genau den durch Locust erreichbaren Zusatznutzen gegen die möglichen Nachteile ab. Da sie vom reibungslosen Betrieb ihres Angebotes abhängig sind, werden sie Beeinträchtigungen des laufenden Betriebes iauch gegen mögliche monetäre oder anderweitige Vorteile nicht tolerieren. Interesse besteht demnach an einer gefahrenlosen Teilnahme, klaren Anweisungen und Beispielen sowie attraktivem Zusatznutzen, um die Anziehungskraft ihrer eigenen Anwendung für ihre Nutzer langfristig zu erhöhen. Die Benutzerfreundlichkeit des Locust Clients dient vor allem dem Erzeugen und dem Erhalt der Infrastruktur.

Web-Terminals *Web*-Terminals verlangen kurze Ladezeiten, möglichst geringen Ressourcenverbrauch und damit Beeinträchtigung ihrer *Online*-Sitzung. Weiterhin werden keine oder wenige *zwingende* Aktionen bei Installation oder Registrierung sowie Sicherheit gefordert. Monetärer oder anderweitiger Zusatznutzen oder eine verständliche Motivation des Problems durch anschauliche und verständliche Darstellung des Beitrags zur Gesamtleistung (Statusanzeige) sind dagegen von eher untergeordneter Bedeutung. Die Benutzerfreundlichkeit in dieser Benutzergruppe sichert für Locust hauptsächlich den fortlaufenden Betrieb durch immer wiederkehrende Leistungserbringer.

5.1.5 Geschwindigkeit

Wie das Akronym Locust (*LOw cost Computing Utilizing idle Time*) bereits andeutet, soll Locust in erster Linie Rechenleistung zu einem Bruchteil der sonst üblichen Kosten anbieten. Da Rechenleistung aber nicht allein preisbezogen nachgefragt wird, sondern ebenfalls qualitative und zeitliche Nebenbedingungen erfüllen muß, sollte eine Locust Implementierung diese Leistung auch in konkurrenzfähiger Zeit bereitstellen können. Das bedeutet, die Berechnung darf länger als in vergleichbaren Hochleistungssystemen brauchen, aber nicht so lange, daß der Preisvorteil gegenüber anderen verteilten und parallelen Umgebungen zunichte gemacht wird.

5.2 Architektur

Dieser Abschnitt behandelt die Architektur des Locust Modells zur Umsetzung der erarbeiteten Entwurfsziele. Die Locust Architektur besteht serverseitig aus dem Locust-Server und dem Locust-Markt und clientseitig aus dem Locust-Client.

1. Locust-Server – Der Locust-Server stellt die Basisfunktionalität zur Verteilung und Verwaltung von Java-Applets zur Verfügung, die opportunistische Ressourcen für parallele und verteilte Tasks abschöpfen

2. Der Locust-Markt – Der Locust-Markt implementiert einen elektronischen Markt, der Angebot und Nachfrage nach dedizierten Ressourcen zusammenführt

3. Der Locust-Client – Der Locust-Client besteht aus einem als *Job* bezeichnetem Komponenten-Paar, das von *Manager* und *Worker* gebildet wird.

   (a) Worker – Da das Locust Applet (Locus) nur für die Initialisierung zuständig ist, muß der Großteil der Funktionalität des Clients demnach von nachgeladenen, anwendungsspezifischen Client-Klassen, dem *Worker*, erbracht werden. Der *Worker* implementiert und übernimmt den rechenintensiven Teil eines bestimmten *Jobs*.

   (b) Manager –Der *Manager* ergänzt serverseitig der Locust-Server um die zur Verwaltung eines bestimmten *Jobs* notwendige Funktionalität. Er verwaltet hierzu die einzelnen Teilaufgaben, visualisiert den Fortschritt der Berechnungen, bietet eine GUI zur Konfiguration und gibt bei Bedarf Meldungen zur Fehlersuche aus.

5.2.1 Der Locust-Server

Um den geforderten Multiplikationsfaktor unter gleichzeitiger hoher Skalierbarkeit zu erreichen, ist besondere Weitsicht beim Entwurf notwendig. Ein hoher Multiplikationsfaktor ist durch einen geringen Code-Umfang, der sehr schnell kopiert und effizient verteilt werden kann, erreichbar. Die geforderte Skalierbarkeit auch bei sehr hoher Nebenläufigkeit muß durch eine redundante, ausfallsichere hierarchische Struktur der beteiligten Server-Dienste und ein extrem schlankes Verbindungsprotokoll unterstützt werden. Die redundante Auslegung der Server-Dienste erfordert darüberhinaus effiziente Lastverteilung.

Modulkonzept Infrastruktur und Aufgabe sind konzeptuell voneinander getrennt. Auf Server- und Client-Seite existiert jeweils ein Basis-Modul (Server- und Client-*Stub*) und ein oder mehrere aufgabenspezifische Module. Die server- und clientseitigen *Stubs*

- übernehmen die Kommunikation zwischen Client und Server,

- bieten eine einfache graphische Benutzeroberfläche und

- verwalten aufgabenspezifische Module.

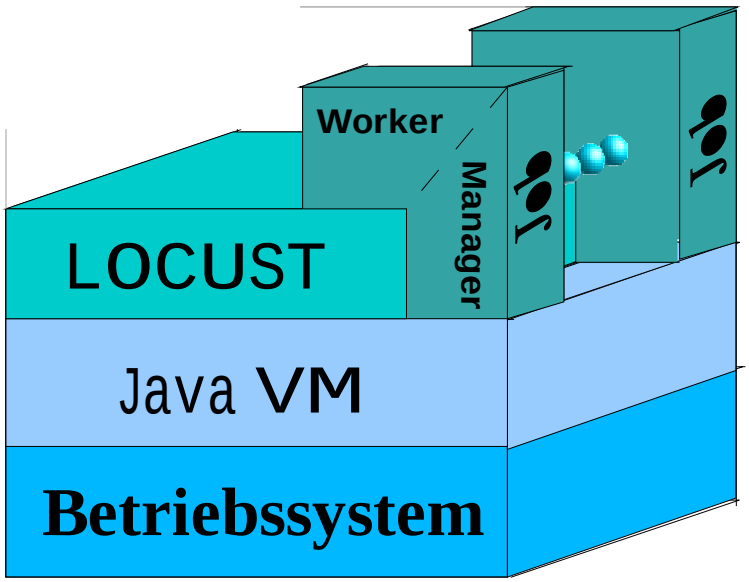

Abbildung 5.1 Das Locust Modulkonzept

Die aufgabenspezifischen Module besitzen ebenfalls eine graphische Benutzeroberfläche, die in die Oberfläche des Locust-Moduls eingebettet wird. Sie bestehen wie in Abbildung 5.1 illustriert aus einem serverseitigen Anteil, dem Verwalter (*Manager*), und einem clientseitigen Teil, dem Arbeiter (*Worker*), der den rechenintensiven Anteil der Berechnungen übernimmt. Die aufgabenspezifischen Komponentenpaare werden im folgenden auch als *Job* bezeichnet. Der *Manager* setzt direkt auf der Java *Virtual Machine* auf und ergänzt den Locust Server um die zur Verwaltung des jeweiligen Jobs notwendige Funktionalität. Hierzu zählen administrative Funktionen sowie Parallelisierungs- und Schedulingstrategie. Die Manager entscheiden, welche Teilaufgabe jeder Client bearbeiten soll und verarbeiten die zurückgelieferten Ergebnisse.

Der *Worker* setzt dagegen sowohl auf seinem server-seitigen Gegenstück als auch auf dem Locust Server auf (vgl. Abbildung 5.1 [Los99]) und ergänzt den Locust Client um die zur Berechnung des jeweiligen Jobs notwendigen Funktionalität.

Multiplikation Aufgrund der Leistungsfähigkeit, Sicherheit und weiten Verbreitung steht Java bereits als Implementierungssprache fest. Die Notwendigkeit geringen Code-Umfangs und der automatischen Code-Ausführung in wenig intelligenten *Web*-Terminals macht darüberhinaus das Java *Applet* Programmiermodell zur geeignetesten Wahl. Es muß nur dafür gesorgt werden, daß der rechenintensive Teil grobgranularer Anwendungen von einem zentralen Ort auf *Worker Applets* verteilt, von *Web*-Terminals ausgeführt und die Ergebnisse an diesem zentralen Ort wieder entgegengenommen und zusammengefügt werden. Als dieser zentrale Ort bietet sich der (oder die) Web-Server an, der die Web-Seite und das darin enthaltene Applet bereitstellt.

Die *Worker Applets* werden in Inhalte und Dienstleistungen der Multiplikatoren mit hohen Zugriffszahlen eingebettet. Auf diese Weise erfolgt die Vervielfältigung (*Cloning*) und der Versand der *Worker* durch den oder die Web-Server, von dem die Applets angefordert werden.

Da HTTP ein statusloses Protokoll ist, sind Web-Server in der Regel nicht in der Lage, mit Client-Browsern zu kommunizieren. Es wird daher ein weiterer Server-Dienst zur Kommunikation mit den *Workern* benötigt, der aufgrund der Applet Sicherheitsrestriktionen ebenfalls auf demselben physikalischen Rechner zur Verfügung stehen muß. Browser besitzen eine Reihe von strengen Sicherheitsanforderungen an potentiell unsicheren, aus dem Internet geladenen Programm-Code. Unter anderem dürfen unbekannte Applets keine anderen Netzwerkverbindungen als zu ihrer ursprünglichen Herkunft aufbauen. Obwohl diese Sicherheitsmaßnahme durch Signieren des Applets aufgehoben werden kann – so daß Verbindungen zu beliebigen Orten aufgebaut werden können –, erfordert dies eine Interaktion des *Web*-Terminals[4], ein Umstand, der nicht den Entwurfsprinzipien entspricht.

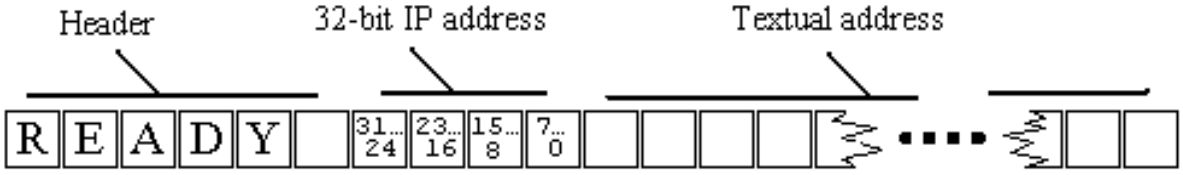

Abbildung 5.2 Locust Kommunikationssprotokoll

Skalierbarkeit Die geforderte Skalierbarkeit, auch bei sehr hoher Nebenläufigkeit, muß durch eine redundante, ausfallsichere hierarchische Struktur der beteiligten Server-Dienste und ein extrem schlankes Verbindungsprotokoll mittels UDP unterstützt werden. Die Initialisierung des *Workers* erfordert mehrere Nachrichten zwischen *Client* und *Server*, die mittels UDP abgewickelt werden (vgl. Abbildung 5.3). Im Gegensatz zu verbindungsorientierter TCP Kommunikation benötigt der Empfänger bei der UDP Kommunikation die Angabe einer Absenderadresse im UDP Paket. Deshalb gibt der Worker beim Anfordern des initialen Jobs dem Paket seine Adresse mit, damit ihn die Antwort auch erreicht. Um zeitintensive DNS Anfragen zu verhindern, sendet das Applet dabei sowohl seine IP Adresse als auch seinen vollständige Domain-Namen (*Fully Qualified Domain Name*). Auf diese Weise kann der Server schnelle, DNS-lose Verbindungen mit dem *Worker* aufbauen und zugleich aussagekräftige Domain-Namen in seinen Aufzeichnungen verwenden. Die anwendungsunabhängige Struktur einer Job Anforderung ist Inhalt von Abbildung 5.2 [Van97]. Um die durch im Zusammenhang mit der Job Anforderung erzeugte Last zu verteilen, werden mehrere Kanäle (*Ports*) für die Kommunikation verwendet.

Scheduling Als Schedulingstrategie wurde beim Locust Server Round-Robin gewählt, die relativ effizient und fair ist. Da bei den meisten Jobs, für die sich Locust eignet, die wahrscheinlich benötigte Laufzeit berechnet werden kann, ist alternativ auch eine LREP (*Least-remaining(-expected)-processing-time*) Strategie denkbar.

[4]Es wird eine Bestätigung verlangt, ob der signierte Applet-Code auszuführen ist

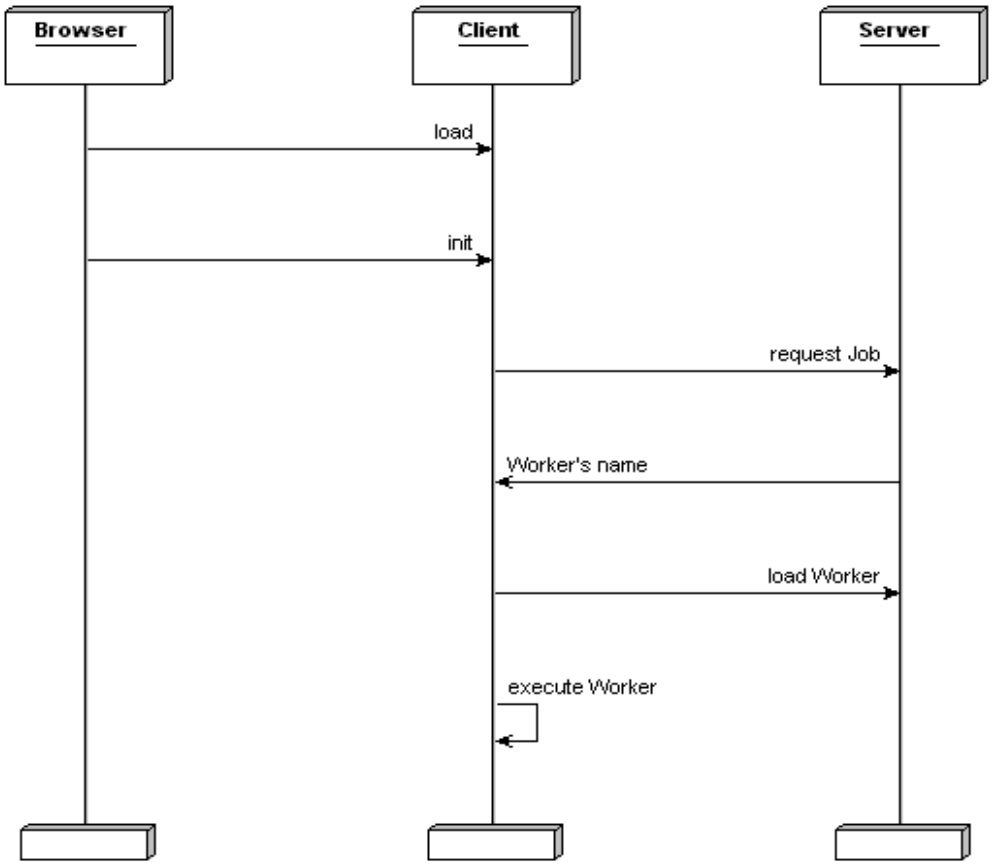

Abbildung 5.3 Initialisierung des Locust *Clients*

Lastverteilung Ist der Server der Engpaß bei der Leistungssteigerung des Gesamtsystems, sollte es möglich sein, mehrere Server parallel einzusetzen. Mehrere Server bringen aber nur dann mehr Leistung, wenn eine Lastverteilung durchgeführt wird. Derzeit ist die Leistungsfähigkeit eines oder einiger weniger Server ausreichend, so daß Lastverteilung, wenn notwendig, durch den Administrator durchgeführt werden kann.

5.2.2 Der Locust-Client

Der auf dem physikalischen Web-Server laufende *Locust-Server* sorgt für die initiale Verteilung der Jobs und abschließende Sammlung der Ergebnisse, so daß sich die Initialisierung des Systems wie aus Abbildung 5.3 ergibt:

1. Das *Web*-Terminal gelangt bei seiner Sitzung auf eine Web-Seite mit einem *Worker* Applet und fordert den Applet-Code vom Web-Server an (`load`)

2. Der Browser initialisiert das Applet (`init`)

3. Um den Code-Umfang zu reduzieren, enthält das Applet außer Basis-Funktionalität keinen anwendungsspezifischen Code. Daher fordert es zunächst beim *Locust-Server* einen *Job* an. (`requestJob`)

4. Der *Locust-Server* beantwortet die Anfrage mit einer Job-Spezifikation, die aus einer Job-Klasse besteht (`WorkerName`)

5. Das Applet lädt die Job-Klasse (`loadWorker`)

6. Nach Laden der Job-Klasse kann der *Worker* mit der Bearbeitung der ihm zugewiesenen Teilaufgabe beginnen (`executeWorker`). Nach Beendigung der Berechnung schickt das Applet das oder die Ergebnisse an den *Locust-Server* zurück.

Threads Ein vielfädiger Client ist in der Lage, mehrere Prozessoren auszunutzen. Dies hängt von der Hardware und dem Betriebssystem des Leistungserbringers ab. Eine nützliche Eigenschaft eines vielfädigen Clients wären *Just-in-Time-Job-Requests*, das heißt während der Worker noch den Job bearbeitet, holt der Client-Thread bereits den nächsten Job, so daß bei Beendigung des aktuellen Jobs der nächste schon bereit steht und so (fast) ohne Pause weitergerechnet werden kann.

Implementiert wurde ein Client jedoch nur mit einem einzigen Steuerfluß, da Multiprozessor-Rechner noch die Ausnahme sind und ein mehrfädiger Entwurf mehr Ressourcen benötigt und größere Klassen zur Folge hat. Dies führt zu längeren Ladezeiten und widerspricht somit der Benutzerfreundlichkeit.

Priorität des Locust-Clients Eine hohe Priorität erhöht die Rechenleistung, läßt aber anderen Prozessen weniger Rechenzeit und beeinträchtigt dadurch die übrige Leistungsfähigkeit des *Web*-Terminals. Empfindet der Terminal die Auswirkung des Clients als zu stark, führt dies im schlimmsten Fall dazu, daß er nicht wieder am Locust System teilnimmt. Aus Sicht der Benutzerfreundlichkeit wäre also eine niedrige Priorität wünschenswert. Tests haben allerdings ergeben, daß die Leistung des *Locust-Client* erheblich absinkt, wenn er unter einer niedrigeren Priorität ausgeführt wird. Aus diesem Grund übernimmt er die vordefinierte Priorität der Java *Virtual Machine*.

Online-Hilfe und statische Informationen Zur Benutzerfreundlichkeit gehören Hilfetexte und andere allgemeine, statische Informationen, die dem Leistungserbringer die Aufgabe erklären und näherbringen soll.

Der einfachste Weg, diese Information bereitzustellen, ist die direkte Einbettung in den Client. Dadurch ist der Zugriff schnell und muß nur über Java-Klassen verwaltet werden.
Diese Informationen können aber sehr umfangreich und aus verschiedenartigen Dokumenten (Texte, Videos, Animationen etc.) bestehen und sollten daher besser auf einer Web-Seite zur Verfügung gestellt werden, deren URL dem Client mitgegeben wird.

5.2.3 Der Locust-Markt

Während der Locust-Client und -Server in erster Linie zur Sammlung und Nutzung opportuner Ressourcen entwickelt wurde, ist darüberhinaus auch die Sammlung von dedizierten Ressourcen vorgesehen. Hierzu dient der sogenannte Locust-Markt, der einen Gütermarkt für Rechenressourcen implementiert. Hierzu werden zwei Teilnehmer am Markt benötigt,

der Anbieter, der ungenutzte Rechenleistung übrig hat, und der Nachfrager, der zusätzliche Rechenressourcen mieten möchte. Diese Parteien treten über den Markt in Kontakt und einigen sich über die Details einer Reservierung.

Teilmärkte Das Locust Marktsystem basiert auf Teilmärkten. Ein Teilmarkt ist ein einzelner elektronischer Markt mit Anbietern und Nachfragern zusammen mit den ihn implementierenden und unterstützenden Server-Diensten. Ein Teilmarkt kann das Intranet einer Firma, eine bestimmte Region oder sogar eine spezielle Anwendung umfassen. Das Hauptmerkmal von Teilmärkten ist ihre Homogenität bezüglich einer Eigenschaft. Sobald ein Teilmarkt die Grenzen seiner Belastbarkeit und Skalierbarkeit erreicht hat, kann er ganz leicht bezüglich einer neuen Eigenschaft in zwei Teilmärkte mit eigenen Server-Diensten zerlegt werden und so die Last gleichmäßig verteilen. Teilmärkte (bzw. ihre Server-Dienste) stehen in Kontakt und sind so in der Lage, Überschüsse und Defizite von Rechenressourcen durch Export und Import auszugleichen. Wenn ein Teilmarkt in Europa während des Tages ausgelastet ist, können zusätzliche Ressourcen aus amerikanischen Teilmärkten importiert werden, wo ein Großteil der Nutzer gerade schläft.

Jeder Teilmarkt wird durch vier Server-Dienste implementiert. Der erste ist der *Accountant*, der Informationen über Nutzer und Maschinen enthält und bereitstellt. Der *Event* Server kann kontaktiert werden, um regelmäßig Informationen über bestimmte Ereignisse des Teilmarktes zu beziehen. Über den *Update* Server wird der von den gemieteten Maschinen auszuführende Code verteilt. Schließlich verwaltet der *Market* Server alle aktuellen Miet- und Vermietungsgebote.

Die Processor-Klasse Ein *Processor* stellt die Grundeinheit der im Locust Markt handelbaren Rechenressourcen dar. Derzeit ist ein *Processor* gleichbedeutend mit einer einzelnen Maschine, obwohl die Unterstützung mehrerer *Processors* einer Mehrprozessor-Maschine zwar aufwendig, aber theoretisch möglich ist. Ein *Processor* wird mittels einer Java Anwendung implementiert, die die Ressourcen eines physikalischen Rechners im Auftrag des Anbieteres verwaltet. Diese Software ist der Dreh- und Angelpunkt des Systems, da sie die notwendigen Voraussetzungen für Vertrauen der Anbieter und Nachfrager in den Ressourcen-Handel schafft. Im Wesentlichen verwendet die *Processor* Applikation das Java *Sandbox* Modell, um mißbräuchliche Nutzung gemieteter Ressourcen zu verhindern.

Marktteilnehmer Teilnehmer mit ungenutzter Rechenzeit (sogenannte Anbieter) registrieren ihre Rechner mit dem *Accountant*. Die Registrierung beinhaltet Daten über Maschinen-Name, -Typ und IP-Adresse sowie Informationen über die Leistungsfähigkeit der Maschine wie CPU, Taktfrequenz und verfügbaren Speicher. Wenn der Teilnehmer seinen Rechner zeitweise nicht benötigt, wird der *Processor* (meistens über einen Bildschirmschoner) gestartet. Der *Processor* tritt in Kontakt mit dem Locust Markt und gibt ein Gebot ab, das die jeweilige Maschine zur Miete anbietet. In diesem Verkaufsangebot sind die notwendigen Bedingungen wie Preis, Verfügbarkeit usw. spezifiziert.

Teilnehmer, die zusätzliche Rechenressourcen benötigen (sogenannte Nachfrager), registrieren sich ebenfalls beim *Accountant*. Wenn zusätzlicher Bedarf an Ressourcen tatsächlich akut wird, treten sie mit Hilfe einer Nachfrage an den Locust-Markt heran, in dem sie die Anzahl und den Typ der gewünschten Maschinen spezifizieren. Diese

Anfrage führt zu einer Liste von passenden Ressourcenangeboten. Der Nachfrager setzt daraufhin einen Reservierungsantrag ab. Nach Akzeptierung dieses Antrags stehen dem Nachfrager die reservierten Maschinen bis zum Auslaufen des Reservierungszeitraums zur Verfügung.

Anbieter Um einen Rechner dem Markt zur Verfügung zu stellen, startet der Anbieter die *Processor* Anwendung, entweder explizit oder über einen Bildschirmschoner, der nach einer gewissen Zeit der Inaktivität anspringt. Der *Processor* kontaktiert daraufhin den Markt und veröffentlicht ein Angebot mit den Nebenbedingungen zur Reservierung der jeweiligen Maschine. Die Hauptbedingung ist der "Maschinenstunden"-Preis, der die Kosten zur Reservierung der Maschine für eine Stunde angibt.

Nachfrager Wenn ein Nachfrager zusätzliche Ressourcen benötigt, gibt er ein Mietgesuch mit den erforderlichen Nebenbedingungen ab. Eine mögliche Nebenbedingung ist beispielsweise, daß der Nachfrager temporären Speicherplatz auf dem Dateisystem der reservierten Maschine braucht. Diese Bedingung wird durch Aufruf von `BuyOffer.setDiskAccess(true)` gesetzt.

Mit dem Mietgesuch kontaktiert der Nachfrager den Markt, um die Parteien zu ermitteln, die an einer Vermietung ihrer Maschinen interessiert sind. Dies geschieht mittels des Aufrufs `Market.query(BuyOffer)`, der eine Liste alle passenden `BuyOffers` zurückliefert. Der Nachfrager kann daraufhin die *Processors* kontaktieren, um die Reservierungen vorzunehmen. Diese Vereinfachung sorgt für eine bessere Skalierbarkeit des Marktsystems: Der Markt ist nur für die Zusammenführung von Anbieter und Nachfrager verantwortlich, nicht für die tatsächlichen Verhandlungen.

Reservierung

Allgemeines Eine Reservierung ist als Interaktion zwischen genau zwei Parteien, dem Anbieter und dem Nachfrager, implementiert. Der Reservierungsprozeß von mehr als zwei *Processors* wird als Iteration über den jeweiligen Maschinen durchgeführt. Die Nachfrager Schnittstelle ist in der Regel ein `Broker`-Objekt, kann aber auch anderer Java Code sein. Die Anbieter Schnittstelle ist ein *Processor*.

Die Reservierung wird normalerweise mittels einer Referenz auf einen *Processor* eingeleitet, die über den Markt erlangt wurde. Danach formuliert der Nachfrager seine Parameter und Nebenbedingungen für einen Ressourcen-Kauf in einem `BuyOffer` Objekt.

Reservierungen Wenn der Nachfrager alle Nebenbedingungen festgelegt hat, erzeugt er ein `ReservationRequest` Objekt, das das `BuyOffer` Objekt sowie den Nachfrager und Anbieter enthält. Diese Reservierung wird dem Anbieter mittels `Processor.makeReservation` zugestellt. Für den weiteren Verlauf der Reservierung bestehen mehrere Möglichkeiten:

1. Der *Processor* ist bereits reserviert – Der *Processor* erzeugt eine `ProcessorNotAvailableException` Ausnahme. In zukünftigen Versionen kann diese Ausnahme Hinweise darauf enthalten, wann die Maschine wieder verfügbar ist.

2. Der *Processor* ist frei und akzeptiert die Reservierung – Falls der Anbieter die Reservierungsbedingungen akzeptiert, erzeugt er ein `Reservation` Objekt, das die Reservierungsanforderung enthält. Das `Reservation` Objekt wird daraufhin in einer `ReservationResponse` zurückgeliefert.

3. Der *Processor* ist frei, akzeptiert die Reservierungsbedingungen aber nicht – Der *Processor* kann mit einem Gegenangebot in Verhandlung mit dem Nachfrager treten. Dazu erzeugt er ein `SellOffer` Objekt, das seine Reservierungspräferenzen enthält. Dieses Objekt wird in einer `ReservationResponse` zurückgeliefert.

4. Der *Processor* lehnt die Reservierung ab – Der Anbieter liefert eine `ReservationResponse` mit dem Grund der Ablehnung zurück. Der Nachfrager kann sein Reservierungsangebot daraufhin überdenken und nochmal stellen.

5. Der *Processor* lehnt die Reservierung kategorisch ab – Erneute, modifizierte Reservierungangebote des Nachfragers sind sinnlos, da der Anbieter kategorische Einwände gegen den Nachfrager besitzt. Die Grund hierfür – zum Beispiel, daß der Nachfrager als unzuverlässig bekannt ist – kann mittels der Methode `ReservationResponse.getReason()` erfragt werden. Die Aufzeichnung vergangener `ReservationEvents` ermöglicht eine eingeschränkte Bewertung der *Zuverlässigkeit* von einzelnen Marktteilnehmern bzw. ihren Maschinen. Obgleich mit dieser Information kein Qualitätsgarantien (*Quality of Service*) getroffen werden können, ermöglicht sie zumindest die Abschätzung der zukünftigen Zuverlässigkeit.

Reservierungsende Die Reservierung kann auf folgende Arten beendet werden:

1. Die Reservierung läuft aus – Ohne weitere Interaktion zwischen Anbieter und Nachfrager läuft die Reservierung nach dem im `Reservation` Objekt festgelegten Zeitraum aus. Danach zerstört der *Processor* alle verwendeten Objekte und schickt ein `ReservationEvent` Ereignis an alle abonnierten Zuhörer.

2. Die Reservierung wird storniert – Falls der Anbieter seine Reservierungsabsicht ändert, kann er die Reservierung zurücknehmen. Der *Processor* "räumt" anschließend auf und versendet den Abbruch als `ReservationEvent`.

3. Die Reservierung wird abgebrochen – Bei einem Abbruch handelt es sich um eine plötzliche Stornierung, die keine Zeit mehr für ein geordnetes Aufräumen und das Versenden eines `ReservationEvent` mehr zuläßt.

4. Die Reservierung wird verlängert – Der Nachfrager benötigt zusätzliche Ressourcen und beantragt eine Reservierungsverlängerung. Dieser Fall wird im folgenden Abschnitt behandelt.

Verlängerung Falls der Nachfrager weitere Rechenzeit benötigt, kann er nicht einfach ein weiteres Reservierungsangebot absetzen, da die jeweilige Maschine bereits reserviert ist. Aus diesem Grund muß der Nachfrager eine Verlängerung mittels `Processor.extendReservation()` beantragen. Die Semantik dieser Methode entspricht

der oben angegebenen von `Processor.makeReservation()` außer, daß Fall 1 nicht vorkommen kann. Stattdessen antwortet der *Processor* mit einem `ACCEPTED, REJECTED` oder `REJECTED_CATEGORICALLY ReservationEvent` Ereignis.

Händler

Händler erlauben es, die beschriebenen Interaktionsschritte mit einem minimalen Aufwand für den Benutzer durchzuführen. Der Händler nimmt hierzu eine abstrakte Beschreibung der Wünsche und Nebenbedingungen des Benutzers entgegen und wickelt anhand dieser Informationen die Verhandlungen und Aktionen selbsttätig ab. Beispielsweise nimmt der *Broker* Parameter wie die minimale und maximale Anzahl und die maximalen Stundenkosten von Maschinen entgegen, und liefert anhand dieser Daten eine Liste zu diesen Konditionen verfügbarer *Processors* zurück. In zukünftigen Versionen sind weitere Händler denkbar, die Anbietern und Nachfragern weitergehende Abstraktion und Komfort bieten können.

5.3 Implementierung

5.3.1 Der Locust-Server

Die Locust Komponenten wurden in Java 1.1 implementiert. Zur Erstellung der grafischen Benutzeroberfläche wurde NetBeans DeveloperX2 2.1 verwendet [Los99]. Der Schritt von den Entwurfsvorgaben zu den erforderlichen Java Klassen war sehr intuitiv und unmittelbar durchführbar. Das Locust-Modul besteht serverseitig aus der Klasse `LocustServer` und clientseitig aus der Klasse `LocustClient`. Zur Implementierung eines Job existiert die Klasse `Job`. Sie besteht aus einer `Manager`-Klasse und einer `Worker`-Klasse.

Datenstruktur zur Verwaltung von Jobs

Die Datenstruktur ist eng mit der Schedulingstrategie verknüpft, da diese Strategie möglichst effizient sein soll. Drei mögliche Strukturen wurden in Betracht gezogen:

- Warteschlangen – Warteschlangen sind sehr flexibel, aber werden mit hohem Verwaltungsaufwand erkauft.
- Hash-Tabellen – Hash-Tabellen bieten hohe Effizienz und geringen Verwaltungsaufwand, sind jedoch relativ unflexibel.
- Warteschlangen in Kombination mit Hash-Tabellen – Kombiniert bieten beide Datenstrukturen hohe Flexibilität und Effizienz bei hohem Verwaltungsaufwand.

Wegen oben genannter positiver Eigenschaften wurden Hash-Tabellen zur Verwaltung von Jobs gewählt.

Kommunikation

Auf Serverseite müssen die von den *Workern* erzeugten Ergebnisse wieder entgegengenommen und zusammengefügt werden. Auch hierfür werden mehrere Kanäle verwendet. Jeder

```
public void playLocustServer () {
  while (true)
      processJobRequests();
}
```

Abbildung 5.4 Hauptsteuerfluß des Locust-Servers

Kanal wird von einem eigenen *Thread* bedient, da die Entgegennahme einer blockierenden Netzwerkverbindung den Steuerfluß bindet (`bind()`) und das Horchen (`listen()`) auf weiteren einkommenden Netzwerkverkehr blockiert. Außerdem kostet die Entgegennahme und Verarbeitung der Ergebnisse relativ viel Rechenleistung. Würden all diese Aufgaben von einem einzigen Thread übernommen, würde er mit Ergebnispaketen überlaufen und es käme zu Paketverlusten aufgrund der Überlastung des Servers.

Die interne Struktur des Locust-Servers reflektiert diese Nebenläufigkeit bei der Behandlung der eintreffenden Ergebnisse. Der Server besteht aus einem Hauptsteuerfluß (*Thread*) und einer konfigurierbaren Anzahl von Nebensteuerflüssen. Der Haupt-*Thread* horcht in einer Endlosschleife am Eingangsport und beantwortet hereinkommende Anfragen nach Jobs mit einer Job-Spezifikation (mittels `processJobRequest()`, vgl. Abbildung 5.4).

Genauso wie der Haupt-Thread sind die Nebensteuerflüsse jeweils für einen eigenen Eingangsport verantwortlich. Beide Parameter, Steuerfluß und Portnummer, sind als Instanz der Klasse `ResultsReceiver` gekapselt. Jede Instanz dieser Klasse horcht ebenfalls in einer Endlosschleife, nimmt die Client-Ergebnisse an diesen Port entgegen und reicht sie an den zugehörigen Manager weiter(Vgl. Abbildung 5.5).

Die Teilergebnisse müssen wegen ihrer Anwendungsabhängigkeit im jeweiligen Manager mit Hilfe der `collateResults()` Funktion in einer zentralen Datenstruktur zusammengefügt werden, die im einfachsten Fall aus einem Feld boolscher Werte besteht. Die Werte repräsentieren, ob ein bestimmter Teil-Job bereits erledigt wurde oder nicht. Trifft ein Teilergebnis ein, wird zunächst geprüft, ob dieses wirklich noch ausständig ist, und falls ja, wird das Ergebnis aufgenommen und als erledigt markiert, sonst verworfen.

Um diese zentrale Datenstruktur auch bei nebenläufigen und potentiell gleichzeitigem Zugriff konsistent zu halten, wird das `synchronized` Schlüsselwort zur Synchronisierung der betreffenden Strukturen verwendet. Gewöhnlicherweise wird `synchronized` zur Synchronisierung vollständiger Funktionen verwendet, zur Vermeidung unnötiger Blockierung wurden jedoch nur die betroffenen Code-Teile synchronisiert.

Der Aufbau der den *Workern* zurückgeschickten Job Spezifikation ist bis auf Ausnahme der ersten zwei Bytes anwendungsabhängig. Diese Bytes enthalten die Port Nummer, die das Applet für Antworten zu verwenden hat. Das Adressfeld wird vom Server *Stub* des Clients entfernt bzw. hinzugefügt, bevor oder nachdem es vom anwendungsspezifischen Teil des Clients bearbeitet wurde. Genauso wird dieser Port auf Serverseite von der Job Spezifikation des anwendungsabhängigen Managers entfernt oder an diesen "angeheftet". Dadurch schirmen die Basis-Module die technischen Details der Kommunikation vor den Job Klassen ab.

```
while(true) {
   DatagramPacket dg = new DatagramPacket(results, JOB_RESULTS_PACKET_LENGTH);
   try {
     dgSocket.receive(dg);
       }
   .
   results = dg.getData();
   .
   //extract the workerClassName
   String workerClassName = new String(results, JOBID_FIELD_POS, JOBID_FIELD_LENGTH);
   .
   //extract requestnextjob field
   if(results[REQUESTNEXTJOB_FIELD_POS] == 1) requestNextJob = true;
   else requestNextJob = false;

   //extract job specific results
   for (int j=0, i=RESULTS_FIELD_POS; i < RESULTS_FIELD_POS+RESULTS_FIELD_LENGTH; )
     resultDetails[j++] = results[i++];

   //lookup job by the workerclassname
   Manager m = locustServer.getManager(workerClassName);
   m.collateResults( resultDetails );
   .
   .
   .
}
```

Abbildung 5.5 Hauptschleife der `ResultsReceiver` Prozedur

Persistenz

Bei der Klasse `LocustServer` wurde auf Persistenz verzichtet, da noch kein Accounting-Modul existiert und alle Parameter auf der Kommandozeile übergebbar sind. Bei der Klasse `RC5KeyManager` ist jedoch Persistenz notwendig, da sonst die getesteten Schlüsselblöcke verloren gehen, falss der Server (vorgesehen oder unvorgesehen) beendet wird. Zwei mehr oder weniger komplexe Alternativen sind denkbar:

- Dateisicherung – Eine Datei, in der alle Attribute des `KeyManager`s gespeichert werden.

- Serialisierung – Die Speicherung des gesamten `RC5KeyManager`-Objekts.

Letzte Alternative ist selbstverständlich die wünschenswertere, denn Serialisierung ist bereits Bestandteil von Java 1.1. und es fallen die Details des Ladens und Speicherns weg. Da aber eine Klasse sich nicht selber laden kann und da der `LocustServer` die konkrete Manager-Klasse nicht kennt (nur über das Manager Interface), blieb nur die erste Alternative übrig. Die Attribute des `RC5KeyManager`s werden vor Beenden des Managers

und bei jedem Übergang zu einem neuen Super-Block automatisch gespeichert. Im Falle des Wiedereinlesens wird das gesicherte Objekt wieder rekonstruiert.

5.3.2 Der Locust-Client

Da das Locust Applet (Locus) nur für die Initialisierung zuständig ist, der Großteil der Funktionalität des Clients demnach von den nachgeladenen, anwendungsspezifischen Client- und Server-Klassen erbracht wird, findet sich die Beschreibung der Clients in Kapitel 6.

5.3.3 Der Locust-Markt

Der Locust-Markt (LM) ist die Software Komponente, die den elektronischen Markt und Marktplatz des Locust Modells implementiert. Während das Locust Modell keine spezifischen Angaben über die Art der zu handelnden Güter und Dienstleistungen macht sondern allgemeine Rechenressourcen modelliert, unterstützt der Locust-Markt nur den Handel und die Nutzung von Prozessorzeit. Mehrere Locust-Märkte und die von ihnen realisierten (Teil-)Märkte lassen sich hierarchisch koppeln und ermöglichen damit den Import und Export von Rechenressourcen zwischen Teilmärkten.

Gebote

Es existieren zwei Arten von Geboten, Miet- und Vermietgebote (`BuyOffers` und `SellOffers`), die beide von einer allgemeinen `Offer` Klasse abgeleitet sind. Diese Klasse besitzt folgende Eigenschaften:

- `Offerer` – Der für das Gebot verantwortliche Benutzer.
- `OfferCreation/OfferExpiration` – Das Erzeugungsdatum und das Ablaufdatum des Gebots.
- `CostPerMachine` – Der gebotene Preis für eine einstündige Reservierung.
- `MinDuration/MaxDuration` – Der für den Bieter akzeptable Bereich von Reservierungen.

In der LM Marktimplementierung wird derzeit nur die Veröffentlichung von (Vermiet-) Angeboten unterstützt. Die Veröffentlichung von Mietgeboten machen insofern nur wenig Sinn, da Nachfrager im allgemeinen nur bei Bedarf an den Markt herantreten und direkt mit den Anbietern verhandeln anstatt ein Kaufgebot abzugeben und auf Antworten zu warten.

Teilmärkte

Jeder Teilmarkt wird durch vier Server-Dienste implementiert.

Accountant-Server – Der *Accountant* ist ein Serverdienst, der Informationen über Nutzer und Maschinen aufzeichnet und bereitstellt.

Market-Server – Der Markt besteht im wesentlich aus einer Liste von Angeboten, die über den *Market*-Server mittels einer Nachfrage abgefragt werden kann. Ergebnis einer solchen Anfrage ist eine Liste von passenden Angeboten mit Referenzen auf die entsprechenden Maschinen bzw. *Processors*. *Processors* werden jedoch nicht vom *Market* Server reserviert, sondern der Nachfrager ist für die Reservierung der jeweiligen Maschinen verantwortlich. Hierdurch skaliert das Marktsystem besser.

Event-Server – Wenn die Teilnehmer von Teilmärkten Ereignisse auslösen, kontaktieren sie den *Event*-Server, um ihn über diese Ereignisse auf dem Laufenden zu halten. Der Server verwaltet eine Liste von Marktteilnehmern, die an verschiedenen Ereignissen interessiert sind, und stellt den betroffenen Parteien die jeweiligen Ereignisse zu.

Update-Server – Der *Update*-Server enthält und verwaltet den Programmcode, den gemietete Maschinen zur Ausführung benötigen. Zum Update erstellt die gemietete Maschine zunächst einen Index der bereits vorhandenen Dateien, erzeugt auf diesem Index eine Prüfsumme (CRC) und sendet diese an den *Update*-Server. Basierend auf dieser Prüfsumme sendet der Server einen Satz von *add/update/delete*-Operationen zurück, die die Client Maschine mit dem *Update*-Server synchronisieren.

Locust-Market API

Der Entwurf, die Modularisierung und die Implementierung des Locust Markets folgt den bei Java Projekten üblichen Richtlinien: die individuellen Funktionalitäten sind in einzelne Pakete aufgebrochen, die sich unter dem Pfad `locust.market` befinden.

Dieser Abschnitt führt zunächst in den internen Aufbau der Locust-Market Pakete ein. Anschließend wird die Initialisierung, die Ereignisbehandlung und der Zugriff auf die Serverdienste des Marktes beschrieben. Abgeschlossen wird der Abschnitt mit einem Einblick in interne Details wie Sicherheitsaspekte, das Laden, die Zwischenspeicherung sowie die Versionsverwaltung von entfernten Java Klassen.

Pakete Die Locust-Market API besteht aus folgenden Paketen:

- `locust.market` – Haupt-Klassen und Schnittstellen
- `locust.market.event` – Verschiedene*Events* und Abonnementen als Java Beans
- `locust.market.processor` – Alle zur Vermietung eines Prozessors notwendigen Klassen
- `locust.market.util` – Verschiedene Hilfs- und Dienst-Klassen

Folgende interne Pakete sind nicht zur direkten Verwendung durch Entwickler gedacht:

- `locust.market.impl` – Implementierung verschiedener Schnittstellen
- `locust.market.loader` – Lokale und entfernte *classloader*
- `locust.market.maint` – Klassen für die Administrierung und Pflege der Server
- `locust.market.update` – *Update* Server und Client Klassen

```
public static void main( String args[] )
throws LMException
{
    LMSystem.init( args );
    Systemout.println( "LM Started Successfully!" );
    LMSystem.shutdown();
}
```

Abbildung 5.6 Beispiel einer typischen Initialisierung des Locust-Market

```
LMSystem.init( args );

Accountant a = LMSystem.getAccountant();
User somedude = a.getUser( "michael" );

System.out.println( "Found user: " + somedude );

LMSystem.shutdown();
```

Abbildung 5.7 Beispiel eines Zugriff auf eine Referenz des Locust-Markets

Initialisierung Die wichtigste Klasse zur Nutzung des Locust-Brokers ist die `locust.market.LMSystem` Klasse. Sie kontrolliert die Initialisierung, die Teilnahme, den Log-Out und die Beendigung des Locust-Brokers. Sie ist gleichzeitig der Zugang zu den verschiedenen Server-Diensten.

Vor jedem Aufruf von API Funktionen muß zunächst das System mittels der Methode `LMystem.init()` initialisiert werden. Die Parameter dieser Methode sind entweder ein Feld von Strings oder ein Applet, abhängig davon, ob sie von der Kommandozeile oder von einem Applet aus aufgerufen wurde. Zu den notwendigen Parametern gehören Authorisierungsinformationen wie Benutzername und Paßwort und der zu kontaktierende Teilmarkt. Die Teilmarkt-Klasse enthält alle systemweiten Namen. Es existieren vier Wege, solche Namen zu setzen:

1. Kommandozeilen-Parameter z.B. `lm -user michael -password passwort`
2. Java Systemeigenschaften z.B. `lm -DLM_USER=michael -DLM_PASSWORD=passwort`
3. Applet Parameter z.B. `<param name=LM_USER value=michael>`
4. Konfigurationsdatei z.B. `java -DLM_HOME=/usr/local/locust/`

Zugriff auf den Locust-Market Server Auf jeden der LM Server wird über eine Java Schnittstelle zugegriffen. Diese verbirgt die technischen Details der Suche, der Bindung und der Nutzung entfernter Server. Darüberhinaus enthält die `LMSystem` Klasse Methoden zur Generierung gültiger Referenzen auf solche Server.

Ereignisbehandlung Die Verwaltung und Weitergabe von Ereignissen durch den *Event* Server erlaubt ein Monitoring des Teilmarktes durch Marktteilnehmer. Die folgenden Ereignisse werden behandelt:

- `ComputationEvent` – Ereignisse, die bei der Miete und Nutzung eines Prozessors durch einen Nachfrager auftreten, wie Start, Fehlschlag, Abbruch und Ende einer Berechnung.
- `MachineEvent` – Statistikereignisse, die im Zusammenhang mit Aufnahme, Modifikation oder Löschung eines Datums im *Accountant* Server stehen.
- `OfferEvent`– Ereignisse, die im Zusammenhang mit der Aufnahme, Modifikation oder Löschung von Geboten entstehen.
- `ProcessorEvent` – Änderungen des Zustands von Prozessoren, zum Beispiel die Bereitstellung, Reservierung oder deren Aufhebung .
- `ReservationEvent` – Ereignisse, die im Zusammenhang mit der Änderung des Reservierungszustands eines Prozessors stehen, wie beispielsweise die Akzeptierung, Ablehnung, Aufhebung oder das Ende einer Reservierung.
- `StatusEvent` – Allgemeine Klasse mit einer Reihe generischer Ereignisse, um Zustände von LM Prozessen zu modellieren.
- `UserEvent` – Ereignisse, die im Zusammenhang mit dem *Accounting* Server entstehen, wie zum Beispiel die Einrichtung, Modifikation oder Löschung von Benutzern.

Processor Bevor ein *Processor* genutzt werden kann, muß zunächst eine Referenz auf einen *Processor* beschafft werden. Dies erfolgt immer über einen `ComputationBroker`. Diese Klasse verwaltet alle notwendigen Beziehungen zwischen Anbieter, Nachfrager und dem Markt und verbirgt Details hinter einer einfachen Schnittstelle, wie im folgenden Beispiel illustriert:

Sicherheit Die *Processor* Anwendung verwendet das Java *Sandbox* Modell, um entfernte, potentiell unsichere Programme an der Ausführung sicherheitskritischer Operationen zu hindern, wie zum Beispiel:

- Zugriff auf Benutzerdateien
- Zugriff auf den Bildschirm (lesend oder schreibend)
- Verlassen des *Processor*s
- Manipulation interner LM Threads

Diese Restriktionen werden durch den `ProcessorSecurityManager` implementiert, der die `java.lang.SecurityManager` Klasse erweitert. Methoden dieser Klasse werden *vor* allen potentiell unsicheren Operationen ausgeführt, die `SecurityException` Ausnahmen auslösen, falls eine unerlaubte Operation entdeckt wird. Die allgemeinste Methode zur

```
// reserve between 1 and 5 Processors for 1 minute (60,000 milliseconds)
BasicComputationBroker broker = new BasicComputationBroker( 1, 5, 60L * 1000L );

// go and reserve the Processors
Vector processors = broker.getProcessors();
for( int i=0; i<processors.size(); i++ )
    {
    // print out the IP address of each Processor's location
    Processor p = (Processor) processors.elementAt( i );
    System.out.println( "Reserved a Processor @ " + p.getPrintableAddress() );

    // create an object on the remote Processor
    p.createObject( "RemoteObj", "some.qualified.ClassName" );

    // invoke a method on the remote object, sending "foobar" as an argument
    p.invokeMethod( "RemoteObj", "methodName", new Object[] { "foobar" } );
    }

// release or "unreserve" each Processor
broker.releaseReservations();
```

Abbildung 5.8 Anwendung der `ComputationBroker` Klasse

Überwachung unsicheren Codes ist der Test, ob die anfordernde Klasse von einem entfernten *ClassLoader* geladen wurde. Die meisten unerlaubten Operationen, wie die Verwendung des AWT oder der Versuch, die Java VM zu verlassen, lassen diesen Test von entferntem Code nicht zu, so daß eine solche Fehlermeldung als Indikator verwendet werden kann.

Andere Operationen wie der Zugriff auf das lokale Dateisystem sollten weniger restriktiv gehandhabt werden. Zum Beispiel sollen über das Netzwerk geladene Klassen zusätzlich in einem lokalen Cache – also auf dem Dateisystem – abgelegt werden. Die wird dadurch behandelt, daß geprüft wird, ob sich der Locust-Market *ClassLoader* auf dem Keller vor fremden Objekten befindet. Falls dem so ist, weiß das System, daß der Befehl vom Locust-Market selbst stammt und zugelassen werden kann.

Threads Da es keine Möglichkeit gibt, Java Code von der Erzeugung eines neuen Steuerflusses und seiner Ausführung abzuhalten, sollte die Java VM zumindest den Ort neu erzeugter Threads bestimmen. Der `ProcessorSecurityManager` veranlaßt alle von fremden Objekten erzeugte Threads innerhalb einer bestimmten fremdenThread-Gruppe zu existieren. Dies ermöglicht, daß alle fremden Threads mittels einem einzigen `ThreadGroup.stop()` Aufruf gestoppt werden können.

Laden entfernter Klassen

Eine der leistungsfähigsten Eigenschaften des Locust-Market sowie von Java selbst ist die Fähigkeit, dynamisch fremden Code entfernter Maschinen laden zu können. Das einfachste

Beispiel für diese Eigenschaft ist ein innerhalb eines Browsers ablaufendes Java Applet. Das `<applet>`[5] HTML Tag veranlaßt den Browser, das referenzierte Applet sowie evtl. zusätzliche Klassen vom Web-Server zu laden und auszuführen. Das Laden von Klassen im LM ist aus nachfolgenden Gründen etwas komplexer.

Caching Im Gegensatz zu Applets, die im Allgemeinen schlank und schnell zu laden sind, können Anwendungen des Locust Markts durchaus sehr umfangreich und entsprechend langsam zu laden sein. Aus diesem Grund enthält der Locust-Markt ein Caching Verfahren, das Klassen und Archive von entfernten Maschinen auf dem lokalen Dateisystem ablegt, um zukünftige Anfragen nach diesen schneller beantworten zu können. Jedes lokale Cache-Element wird nach Ablauf des Reservierungszeitraums als *dirty* markiert, um ein Update eventuell veränderter Klassen zu ermöglichen.

Versionsverwaltung Das Caching von Klassen scheint auf den ersten Blick das Laden neuer Klassen-Versionen zu erschweren, da augenblickliche Java VMs (bis Version 1.2) keine Klassen "entladen" können, um eine neue Version zu laden. Über den eigenen Namensraum von *ClassLoadern* läßt sich dennoch ein Laden neuer Versionen einrichten. Zwei verschiedene *ClassLoader* A und B können also über zwei Klassen mit demselben Namen `locust.market.beispiel` verfügen. Die Haupt-Java-Klassen (`java.lang.*`) bleiben jedoch im gleichen Namensraum, da sie vom System *ClassLoader* geladen werden.

Wenn eine Maschine über den `Processor` reserviert wird, wird ein entfernter *ClassLoader* erzeugt, der auf die Maschine des Nachfragers zeigt. Dies bedeutet nichts anderes, als daß eine Klasse von der Nachfrager-Maschine zu laden ist, falls sie auf dem lokalen Dateisystem nicht gefunden wird. Wenn die Reservierung endet, wird der entfernte *ClassLoader* einfach entladen und durch einen neuen *ClassLoader* ersetzt. Der neue *ClassLoader* entlädt die alten Klassen nicht automatisch, sondern der *Garbage Collector* wird zu einem bestimmten Zeitpunkt die nicht mehr referenzierten Klassen aus dem Speicher entfernen, während der neue *ClassLoader* bereits neue Versionen dieser Klassen laden und ausführen kann.

[5] `bzw. <embed>`

Kapitel 6

Locust Anwendungen und Leistungsfähigkeit

Dieses Kapitel beschäftigt sich mit der Implementierung von Anwendungen für die Locust-Infrastruktur und versucht dabei eine Brücke zu schlagen von implementierten und im Detail evaluierten Anwendungen aus dem verteilten Hochleistungsrechnen, über einige ausgereifte Anwendungsstudien aus dem verteilten Hochdurchsatzrechnen ohne konkrete Realisierung, bis hin zu skizzenhaft präsentierten Anwendungsmöglichkeiten in zukünftigen Forschungsarbeiten.

Das Kapitel beginnt zunächst mit der Beschreibung der im Rahmen dieser Arbeit implementierten und evaluierten Locust-Anwendungen, der parallelen RC5 Dechiffrierung und der parallelen Raytracing Anwendung. Es folgen detaillierte Konzepte weiterer Anwendungen aus dem *Web-Caching*, *Client-Server*-Tests sowie dem *Distributed Object Computing*, die mit ausgearbeiteten Implementierungsplänen versehen sind. Das Kapitel wird mit einigen nur kurz umrissenen, eher langfristig erreichbaren Anwendungsmöglichkeiten der Locust-Infrastruktur zusammen mit schematischen Implementierungsskizzen abgeschlossen.

6.1 Einführung

Anwendungen für die Locust-Infrastrukturen sind Aufgaben, die den drei Hauptprinzipien des Locust-Modells – Multiplikation, Lokalität und Rekursivität – folgen und diese ausnutzen. Diese Aufgaben lassen sich wiederum in Anwendungen des verteilten Hochleistungsrechens (*High Performance*), wie beispielsweise grobgranulare parallele Algorithmen, und des verteilten Hochdurchsatzrechnens (*High Troughput*), wie zum Beispiel Web-Caching und Client-Server-Tests, unterscheiden.

1. Multiplikation – Die Reichweite der Locust-Infrastruktur, und damit ihre Leistungsfähigkeit, kann durch einen hohen Multiplikationsfaktor vervielfacht werden. Dies wird bei den Anwendungen dieses Kapitels zumeist durch Ausnutzung des hierarchischen, baumartigen Aufbaus der Locust-Infrastruktur und seiner Serverdienste erreicht. Mit jeder zusätzlichen Ebene dieser Hierachie vervielfacht sich die Anzahl von Teilnehmern und damit die erzielbare Leistung exponentiell, während der Verwaltungsaufwand nur linear steigt.

   (a) Grobgranulare parallele Anwendungen – Bei grobgranularen parallelen Anwendungen wie der RC5 Dechiffrierung und dem Raytracing wird versucht, mit ei-

nem bestimmten Verwaltungsaufwand (n Verwaltungseinheiten) eine möglichst hohe Rechenleistung in Form von parallel bearbeiteten Schlüsseln oder Pixeln zu erreichen. Mit Hilfe der baumartigen Locust-Infrastruktur lassen sich mindestens $nlogn$ Schlüssel oder Pixel gleichzeitig berechnen.

(b) Web-Caching – Unter Web-Caching wird die Anforderung und Zwischenspeicherung von Web-Dokumenten wie HTML-Seiten und -Grafiken verstanden, mit dem Zweck, zukünftige Anfragen nach diesen Dokumenten schneller und sparsamer aus dem Zwischenspeicher beantworten zu können. Die Effizienz des Caching ist umso besser, je höher die Quote vorrätiger Dokumente (*Hit Rate*) ist und je mehr Clients von der beschleunigten Auslieferung profitieren können. Ein hoher Multiplikationsfaktor des Knotens, der die Caching-Anwendung ausführt, erhöht daher dessen Effizienz.

(c) Client-Server-Tests – Bei Client-Server-Tests soll die Leistungsfähigkeit von Serverdiensten im Gegensatz zur bloßen Simulation von Server-Last mit realen Clients auf tatsächlich vorhandenen Maschinen, die über eine Locust-Applikation gesteuert werden, getestet werden. Diese Anwendung könnte man auch als positiven *Denial of Service*-Angriff bezeichnen, der durchgeführt wird, um die interessanten Parameter – insbesondere die maximale Anzahl von Clients – eines Servers zu ermitteln, bevor er blockiert. Ein hoher Multiplikationsfaktor ermöglich bei dieser Anwendung die Einbeziehung und parametrisierbare Steuerung einer großen Anzahl von Clients, die eine realistische Last auf dem zu testenden Server erzeugen können.

(d) Distributed Object Computing (DOC) – Beim verteilten Rechnen nach dem DOC-Modell wird der hohe Multiplikationsfaktor der Locust-Infrastruktur weniger zur Erzielung möglichst hoher Leistung als vielmehr zur Reduktion des Verwaltungsaufwandes eingesetzt.

2. Lokalität – Der durch die Einführung zusätzlicher Ebenen der hierarchisch aufgebauten Locust-Infrastruktur linear steigende Verwaltungsaufwand läßt sich durch Ausnutzung lokaler Besonderheiten und Eigenschaften deutlich reduzieren. Diese Aufgabe übernimmt ein lokaler Verwalter, der die Besonderheiten des ihm zur Verwaltung unterstellten Teilraumes kennt.

(a) Grobgranulare parallele Anwendungen – Bei der RC5/Raytracing Anwendung werden die Tatsachen ausgenutzt, daß der rechenintensive Client in einer attraktiven Web-Seite eingebettet ist und daß alle Client-Applets einen benachbarten Bereich von Schlüsseln testen. Die erste Tatsache führt dazu, daß die beim Teilnehmer verbrauchte Rechenleistung nicht explizit bezahlt zu werden braucht, sondern indirekt und transparent über den Seiteninhalt abgerechnet wird. Dies verringert die Trasaktionskosten der beteiligten Parteien. Die zweite Tatsache ermöglicht, daß alle Client-Applets einer Web-Seite ihre Schlüssel von einem Schlüssel-Server laden können, der nur einen bestimmten Super- oder *Master*-Block verwalten muß. Dieser Schlüssel-Server kann auf der jeweiligen Web-Site lokalisiert sein. Auf diese Weise reduziert sich der Verwaltungsaufwand und die Server-Last auf dem Hauptserver.

(b) Web-Caching – Caching-Anwendungen beruhen auf Geschwindigkeits- und Latenzvorteilen, die durch Auslieferung lokaler anstelle von entfernten Dokumenten erzielt werden. Genauso wie die Locust-Infrastruktur die Verteilung und Zwischenspeicherung von mobilem Java-Code effizient bewerkstelligt, kann sie verwendet werden, um Web-Dokumente zwischenzuspeichern, um zukünftige Anfragen direkt aus dem Cache beantworten zu können. Die gleichen Klassen und Methoden, die sonst zur Prüfung verwendet werden, ob Java-Code noch gültig ist und gegebenenfalls eine Neuanforderung auslösen, können ebenfalls zur Überprüfung der Gültigkeit von Web-Dokumenten verwendet werden. In anderen Worten wird zunächst der *Update* Mechanimus verwendet, um die Caching-Anwendung vom nächsten Locust-Server auf den lokalen Knoten zu laden und auszuführen, um anschließend mit dem gleichen Mechanismus Web-Dokumente von ihren jeweiligen Servern zu laden und zwischenzuspeichern.

(c) Client-Server-Tests – Zusammengehörige Clients, die aus einem Teilmarkt stammen, können in Test-Szenerien gezielt Schwachstellen des zu testenden Servers auffinden und gebündelt angreifen. Zu solchen, sonst nur schwer auffindbaren Schwachstellen gehören zum Beispiel der gleichzeitige Zugriff verschiedener Clients über eine einzige IP Adresse, wie es zum Beispiel bei Verwendung eines gemeinsamen Proxy-Servers vorkommt.

(d) Distributed Object Computing (DOC) – Die Lokalität kann bei DOC Anwendungen zur Verringerung der Latenz und Bandbreite bei der Kommunikation zum örtlich nächsten Server verwendet werden, der einen bestimmten Dienst anbietet. Dies können insbesondere Dienste sein, die CORBA (noch) nicht unterstützt wie zum Beispiel der Locust *Update*-Server, der für die Aktualität von Code oder von Objekten zuständig ist.

3. Rekursivität – Da die Locust-Infrastruktur einem hierarchischen, baumbasierten Aufbau besitzt, ist es naheliegend die Rekursivität der Infrastruktur auszunutzen. Durch rekursives Design der Locust-Serverdienste erreichbare Vorteile sind verbesserte Leistungsfähigkeit durch Parallelität, Lastverteilung und Redundanz.

   (a) Grobgranulare parallele Anwendungen – Durch Hintereinanderschaltung von Managern zur Schlüssel- oder Pixelverwaltung können die zu bearbeitenden Blöcke besser und damit die Last besser auf die Clients verteilt werden. Der gesamte (Such-)Raum wird von Hauptmanager in Superblöcke aufgeteilt und an untergeordnete Manager delegiert. Diese teilen ihre Superblöcke in gewöhnliche Blöcke auf und geben diese zur Verwaltung an ihre untergeordneten Manager weiter, die jeweils Clients einer einzigen Web-Site mit zu berechnenden Schlüsseln bzw. Pixeln versorgen. Die Koppelung von Managern nimmt die Server-Last vom Haupt-Server weg und verteilt sie gleichmäßig auf alle verfügbaren Server.

   (b) Web-Caching – Durch Kopplung von *Processors*, auf denen eine Caching-Anwendung läuft, entsteht eine Cache-Hierarchie. Caches, die eine Anfrage nach einem Dokument nicht aus dem lokalen Zwischenspeicher erfüllen können, befragen den ihnen übergeordneten Cache und vermeiden daduch unter

Umständen die kostspielige Anforderung des Dokuments vom ursprünglichen Server. Auf diese Weise erhöht sich die *Hit Rate* des Caches und damit seine Effizienz. Gleichzeitig wird die Latenz und Bandbreite zur Verfügungstellung von Web-Dokumenten weiter reduziert. Darüberhinaus können Ort und Anzahl zusätzlicher Caches der Dokumentennachfrage dynamisch angepaßt werden.

(c) Distributed Object Computing (DOC) – Auch DOC Anwendungen profitieren von Cache-Hierachien, da auch sie Code zum Client bringen müssen, bevor eine verteilte CORBA Anwendung initialisiert werden kann. Die Lade- und Initialisierungszeit läßt sich durch das Caching von mobilen Code erheblich reduzieren. Darüberhinaus kann eine Caching-Anwendung leicht die Gültigkeit von Code feststellen und gegebenenfalls eine aktuellere Version anfordern und zur Verfügung stellen.

6.2 Grobgranulare parallele Verfahren

Web-Computing Systeme unternehmen den Versuch, die Rechenleistung hunderttausender ans Internet angeschlossener PCs zu bündeln und zentral für rechenintensive Aufgaben zu nutzen. Der bisher erfolgreichste Versuch, über das Internet gebündelter Rechenleistung einzusetzen, sind verteilte *Brute-Force*-Angriffe auf kryptographische [Riv95] oder mathematische Algorithmen [Rob54] oder Geo-Daten [SET]. Ein *Brute-Force*-Angriff löst ein gegebenes Problem durch Erschöpfung des Suchraums, bis alle Möglichkeiten behandelt sind. Dieser Ansatz eignet sich besonders gut als Anwendung gebündelter Rechenleistung, da er sich in paarweise disjunkte Teilmengen zerlegen läßt.

Eine der erfolgreichsten Gruppen, die verteilte Angriffe auf kryptographische Verfahren organisiert, ist `distributed.net` [dis]. Zu den Erfolgen von `distributed.net` zählen der Gewinn einer Reihe von öffentlichen Wettbewerben, die von RSA Labatories [Sec] zur Dechiffrierung der Verschlüsselungsverfahren DES-II, DES-III und RC5 ausgerufen wurden.
Das Verfahren von `distributed.net` ist einfach und effizient: Jeder Teilnehmer installiert auf seinem Rechner ein auf die Architektur und das jeweilige Verschlüsselungsverfahren zugeschnittenes Client-Programm. Dieses lädt von einem *Key-Server* einen Bereich zu testender Schlüssel. Die Schlüssel werden lokal geprüft und die Ergebnisse an den Server zurückgesendet. Die maschinennahe Optimierung hat leider eine mangelnde Portabilität des Clients zur Folge. Da die Clients auf das jeweilige Problem zugeschnitten sind, muß jeder Teilnehmer zur Bearbeitung eines anderes Verschlüsselungsverfahrens einen neuen Client installieren. Das System ist daher nicht wiederverwendbar.

Eine herausragende Eigenschaft verschiedener Web-Computing-Systeme ist ihre Fähigkeit, einen großen und loyalen Teilnehmerkreis zu mobilisieren. Jede noch so geringe und ineffiziente Rechenleistung kann, auf Grund der Millionen von potentiellen Clients, ausgeglichen werden. `distributed.net` löst dieses Problem über zwei Besonderheiten: ihr gutes Motivationsvermögen sowie ihre Orientierung auf fachkundige Teilnehmer. Das Brechen des Schlüssels ist als Wettbewerb zwischen den Teilnehmern organisiert. Der Finder des richtigen Schlüssels erhält einen Teil des Preisgeldes. Einen Teil behält `distributed.net` selbst und das restliche Preisgeld wird einer gemeinnützigen Organisation gespendet, auf die sich alle Teilnehmer per demokratischer Abstimmung geeinigt haben. Je aktiver ein

Teilnehmer wird, das heißt je mehr Schlüssel er testet, desto größer werden sein Gewinnchancen. Darüberhinaus bietet `distributed.net` die Möglichkeit, Teams zu bilden. Dies erhöht, ähnlich wie bei Lottotip-Gemeinschaften, die Gewinnchancen und damit den Wettbewerb. Als Multiplikatoren setzt `distributed.net` auf fachkundiges Teilnehmerpublikum wie Programmierer und Administratoren. Diese administrieren in der Regel einen umfangreichen Rechnerpark und sind damit in der Lage, die Reichweite des Clients zu vervielfachen. Die technischen Vorzüge des Systems sind:

- hierarchisches System von *Key-Servern* zur Lastverteilung
- geringe Beeinträchtigung der Arbeit des Teilnehmers, da das Programm selbstständig (ohne Interaktion) im Hintergrund abläuft
- schneller, auf die Architektur handoptimierter, Maschinensprache-Kern
- relativ hoher Multiplikationsfaktor durch Orientierung auf fachkundiges Teilnehmerpublikum wie Programmierer und Administratoren

6.2.1 Kryptoanalyse

Dieser Abschnitt behandelt das Design der Beispiel-Anwendung, dem RC5 *Job* [Los99]. Der RC5 Job besteht aus dem RC5 *Manager* und *Worker*. Der in Abbildung 6.1 abgebildete RC5 Manager verwaltet hierzu den Suchraum, visualisiert den Fortschritt der Berechnungen, bietet eine GUI zur Konfiguration und gibt bei Bedarf Meldungen zur Fehlersuche aus.

Der RC5 *Worker* dagegen testet einen Bereich von Schlüsseln und zeigt, wie in Abbildung 6.5 illustriert, den Fortschritt der (lokalen) Berechnungen an.

Der RC5 Algorithmus

Im Jahr 1995 wurde von der RSA Data Security Inc. [Sec] ein neuartiger Verschlüsselungsalgorithmus angekündigt, der RC5 [Riv95] [BR96], der als Alternative zum *Data Encryption Standard* (DES) [Pub93] mit folgenden Zielen entwickelt worden ist:

- einfache, Wort-orientierte symmetrische Block-Verschlüsselung
- an verschiedene Wortlängen einer Vielzahl von Prozessoren anpaßbar
- für kryptographische Schlüssel variabler Länger einsetzbar
- für Soft- und Hardware-Verschlüsselung geeignet
- geringe Speicheranforderungen bei hoher Sicherheit

Das Hauptmerkmal des RC5 gegenüber anderen kryptographischen Verfahren ist sein starker Gebrauch datenabhängiger Rotationen. Der Anteil von Rotationen ist nur von den Eingabedaten abhängig und daher nicht vorherbestimmbar.

Durch die offensive Vermarktung mit Hilfe der angeblichen fundamentalen Überlegenheit gegenüber dem DES hat RSA eine Menge von Leuten herausgefordert, diese Behauptung nachzuprüfen und den Sicherheitsvorsprung von RC5 zu quantifizieren. RSA selbst

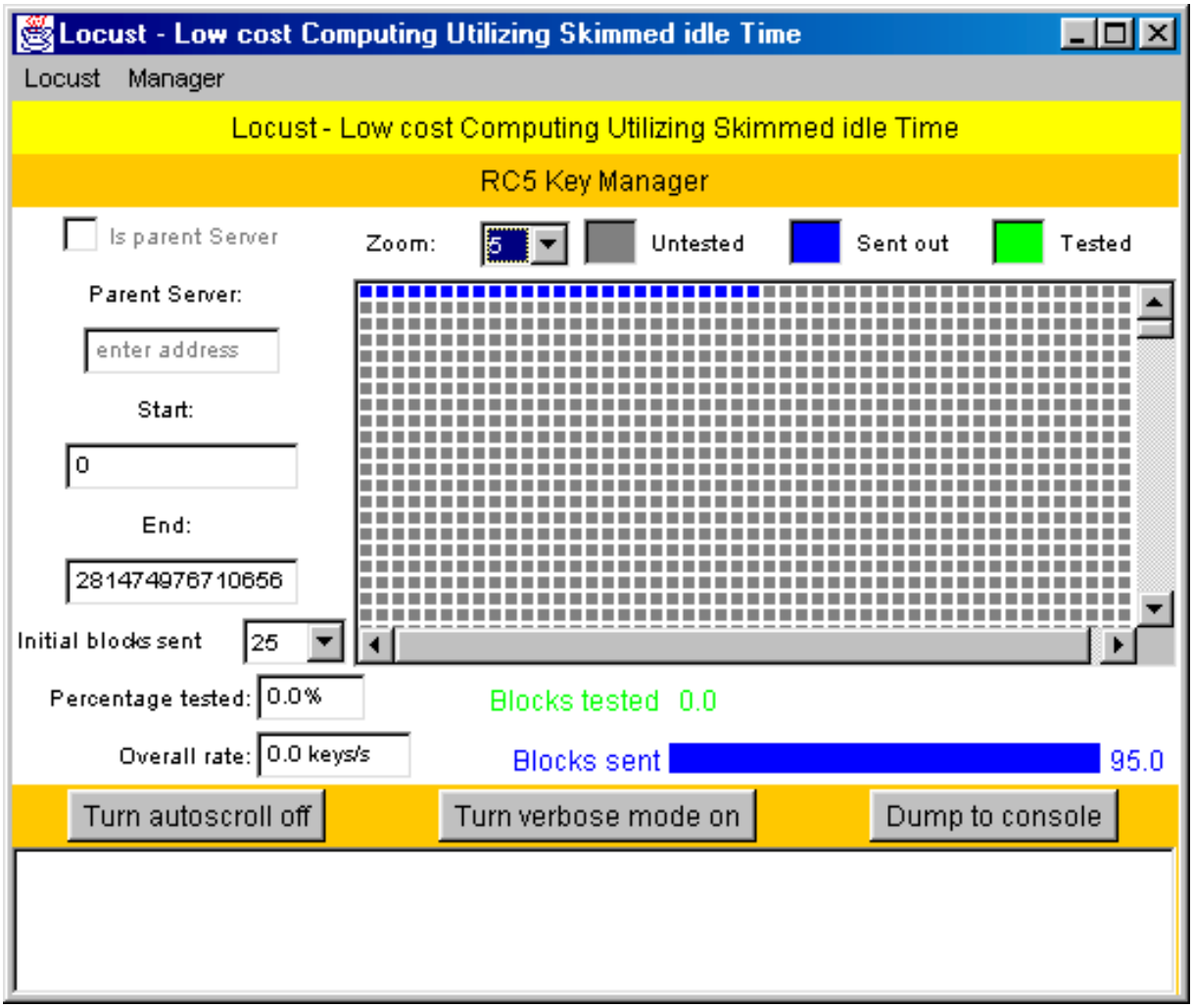

Abbildung 6.1 Screenshot des RC5 *Managers*

hat diese Bemühungen durch eine Reihe kryptographischer Wettbewerbe mit Preisgeldern bis zu 10.000,- Dollar weiter angeheizt. Im Januar 1997 wurde der RSA *Secret-Key* Wettbewerb als Serie von 13 einzelnen Wettbewerben angekündigt, mit dem Ziel, die Sicherheit der DES und RC5 Algorithmen mit verschieden großen Schlüssellängen zu quantifizieren. Inzwischen wurden DES und RC5 Verschlüsselung bis zu einer Schlüssellänge von 56 bit gebrochen, während laufend Anstrengungen unternommen werden, auch die 64 bit Versionen zu "knacken".

Schlüsselverwaltung

Die anspruchsvollste Aufgabe beim RC5 Angriff ist die Schlüsselverwaltung [Los99]. Es gibt bei einer Schlüssellänge von 64 bit 2^{64} mögliche Schlüssel, die überprüft werden müssen, ob sie den verschlüsselten Text entschlüsseln können. Bei sequentieller Vorgehensweise muß nur der aktuelle Schlüssel gespeichert werden, um festzuhalten, daß alle vorhergehenden Schlüssel geprüft und alle nachfolgenden noch zu prüfen sind.

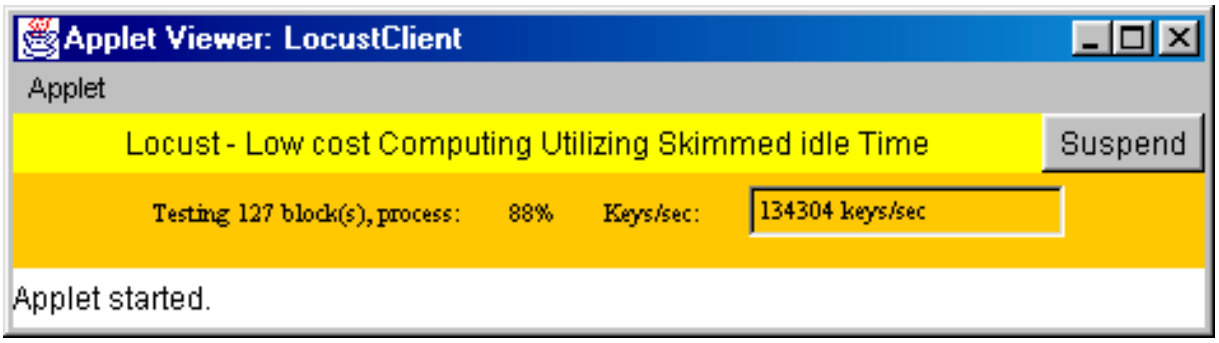

Abbildung 6.2 Screenshot des RC5 *Workers*

Bei paralleler Vorgehensweise wird die Verwaltung um einiges komplizierter. Der günstigste Fall wäre, alle 2^{64} Schlüssel parallel zu bearbeiten, dann läge das Ergebnis sofort vor. Leider benötigt dieses Vorgehen 2^{64} Prozessoren. Wenn die Schlüssel zufällig auf alle verfügbaren Prozessoren verteilt werden, so muß für jeden Schlüssel einzeln erfaßt werden, ob dieser schon getestet wurde oder nicht. Die Kosten hierfür sind 2^{64} bit $= 2^{16}$ Terabyte Speicherplatz. Um den benötigten Speicherplatz zu reduzieren, wurde der Suchraum (alle möglichen Schüssel) in kleine Blöcke von aufeinander folgenden Schlüsseln unterteilt. Dadurch werden sowohl weniger Prozessoren, da jeder Prozessor einen ganzen Block testet, als auch weniger Speicher gebraucht, da nur noch pro Block und nicht pro Schlüssel ein Bit benötigt wird (getestet oder nicht). Dabei wird aber in Kauf genommen, daß die Berechnung länger dauert. Heutige PC-Prozessoren können zwischen 2^{16} und 2^{24} Schlüssel in wenigen Minuten testen. Dies bringt den notwendigen Speicherplatz zumindestens in den Bereich von Gigabytes. Um diesen Speicherplatz noch einmal zu reduzieren, verwendet der Server Blöcke von Blöcken (Super-Blöcke). Erst wenn alle Blöcke eines Super-Blocks getestet sind, wird der nächste angefangen.

Blockgröße

Der Unterschied zwischen 2^{16} und 2^{24} ist enorm und darf nicht unterschätzt werden. Wie groß sollte ein Block sein, um langsame Prozessoren nicht zu über- und schnelle Prozessoren nicht zu unterfordern? Eine Möglichkeit dieses Dilemma zu umgehen, ist eine variable Blockgröße zu wählen. Dies bietet eine Reihe von Vorteilen:

- Die Blockgröße kann nicht nur von der Prozessorleistung abhängig gemacht werden, sondern auch von anderen Parametern, wie zum Beispiel von der durchschnittlichen Verweildauer eines *Web*-Terminals auf einer Web-Seite mit dem Locust Applet.

- Falls der Client in seiner Arbeit (z.B. durch ein Verlassen der Web-Seite) unterbrochen wird, kann der Client dem Server mitteilen, wie weit getestet worden ist.

Die variable Blockgröße bringt aber auch Nachteile mit sich. Besonders gravierend ist die gestiegene Speicherplatzanforderung, da jetzt die Länge des Blockes mit abgespeichert werden muß. Bei einer maximalen Blockgröße von 2^{24} werden 3 Bytes benötigt, um die Länge zu speichern, damit ist der gesamter Zugewinn durch den Einsatz von Blöcken

CPU	Takt	OS	Java VM	Schl./s	nativ
Pentium II	266 MHz	Windows NT 4.0	Netscape 4.03	157,000	660,000
AMD K6	200 MHz	Windows 95	Netscape 4.03	118,000	361,000
Pentium Pro	200 MHz	Windows NT 4.0	Netscape 4.03	116,000	430,000
Cyrix 6x86	166 MHz	Windows 95	Netscape 4.03	74,700	296,000
AMD K6	200 MHz	Windows 95	Internet Explorer 4.0	52,700	361,000
Pentium Pro	200 MHz	Windows NT 4.0	Internet Explorer 3.02	52,000	483,000
Pentium	100 MHz	Windows NT 4.0	Netscape 4.03	37,800	130,000
UltraSPARC	167 MHz	Solaris 2.6	Sun JDK	20,300	
Pentium	133 MHz	OS/2 Warp 4	Netscape 2.02	8,250	
PowerPC 604e	200 MHz	MacOS	Netscape 4.03	6,200	642,000
Pentium II	300 MHz	Linux	Sun JDK	5,200	
PowerPC	180 MHz	MacOS	Internet Explorer 4.0	3,200	
Pentium II	300 MHz	Linux	Netscape 4.03	2,300	
PowerPC 604e	200 MHz	MacOS	Runtime for Java 1.5	2,150	642,000
68040	40 MHz	MacOS 8	Runtime for Java 1.5.1	1,380	32,000
Mips R4400	200 MHz	Irix 5.3	Netscape 3.01	1,050	
UltraSPARC	167 MHz	Solaris 2.6	Netscape 4.03	1,050	
PowerPC 601	120 MHz	MacOS 7.5.5	Netscape 3.01	920	212,000
SPARC 5	110 MHz	Solaris	Netscape 4.03	587	

Tabelle 6.1 Schlüsselraten der Java- und der nativen RC5-Implementierung

dahin. Aus diesem Grund wird eine feste Blockgröße verwendet. Um aber auf die Vorteile einer variablen Blockgröße nicht verzichten zu müssen, bearbeitet jeder Client mehrere Blöcke auf einmal. Die Speicherplatzanforderung entspricht der von festen Blockgrößen, aber mit annähernd den Vorteilen von variablen Blockgrößen mit einer minimalen Größe der Blöcke.

Scheduling

Da mit einer variablen Anzahl von Blöcken gearbeitet wird, muß die Schedulingstrategie entscheiden, wieviele und welche Blöcke ein Client bearbeiten soll. Eine gute Schedulingstrategie besitzt folgende Eigenschaften:

1. Optimale Zuteilung – Die Schedulingstrategie weist dem Client genausoviel Blöcke zu, wie dieser bearbeiten wird, um die notwendige Kommunikation möglichst gering zu halten.

2. Geringe Fragmentierung – Ist der Suchraum stark fragmentiert, müssen sehr oft wenige Blöcke einem Leistungserbringer zugewiesen werden, was zu unnötiger Kommunikation führt.

3. Effizienz – Wollen sehr viele Clients bedient werden und sind sehr aufwendige Berechnungen beim Scheduling durchzuführen, führt dies zu einem Engpaß. Dies läßt sich durch mehrere Zuweisungs-Threads umgehen. Allerdings darf stets nur ein Thread

Rechner	OS	Java VM	Schl./sek.
Sun Ultra 60/2300	SunOS 5.7	1.1.7	191213
3 Sun Ultra 60/1300	SunOS 5.7	1.1.7	420822
4 Sun Ultra 60/1300	SunOS 5.7	1.1.7	519148
5 Sun Ultra 60/1300	SunOS 5.7	1.1.7	628236
6 Sun Ultra 60/1300	SunOS 5.7	1.1.7	746799
10 Sun Ultra 60/1300	SunOS 5.7	1.1.7	1176170
17 Sun Ultra 60/1300	SunOS 5.7	1.1.7	1756241

Tabelle 6.2 Schlüsselraten der verteilten RC5 Anwendung

gleichzeitig auf die Verwaltungs-Datenstruktur zugreifen, um die Schedulingstrategie durchzuführen und doppelte Zuweisungen zu verhindern. Der hierzu notwendige Synchonisationsaufwand kann allerdings kontraproduktiv wirken.

Die erste Forderung läßt sich nur bedingt erfüllen, denn im allgemeinen kann man über die Verweilzeit eines *Web*-Terminals auf einer Locust Web-Seite keine Aussage machen. Auch die zweite Bedingung hängt indirekt von der Verweildauer des Terminals ab und läßt sich daher nur bedingt erfüllen. Beendet der Terminal das Applet (z.B. durch Verlassen der Seite), benachrichtigt das Applet den Server, welche Blöcke es schon getestet hat. Wurden nicht alle zugewiesenen Blöcke getestet, führt dies zu Fragmentierung. Lediglich die dritte Bedingung läßt sich, wenn auch auf Kosten der anderen beiden, gut erfüllen. Die implementierte Schedulingstrategie weist dem Applet beim ersten Kontakt mit dem Server eine bestimmte Grundanzahl von Blöcken zu. Schafft es das Applet, alle zu testen, bekommt es beim nächsten mal doppelt so viele zugewiesen, maximal 127. Findet die Schedulingstrategie keinen Satz von aufeinanderfolgenden Blöcken mit der gewünschten Anzahl, wird die Anzahl halbiert (und auf die nächste ganze Zahl aufgerundet). Es wird immer der zuerst passende Satz mit der gewünschten Anzahl von Blöcken dem Applet zugewiesen. Dies verhindert keine Fragmentierung, ist aber effizient. Auch wenn in den ungetesteten Blöcken die perfekte Lücke für den Client gefunden wird, bleibt unsicher, ob das Applet alle Schlüssel testen wird.

Abbruch Wenn der Block mit dem korrekten Schlüssel von einem Client getestet wird, bricht dieser ab und sendet den korrekten Schlüssel an den Manager. Der Manager meldet dann `Found key` und der `LocustServer` versendet den Job nicht mehr an Clients.

Geschwindigkeit Die maßgeblichen Eigenschaften des RC5 Verfahrens, die für eine leichte Implementierung des Algorithmus in C oder Java verantwortlich sind, sind die Einfachheit sowie die Unabhängigkeit von Hardware-Unterstützung. Wegen der mangelnden mathematischen Unterstützung oder geeigneter Bibliotheken in Java, kann mit Hilfe einer naiven Implementierung der in [Hew] beschriebenen Initialisierungs-, Ver- und Entschlüsselungsalgorithmen unter Zuhilfenahme von simplen Schleifen nur relativ geringe Leistungsfähigkeit erreicht werden (Vgl. Tabelle 6.1).

Um eine mit nativer Implementierung standhaltende Leistung zu erreichen, wurden in den entwickelten Java Klassen alle Schleifen aufgebrochen (*loop unrolling*). Die gemes-

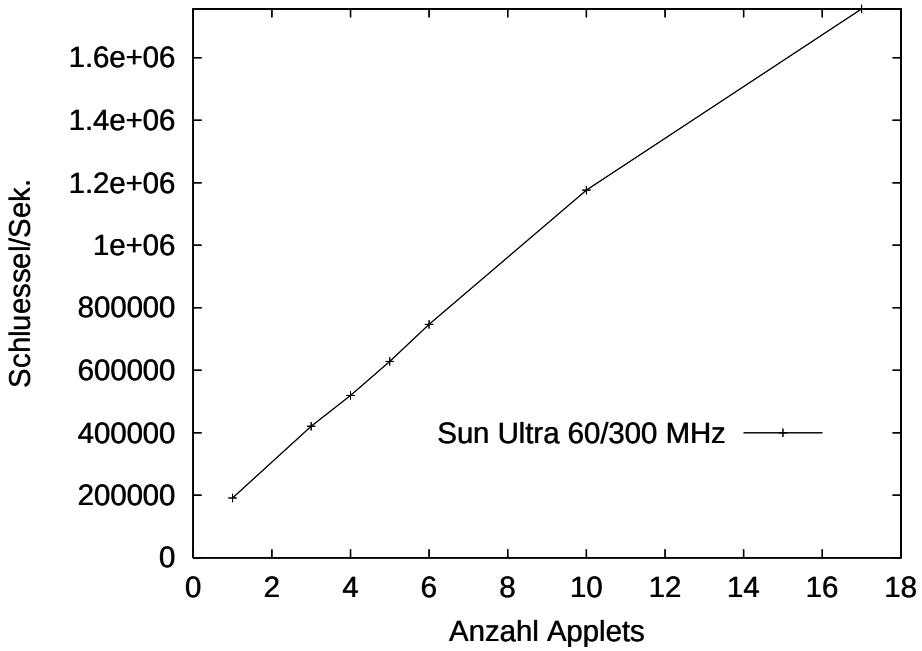

Abbildung 6.3 Mit dem Appletviewer ermittelter Speed-Up der RC5 Anwendung

senen Leistungswerte in Tabelle 6.3 der optimierten Klassen zeigen, daß mit Hilfe einer effizienten Java Laufzeitumgebung mit *Just-In-time* (JIT) Compilation bis zu einem Fünftel der Raten einer nativen Implementierung mit Hand-optimiertem Assembler Code [dis] erreicht werden können. Aus diesem Grund kann die Leistung jeder nativen RC5 Anwendung durch Verwendung von mehr als fünf Applets eingeholt und übertroffen werden [May99a].

Leistung Wie die Testergebnisse in Abbildung 6.4 zeigen ist die technische Umsetzung von Locust gelungen. Das System ist gut skalierbar. Für den Appletviewer wurden die in Tabelle 6.3 angegebenen Werte ermittelt. Die Leistungswerte für den Communicator von Netscape sind enttäuschend und bedürfen noch einer genaueren Untersuchung. Die mit dem Internet Explorer von Microsoft erzielten Werte liegen dagegen im Bereich der Appletviewer Daten.

1. Netscape Communicator 4.7, Java 1.1.5:

   (a) Sun Ultra 10 (440-MHz UltraSPARC-IIi, 256 MB RAM): 3127 keys/sec

2. Internet Explorer 5:

   (a) Pentium II 350 MHZ, 192 MB Hauptspeicher, JDK 1.2: 191768 keys/sec

   (b) Pentium II 350 MHZ, 192 MB Hauptspeicher, JRE 1.1.7B: 191561 keys/sec

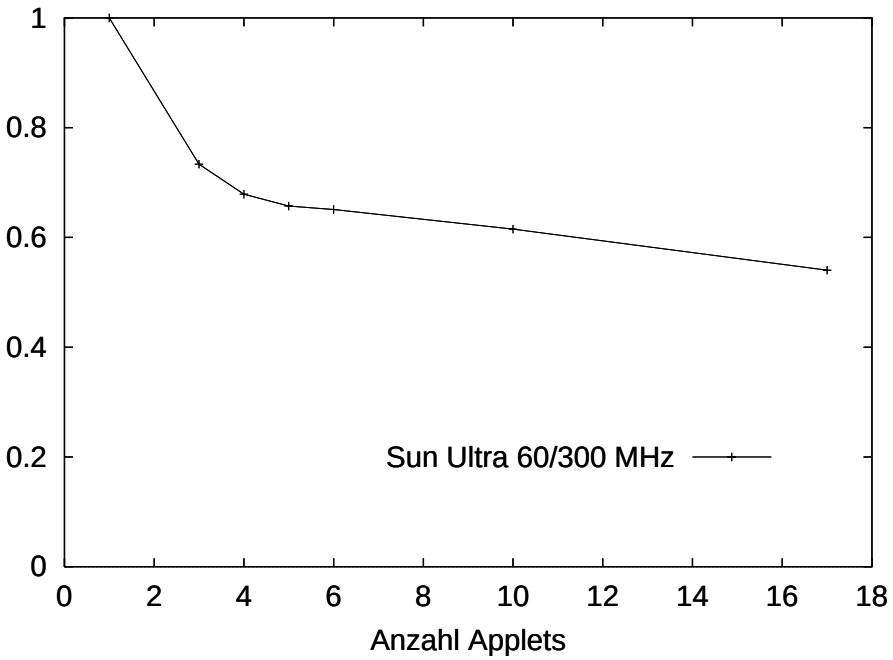

Abbildung 6.4 Mit dem Appletviewer ermittelte Effizienz der RC5 Anwendung

6.2.2 Raytracing

Dieser Abschnitt behandelt das Design der verteilten *Raytracing* Anwendung. Der Raytracing Job besteht aus dem Raytracing *Manager* und *Worker*. Der *Manager* verwaltet das zu rendernde Bild, visualisiert den Fortschritt der Berechnungen und gibt bei Bedarf Meldungen zur Fehlersuche aus.

Der Raytracing *Worker* dagegen berechnet einen Bereich, genauer eine Zeile, von Pixeln und zeigt den Fortschritt der (lokalen) Berechnungen an.

Der Raytracing Algorithmus *Raytracing* ist ein numerisches Verfahren, das für sehr lebensnahe, Foto-realistische synthetische Bilder verantwortlich ist. Einige der verblüffendsten Spezial-Effekte der Filmwelt der vergangenen Jahre verdanken ihre Existenz dem Raytracing.

Rechner-generierte Bilder benötigen eine mathematische Beschreibung der zu erzeugenden Szenerie. Sämtliche Details müssen dabei aus dreidimensionalen mathematischen Beschreibungen einfacher geometrischer Einheiten wie Polygonen, Ebenen, Würfeln, Quadern, Zylindern und Linien zusammengesetzt werden. Um realistische Bilder zu erzeugen, sollte der verwendete Algorithmus ebenfalls Farbe und Licht sowie alle notwendigen physikalischen Seiteneffekte wie Schatten, Reflektionen, Transparenz, Helligkeit und Dunkelheit so nahe an der Realität wie nur irgend möglich modellieren. Bei diesen sehr rechenintensiven Modellierungen setzen preiswertere Bilderzeugungs-Verfahren einige Optimierungsmaßnahmen und Berechnungsabkürzungen zugunsten geringeren Ressourcenverbrauchs ein. Das Raytracing Verfahren dagegen vernachlässigt die Geschwindigkeit der Berechnung zugunsten höherem Darstellungsrealismus. Hierzu bildet es den Weg des

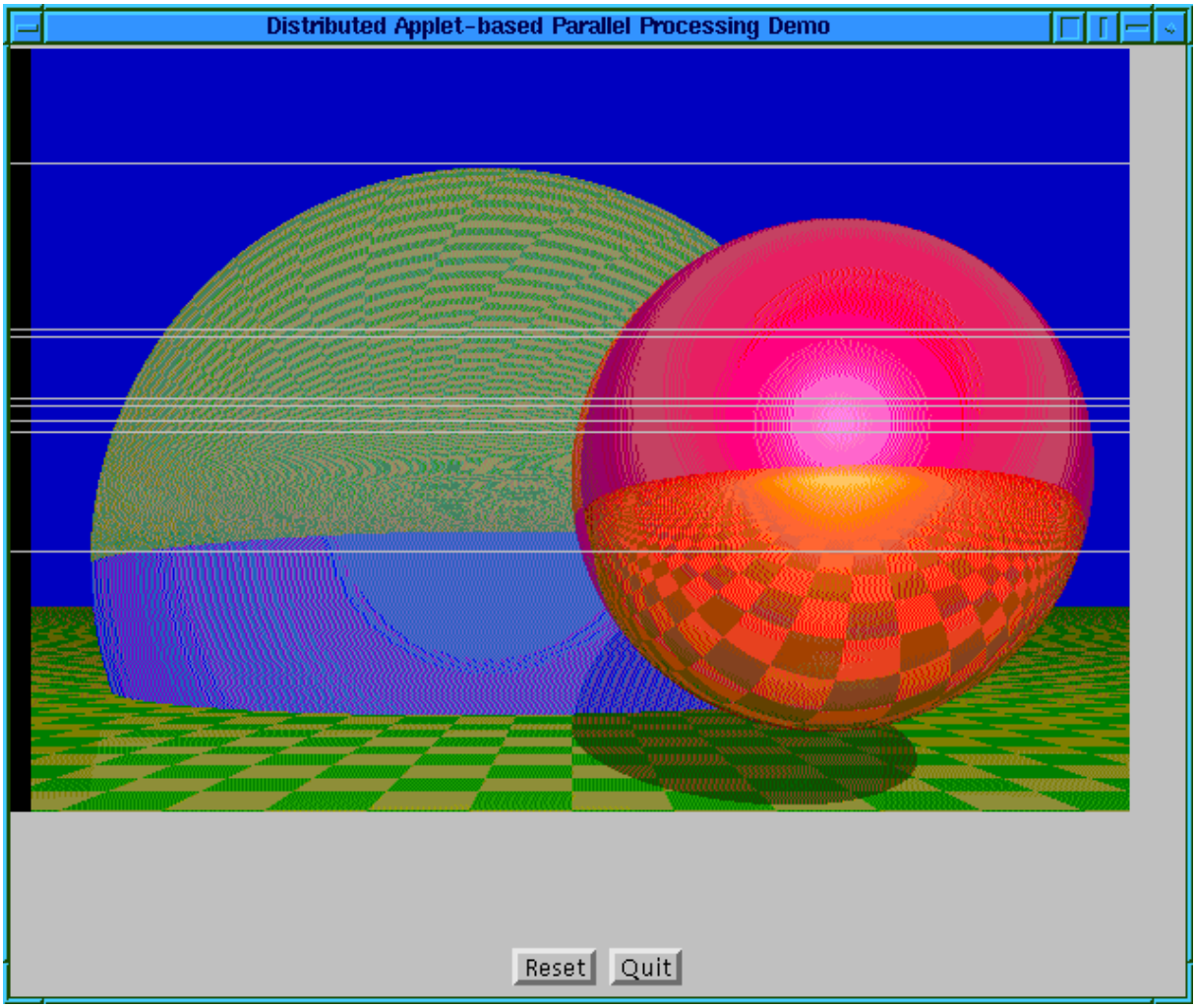

Abbildung 6.5 Screenshot des Raytracing *Managers*

Lichtstrahls nach, vom Austreten aus einer oder mehrerer Lichtquellen, über die Interaktion mit dem physikalischen Modell und schließlich bis zum Auftreffen auf der Netzhaut des Betrachters. Durch diese Modellierung einzelner Lichtstrahlen ist Raytracing ein zwar sehr simpler, aber unglaublich aufwendiger *Rendering*-Algorithmus: für jedes zu berechnende Pixel wird ein imaginärer Lichtstrahl vom Auge des Betrachters (des Betrachtungsstandpunktes) durch die Bildebene, die die Bildpunkte enthält, bis zum Auftreffen auf die zu modellierende Bildszene gezogen. Im Auftreffpunkt ermittelt der Raytracing Algorithmus die Farbe und Intensität des Pixels durch Modellierung weiterer Lichtstrahlen, die diesmal von den Lichtquellen des Bildmodells ausgehen.

Der Raytracing Algorithmus ist damit ein *Brute-Force* Verfahren, das heißt jeder Bildpunkt kann vollständig unabhängig von allen anderen berechnet werden. Daher eignet sich dieses Verfahren besonders zur Anwendung mit gebündelter Rechenzeit. Hierfür wurde ein bestehender Raytracing Code in Java [dM] an die Locust-Infrastruktur angepaßt.

```
    while ( runs-- > 0 ) {

        // contact server and ask it to give us a job to do..
        byte[] jobDetails = requestJob();

        // extract job information from byte packet
        scanlineToDo = (jobDetails[0] & 255) << 8;
        scanlineToDo+= (jobDetails[1] & 255);

        computedScanline[index++] = (byte) (scanlineToDo >> 8);
        computedScanline[index++] = (byte) (scanlineToDo & 255);

        for(i=0; i < IMAGE_WIDTH; i++) {

                ray.normalize();
                Trace(viewPoint,ray,false,0);

                computedScanline[index++] = (byte)(cor.x*255);
                computedScanline[index++] = (byte)(cor.y*255);
                computedScanline[index++] = (byte)(cor.z*255);

                dx = dx+step;
        }

        try {
            returnResults ( computedScanline );
        }
}
```

Abbildung 6.6 Hauptschleife des Raytracing-Workers

Pixelverwaltung Der Raytracing Job besteht aus einem Worker und einem zugehörigen Manager. Der Manager verteilt das zu berechnende Bild durch Verschicken von Job Spezifikationen, die jeweils den Index einer Bildzeile enthält. Die für die Berechnung der Zeile notwendigen Szenerie-Informationen erhält der Worker durch das Laden der entsprechenden Raytracing Klasse. Nach der Fertigstellung der Bildzeile liefert das Applet mittels `returnResults` eine komplette 24-bit Farbzeile als Ergebnis an den Raytracing Manager zurück, die sofort im Raytracing Manager dargestellt wird, um den Berechnungfortschritt zu visualisieren.

Um die Verteilung an die *Worker* Applets zu verwalten, unterhält der Raytracing Manager eine zentrale Datenstruktur, nämlich ein Feld von Boolschen Werten. Die Anzahl dieser Werte entspricht der Höhe der zu berechnenden Grafik (`IMAGE_HEIGHT`). Diese *Flags* halten fest, welche der Bildzeilen bereits erfolgreich versendet und wieder empfangen wurden und welche noch nicht.

Nach Ankunft eines Ergebnisses, prüft der Manager mittels `collateResults()` zunächst, ob der entsprechende Job (Pixel oder Zeile) ausständig ist. Das fehlertolerante

Rechner	OS	Java VM	Sek.
1 Sun Ultra 60/1300	SunOS 5.7	1.1.5	245,22
2 Sun Ultra 60/1300	SunOS 5.7	1.1.5	154,20
4 Sun Ultra 60/1300	SunOS 5.7	1.1.5	109,98
8 Sun Ultra 60/1300	SunOS 5.7	1.1.5	66,99
12 Sun Ultra 60/1300	SunOS 5.7	1.1.5	58,85
16 Sun Ultra 60/1300	SunOS 5.7	1.1.5	51,71
20 Sun Ultra 60/1300	SunOS 5.7	1.1.5	45,54
32 Sun Ultra 60/1300	SunOS 5.7	1.1.5	56,43

Tabelle 6.3 Rendering-Zeiten der verteilten Raytracing Anwendung

Eager Scheduling des Locust Managers sorgt dafür, daß durchaus mehrere *Worker* am selben Teilproblem arbeiten können. Treffen daher zwei Ergebnisse des gleichen Jobs ein, wird der zuletzt eingetroffenen Job verworfen.

Aus dieser Fehlertoleranz ergibt sich das Problem, daß zwei Steuerflüsse, die gleichzeitig schreibend auf die entsprechende Verwaltungstruktur zugreifen, diese unter Umständen inkonsistent machen können. Das Boolsche Feld wird darüberhinaus noch durch einen Zähler ergänzt, der festhält, wieviele Ergebnisse bereits eingetroffen sind. Beide Datenstrukturen bilden eine logische Einheit und müssen strikt konsistent gehalten werden, damit der Manager seine Buchführungsaufgaben erfüllen kann. Hierzu ist ein Synchronisationsmechanismus notwendig, der vor einem schreibenden Zugriff auf diese Datenstrukturen belegt werden muß. In Java steht hierzu der Schlüsselbegriff `synchronized` zur Verfügung.

Blockgröße Um auch sehr komplexe Modelle und verschiedene Größen zu berechnender Grafiken zu unterstützen, läßt sich die Blockgröße eines Raytracing Blocks parametrisieren. Bei sehr komplexen Modellen läßt sich das zu rendernde Bild pixelweise aufteilen und parallel rechnen, während sehr einfache Modelle oder sehr kleine Grafiken zeilenweise verschickt und gerendert werden können. Für Zwischengrößen lassen sich auch mehrere Pixel zu einem einzelnen Block zusammenschnüren.

Auch wenn die bisherige Beschreibung dies suggeriert, werden jedoch keine Bildpixel oder -zeilen tatsächlich verschickt. Vielmehr wird nur der Index oder die Indizes der zu rendernden Bildpunkte versandt. Die Modell- und Bilddaten sind Bestandteil der Raytracing Klasse, die unmittelbar nach der Initialisierung des *Workers* vom Locust Server angefordert wird. Die Hauptschleife des Raytracing-Workers ist teilweise in Abbildung 6.6 zu sehen.

Geschwindigkeit Die Leistungsbetrachtungen wurden in einer LAN-Umgebung auf mehreren Sun Ultra 60/1300 bzw. 2300 Workstations unter Solaris Version 5.7 durchgeführt, die mit einem bzw. zwei 64bit-300MHz-UltraSPARC II-Prozessor und 384 MB RAM ausgestattet waren. Als Browser kam Netscape Communicator 4.7 mit der Java Virtual Machine Version 1.1.5 zur Anwendung.

Die Raytracing-Anwendungen bietet in einem weit nutzbaren Bereich der Zahl der

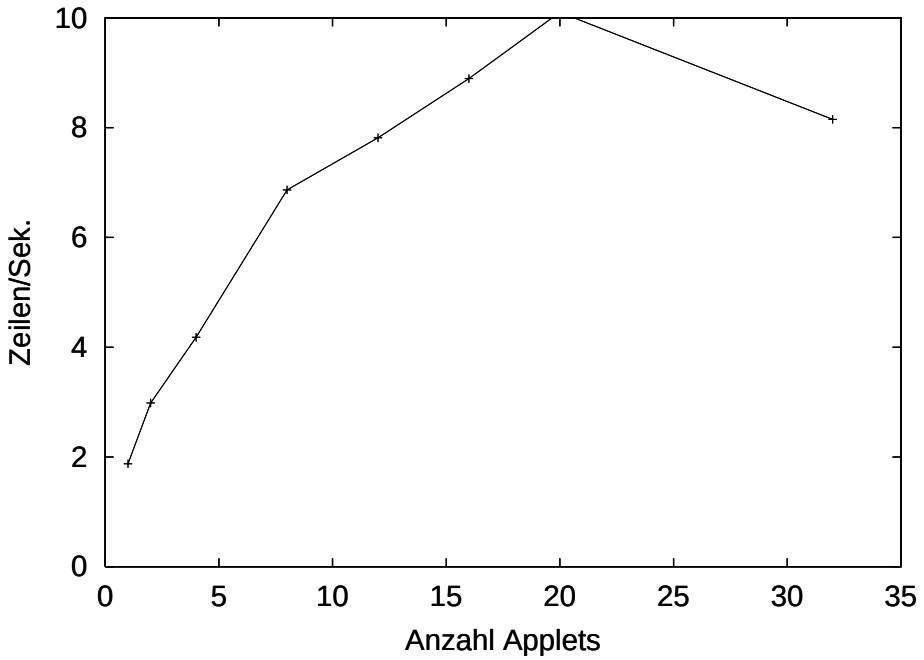

Abbildung 6.7 Mit Communicator 4.7 ermittelter Speed-Up der Raytracing Anwendung

Teilnehmer skalierbare Leistung (Vgl. Abbildungen 6.7 und 6.8). Nur am Anfang und Ende des Leistungsspektrums machen sich der zusätzliche Aufwand der Locust-Infrastruktur sowie fehlende Hierachie-Ebenen zur Reduzierung der Verwaltungs- und Transaktionskosten bemerkbar.

6.3 Zwischenspeicherung - Caching

Bei (Web-) *Caching* handelt es sich um die temporäre Speicherung von (Web) Objekten wie HTML-Dokumenten und -Grafiken, um die weitere Verteilung dieser Dokumente zu erleichtern. Caching bietet drei grundlegende Vorteile gegenüber anderen Techniken wie Datenkomprimierung und verbesserten, schnelleren Netzwerktechnologien: geringerer Bandbreitenbedarf (durch Ausnutzung von *Multiplikation*), geringere Server-Last (ebenfalls unter Ausnutzung von Multiplikationseffekten) und geringere Latenz (durch Ausnützung der *Lokalität*) bei der Auslieferung von Web-Objekten. Diese drei Verbesserungen werden durch eine geringere Zahl von Objekten erreicht, die angefordert, vom Server bearbeitet und ausgeliefert werden müssen, da ein großer Teil der Anfragen aus dem (lokalen) Zwischenspeicher erfüllt werden kann, der aufgrund der örtlichen Nähe eine geringere Reaktionszeit besitzt. Insgesamt reduziert Caching die laufenden Ressourcen-Kosten von Web-Anwendungen, und zwar im Hinblick auf die Ressourcen Zeit, Geld und Infrastruktur.

Zur Implementierung eines Caching-Modells, das die Prinzipien der Multiplikation und der Lokalität ausnutzt, ist natürlich eine Infrastruktur besonders geeignet, die entwickelt wurde, um genau diese Prinzipien zu unterstützen und zu verwenden.

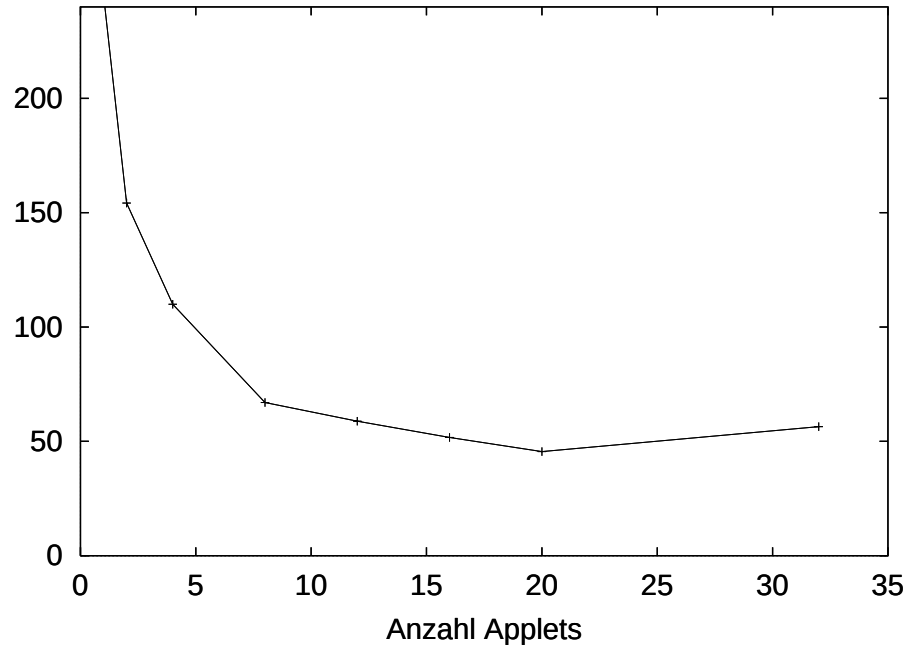

Abbildung 6.8 Mit Communicator 4.7 ermittelte Effizienz der Raytracing Anwendung

6.3.1 Einführung

Definition 6.1 (Web-Cache) *Ein* Web-Cache *befindet sich zwischen Web-Servern (den Ausgangsservern) und einem oder mehreren Clients und vermittelt den Austausch von Anfragen und Antworten zwischen Client und Server. Von jedem vorbei kommenden Web-Objekt (wie HTML-Seiten und -Grafiken) legt der Cache eine Kopie an, um damit zukünftige Anfragen nach diesem Objekt, innerhalb seiner Gültigkeitsdauer, zu beantworten.*

Das *Caching* von Objekten verbessert die Client-Server-Interaktion im WWW auf drei Ebenen:

1. Bandbreite – Da jedes zwischengespeicherte Objekt nur ein einziges Mal vom ursprünglichen Server angefordert und geladen wird, vermindert sich der Bandbreitenbedarf der anfordernden Partei, beispielsweise ein Proxy oder ein Client. Ausgenutzt wird hierbei die Tatsache, daß ein Objekt repliziert bzw. seine Reichweite **multipliziert** wird. Falls der Anforderer seine Bandbreite nach übertragenem Datenvolumen bezahlt, reduzieren sich hierdurch seine Kosten.

2. Latenz – Weil die Anfrage nach einem Objekt aus einem (lokalen) Cache bedient werden kann, der sich näher beim Client befindet, benötigt der Client (Browser) weniger Zeit, um das Objekt zu holen und darzustellen. Ausgenutzt wird hierbei die **Lokalität** der Zwischenspeicher, die ihre Clients auf direkterem und damit schnellerem Wegen erreichen können als der Ursprungsserver. Auf diese Weise erscheinen Web-Seiten schneller.

3. Server-Last – Wenn der Server seine Objekte an Proxies zustellt, die diese wiederum

an ihre vielen Clients weiterreichen, erhöht der Ursprungs-Server die Reichweite bzw. den **Multiplikationsfaktor** seiner Objekte ohne dafür selbst Ressourcen zu verbrauchen. Ausgenutzt wird die multiplizierte Reichweite durch Objekt-Auslieferung an Proxy-Server. Dadurch reduziert sich die dem Ursprungs-Server aufgebürdete Server-Last.

6.3.2 Caching-Algorithtmen

Caching-Algorithmen verwalten Objektkopien in erster Linie über die sogenannte *Frische*, *Gültigkeit* und *Validatoren* der Objekte. Eine *frische* Instanz eines Objekts kann sofort einem Client zugestellt werden, während eine auf seine Gültigkeit überprüftes Objekt zumindest die erneute Anforderung des gesamten Objektes von Ursprungs-Server vermeidet.

Definition 6.2 (Frische) *Ein (Web-) Objekt ist frisch, falls es ohne Überprüfung seiner Gültigkeit an Anfrager weitergegeben werden kann.*

Definition 6.3 (Gültigkeit) *Ein (Web-) Objekt ist gültig, falls sich der Cache beim Ursprungs-Server von der Gültigkeit seiner Kopie des Objekts überzeugt hat.*

Definition 6.4 (Validator) *Ein Validator ist eine externe Objektmethode zur Überprüfung der Gültigkeit des Objektes.*

Die Anforderung eines Objektes wird von einem Cache wie folgt bearbeitet:

1. Falls der Kontext eines Objektes einer Zwischenspeicherung widerspricht, wird keine Kopie angelegt. Auch wenn das Objekt keinen Validator besitzt, kann es nicht gecachet werden.

2. Wenn sich der Client dem Objekt gegenüber authentifiziert oder die Verbindung *sicher* (über SSL) ist, kann das Objekt nicht zwischengespeichert werden.

3. Ein gespeichertes Objekt wird als frisch erachtet, das heißt es kann ohne Überprüfung der Gültigkeit an den Client gesandt werden, falls

   (a) es ein Verfallsdatum besitzt, das noch nicht abgelaufen ist

   (b) ein Browser Cache (der einmal pro Sitzung seine Objekte prüft) bereits seine Gültigkeit geprüft hat

   (c) ein Proxy Cache bereits seine Gültigkeit geprüft hat

4. Falls ein Objekt keine Gültigkeit mehr besitzt, es also *verfallen* ist, muß der Cache die Gültigkeit erneut bein Ursprungs-Server prüfen, ob die Instanz des Objekts noch gültig ist, bevor die Kopie an einen Client geliefert werden kann. Andernfalls wird die aktuelle Version des Objekts angefordert.

6.3.3 Cache-Arten

Caching kann an unterschiedlichen Stellen in der Kette zwischen Client, Netzwerk und Server angesetzt werden.

Client Caching Lokale Zwischenspeicherung kann bereits im Client erfolgen und wird in aktuellen Generationen verbreiteter Browser bereits transparent durchgeführt. Daneben existieren eine Reihe von Anwendungen, die das Browser Caching mit Hilfe von größeren Zwischenspeichern, besserer Konfigurierbarkeit oder größerer Leistung erweitern oder ersetzen. Allen Systemen gemeinsam ist die Tatsache, daß sie Web-Objekte verschiedener Server für einen *einzigen* Benutzer speichern. Vorteil von persönlichen Caches ist in erster Linie die Reduktion der Latenz.

Proxy Caching Führt man die Zwischenspeicherung zwischen Client und Server ein, spricht man *Proxy* (Stellvertreter) Caching. Proxies befinden sich oft in der Nähe von *Gateways*, also Routern zwischen zwei Netzwerken, um teuren Verkehr auf kostspieligen Standleitungen zu reduzieren. Solche Caching Systeme bedienen sehr viele Benutzer (Clients) mit Objekten verschiedender Server. Der Nutzen solcher Proxies besteht in erster Linie in der Zwischenspeicherung von Objekten, die von einem Benutzer angefordert wurden, um damit spätere Anfragen anderer Nutzer zu erfüllen. Um die Leistungsfähigkeit weiter zu erhöhen, werden Proxies oft hierarchisch organisiert, so daß Proxies zunächst benachbarte oder übergeordnete Proxies nach Web-Objekten befragen, bevor das angeforderte Objekte direkt beschafft wird. Hauptsächlich verwendeter Vorteil von Proxy-Konfigurationen ist die Verringerung der Bandbreite.

Server Caching Schließlich können Zwischenspeicher direkt an oder vor einem Server betrieben werden, um die Anzahl von Anforderungen zu reduzieren, die der Server zu beantworten hat. Die meisten Proxies können ebenfalls als Server Cache betrieben werden und werden dann meist als inverser Cache (*reverse cache, accelerator*) bezeichnet. Server-Caches sollen in erster Linie die Server-Last reduzieren und bedienen hierzu viele Clients aus einem einzigen Server.

6.3.4 Caching als Locust-Anwendung

Wie im Kapitel 6.4.5 beschrieben, unterstützt die Locust-Infrastruktur bzw. seine Teilmärkte eine Zwischenspeicherung von umfangreichem mobilen Java-Code, um weitere Anfragen nach diesen Klassen schneller beantworten zu können. Dieser Caching-Mechanismus kann aber genauso gut verwendet werden, um Web-Dokumente zwischenzuspeichern und zukünftige Anfragen direkt aus dem lokalen Dateisystem – oder dem Locust-Server der nächsthöheren Ebene als Proxy – beantworten zu können. Es existieren Methoden zur Überprüfung der Frische und Gültigkeit dieser Java-Klassen und gegebenenfalls zur erforderlichen Neuanforderung. Aufgrund des orthogonalen Designs des Locust-Modells können die gleichen Methoden ebenfalls zur Überprüfung der Gültigkeit und Aktualisierung von Web-Dokumenten verwendet werden. Der *Update*-Mechanimus von Locust wird also zunächst dazu verwendet, die Caching-Anwendung selbst vom nächsten Locust-Server auf den lokalen Knoten zu laden und auszuführen. Anschließend werden mit

Hilfe des gleichen Mechanismus Web-Dokumente von ihren jeweiligen Servern geladen und zwischengespeichert.

Der Caching-Mechanismus von Locust Locust-Applets sind aufgrund der in Kapitel 5 erarbeiteten Entwurfsbedingungen von geringer Größe und schnell zu laden. Anwendungen für Locust-Teilmärkte dagegen können wegen ihrer komplexeren Funktionalität durchaus sehr umfangreich und entsprechend langsam zu laden sein. Aus diesem Grund unterstützt der Locust-Market die lokale Zwischenspeicherung von Java Code. Klassen und Archive von entfernten Maschinen werden auf dem lokalen Dateisystem abgelegt, um zukünftige Anfragen nach diesen schneller beantworten zu können.

Zum Update erstellt die gemietete Maschine zunächst einen Index der bereits vorhandenen Dateien, erzeugt auf diesem Index eine Prüfsumme (CRC) und sendet diese als Parameter der `validate()` Methode an den *Update* Server. Der Server validiert die Gültigkeit der Objekte und gibt bei Bedarf einen Satz von *add/update/delete*-Operationen zurück, die die Client Maschine mit dem *Update* Server synchronisieren.

Jedes lokale Objekt wird nach der Validierung für die Dauer des Reservierungszeitraums als frisch betrachet, es können aber auch beliebige andere Caching-Bedingungen vergeben werden. Nach Reservierungsende werden die Objekte ungültig, um eine Neuanforderung der Klassen oder Archive auszulösen.

Der Caching-Mechanismus für Web-Dokumente In diesem Abschnitt folgt eine detaillierte Beschreibung des Caching-Mechanismus zur Implementierung von Web-Caching.

1. Eine freie Maschine startet den *Processor* und wird nach erfolgreicher Verhandlung mit einem Mieter für die Caching-Anwendung reserviert

2. Der *Processor* kontaktiert den *Update*-Server seines Teilmarktes und erhält daraufhin eine Folge von Aktualisierungsoperationen (*add/update/delete*), die ihn zum Laden der erforderlichen Java Klassen veranlassen, die die Caching-Funktionalität implementieren. Dies ist der Code für den *Update*-Server selbst

3. Der Cache horcht nun auf seinem konfiguriertem Netzwerk-Port und nimmt Anfragen nach Web-Dokumenten entgegen, die er zunächst nicht aus seinem lokalen Zwischenspeicher erfüllen kann. Diese Anfragen reicht er an die ihm übergeordneten *Update*-Server oder, falls er der Haupt-Cache ist, an den Ursprungs-Server weiter.

4. Jedes neu (vom übergeordneten Cache oder Ursprungs-Server) angefordertes Web-Dokument wird im lokalen Zwischenspeicher abgelegt und als neuer Bestandteil der Cache-Anwendung beim *Update*-Server angemeldet. Damit wird es zu einem Code-Objekt und kann über die `validate()` Methode überprüft werden.

5. Der Hauptsteuerfluß der Caching-Anwendung verfährt mit neuen Dokumentenanforderungen wie folgt

   (a) Ein frisches Objekt wird direkt aus dem Zwischenspeicher ausgeliefert

   (b) Ein gültiges Objekt wird nach seiner Validierung (beim *Update*-Server) aus dem Zwischenspeicher ausgeliefert

(c) Ein ungültiges oder nicht existentes Objekt wird über den *Update*-Server angefordert

6. Gelangt die Cache-Anwendung an ihre Maximalkapazität, reserviert sie zwei neue *Processors* als weitere Caches und propagiert die URLs der beiden Server, um tatsächlich Lastverteilung zu erreichen.

6.4 Client-Server Tests

Viele Firmen entwickeln komplexe Client-Server Software für eine Reihe leistungshungriger Anwendungen wie zum Beispiel WWW-, CORBA-, *Audio/Video on demand*- oder Datenbank-Server, die für eine sehr hohe Zahl von Benutzern ausgelegt sind. Ein erschöpfender Test dieser Anwendungen ist mangels ausreichender infrastruktureller und menschlicher Ressourcen nicht möglich und wird daher meist mit Skripten oder teurer Spezialsoftware simuliert. Die Locust-Infrastruktur ermöglicht dagegen Tests unter Real-Bedingungen mit hunderttausenden verschiedener Anwender, die darüberhinaus noch spezielle Profile erfüllen. Diese Profile können Plattform, Betriebssystem, Leistung, Ort, Uhrzeit oder sogar den Browsertyp und -version des *Web*-Terminals umfassen.

6.4.1 Erzeugung von Server-Last

Definition 6.5 (Server-Last) *Unter Server-Last wird im folgenden die Last verstanden, die durch die Beantwortung von Client Anfragen an einen Server im einem Client-Server-System erzeugt wird.*

Es existieren mehrere Möglichkeiten auf einem oder mehreren Servern Server-Last zu Testzwecken, also parametrisiert, zu erzeugen:

1. Fernsteuerung (Remote Control) – Die einfachste Möglichkeit, Server-Last zu erzeugen ist die Fernsteuerung der Oberfläche des Clients, das auf den Server zugreift, durch ein Skript.

   Für jeden zu simulierenden Benutzer muß eine eigene Client-Anwendung ausgeführt und ferngesteuert werden, eine Tatsache, die diesem Ansatz ressourcenintensiv und nicht sehr skalierbar macht, denn es lassen sich nicht mehr Benutzer simulieren als Hardware und andere, insbesondere Netzwerkressourcen für sie zur Verfügung stehen. Allerdings sind zur Fernsteuerung keine Kenntnisse interner Sitzungdetails zur Parametrisierung notwendig.

2. Sitzungsaufzeichnung (Session recording) – Durch Aufzeichnung einer Sitzung eines Clients zum Server inklusive aller Benutzereingaben und des Netzverkehrs, und anschließendes Abspielen lassen sich Client-Sitzungen sehr einfach simulieren. Voraussetzung zur Simulation *verschiedener* Benutzer ist, das die aufgezeichnete Sitzung parametrisierbar ist, daß also die clientabhängigen Parameter identifiziert und ersetzt werden können.

   Vorteil dieses Ansatzes ist sein geringerer Ressourcenbedarf. Da die Client Anwendung nicht mehr selbst, sondern nur noch die Client-Logik und der Netzwerkverkehr

ausgeführt werden muß, können über Sitzungsaufzeichnung mehr Benutzer simuliert werden, als Ressourcen für sie zur Verfügung stehen. Dennoch ist auch diese Methode nicht unbegrenzt skalierbar, sondern durch die lokal vorhanden Hardware und Netzwerkbandbreite beschränkt.

Die meisten kommerziell erhältlichen Systeme zum Test von Client-Server Systemen verwenden das letzte Verfahren.

6.4.2 Software für Client-Server Systeme

In folgenden Abschnitt werden anhand der Software *LoadRunner* von Mercury Interactive die Funktionsweise, Anwendung und Nachteile typischer Testsoftware für Client-Server Systeme beschrieben.

LoadRunner

Die Software *LoadRunner* wird derzeit von der Firma Mercury Interactive [Int] in der Version 6.0 ausgeliefert. Die Software besteht aus mehreren Komponenten, einer Steuerkomponente (*Controller*), einem Benutzergenerator, mehreren Agenten sowie einer Analysekomponente. Der Controller dient als zentrales Steuerelement zur Aufzeichnung, Parametrisierung und Ausführung von Last-Tests während die Agenten die parametrisierten Aufzeichnungen (*Skripte*) "abspielen".

1. Steuerkomponente – Die Steuerkomponente ist das zentrale Steuerelement zur Aufzeichnung, Parametrisierung und Ausführung von Last-Tests.

2. Benutzergenerator – Der Benutzergenerator ist ein Teil der Steuerkomponente, die Benutzeraktivitäten aufzeichnet und daraus abspielbare Sitzungen erzeugt. Die Aufzeichnung erfolgt auf Protokollebene durch Erkennung, Aufzeichnung und eventuell Modifikation bzw. Parametrisierung von Protokoll-Events. Das heißt, der Benutzergenerator kann zwar beliebige (Netzwerk-)Sitzungen zwischen Client und Server aufzeichnen, eine Erkennung und Parametrisierung von einzelnen *Events* ist jedoch nur möglich, falls der Generator die jeweiligen Protokolle "kennt". Ansonsten wird die aufgezeichnete Sitzung einfach als Byte-Strom wieder abgespielt. Der LoadRunner untersützt folgende Protokolle

   (a) Microsoft Database Protocol

   (b) Hyper Text Transport Protocol HTTP

   (c) Java Remote Methode Invocation RMI

   (d) Internet Inter ORB Protocol IIOP

   Die anschließende Parametrisierung der *Events* erfolgt manuell, da die Software die wechselseitigen Abhängigkeiten der Events nicht erkennen und zusammenfassen kann. Dies erfolgt in der Regel über eine grafische Benutzeroberfläche durch den Administrator. Darüberhinaus ist noch ein ausgiebiger Test und eine Nachbearbeitung der parametrisierten Test-Skripten notwendig. Insbesondere *Hand-shake* Szenarien zwischen Client und Server erfordern viel manuelle Optimierung der Skripte.

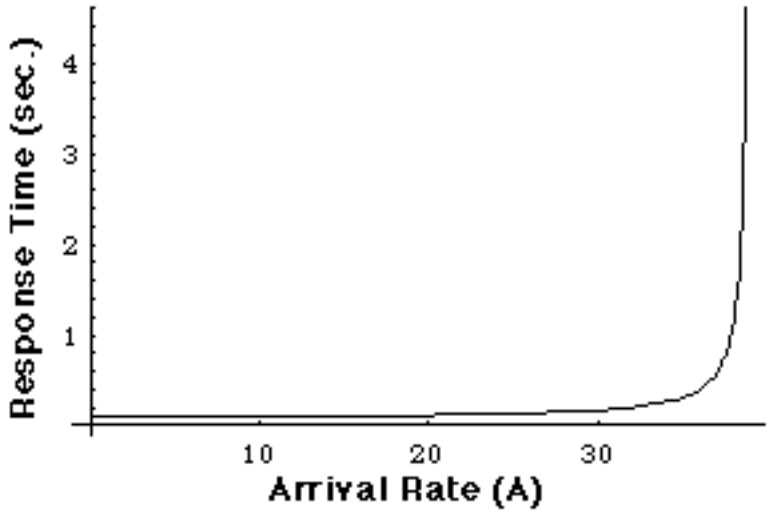

Abbildung 6.9 Antwortzeit eines Warteschlangen-Modells

3. Agenten – Verteilte, von der Steuerkomponente gesteuerte Skripte, die, mehrfach parametrisiert und wiederholt abgespielt, die gewünschte Last erzeugen. Durch Verteilung und Ausführung der parametrisierten Skripte entstehen *virtuelle* Benutzer. Ihre Anzahl sowie ihre virtuelle Netzanbindung (Modem, ISDN, 10 MBit, 100 MBit) kann in einem Profil festgelegt werden. In einem solchen Profil können auch Wartepunkte enthalten sein, an denen Agenten synchronisiert und gleichzeitig Last erzeugen können.

4. Analysekomponente – Komponente, die die auf dem oder den Servern simulierte Last aufzeichnet und analysiert. Zuvor werden die zu erfassenden Last-Parameter vom Administrator über die Steuerkomponente festgelegt.

6.4.3 Modellierung von Server-Last

Server bearbeiten in der Regel viele nebenläufige Aufgaben wie zum Beispiel Dateianforderungen, die jeweils ein oder mehrere exklusive Ressourcen wie Prozessorzeit, Dateisystem-Zugriffe oder Netzwerk Bandbreite benötigen. Da auf exklusive Ressourcen nicht parallel zugegriffen werden kann, müssen nebenläufige Ressourcen-Anforderungen vom Server serialisiert werden. Dies erfolgt meist mit Hilfe einer Warteschlange, in die zu erledigende Anfragen eingetragen und nach Erledigung wieder ausgetragen werden. Zur Modellierung solcher serialisierter Server-Dienste kann die Warteschlangen-Theorie eingesetzt werden.

Warteschlangen-Theorie Die Warteschlangen-Theorie betrachtet jeden Dienst und jede Ressource als abstrakte Warteschlange, die in einen oder mehrere Server mündet. Eine abstrakte Warteschlange ist durch folgende Parameter gekennzeichnet: die durchschnittliche Ankunftsrate von Anfragen A, die durchschnittliche Bearbeitungszeit T_s sowie die

durchschnittliche Verweilzeit einer Anfrage in der Warteschlange T_q. Aus diesen Parametern ergibt sich die durchschnittliche Antwortzeit eines Systems aus $T = T_s + T_q$ [P.S96].

Definition 6.6 (Stabiles System) *Ist die Ankunftsrate A kleiner als die Bearbeitungsrate $A < \frac{1}{T_s}$, wird das Warteschlangen-System als stabil bezeichnet.*

Definition 6.7 (Instabiles System) *Ist die Ankunftsrate A grösser oder gleich der Bearbeitungsrate $A >= \frac{1}{T_s}$, wird das Warteschlangen-System dagegen als instabil bezeichnet.*

Aus dem Produkt $U = AT_s$ aus Ankunftsrate und Bearbeitungszeit ergibt sich die durchschnittliche Auslastung U des Servers, die im Bereich zwischen 0 und 1 liegt. Eine Auslastung von 0 erfaßt einen untätigen, ein Wert von 1 dagegen bezeichnet einen unendlich ausgelasteten, also blockierten Server.

Falls der Zeitraum zwischen einzelnen Anfragen zufällig und nicht vorhersehbar ist, folgt die Ankunftsrate einer exponentiellen Verteilung. Eine solche Verteilung ist für die Modellierung von Warteschlangen von großer Wichtigkeit, denn die zufällige Verteilung von Anfragen bedeutet, daß der augenblickliche Zustand eines Warteschlangen-Systems für das zukünftige Verhalten irrelevant ist und in der weiteren Simulation vernachlässigt werden kann, was die Komplexität der Modellierung stark vereinfacht.
Ein solches günstiges Warteschlangen-System mit exponentiell verteilter, *zustandsloser* Ankunftsrate und Bearbeitungszeit wird als M/M/c Warteschlange bezeichnet, wobei der Buchstabe M für eine Markov- bzw. zustandslose Modellierung der Ankunfts- und Bearbeitungsrate steht und c die Anzahl von Servern bezeichnet. Die durchschnittliche Antwortzeit eines M/M/1 Systems ergibt sich aus

$$T = \frac{T_s}{(1 - U)} \tag{6.4.1}$$

Abbildung 6.9 veranschaulicht die Antwortzeit einer M/M/1 Warteschlange in Abhängigkeit der Ankunftsrate. Bei Auslastung 0 entspricht die Reaktion- der Bearbeitungszeit, da keine Warteschlangen-Zeit auftritt. Mit zunehmender Auslastung steigt die Antwortzeit langsam an, bis sie bei asymptotischer Annäherung an die Auslastung 1 gegen Unendlich strebt.

Das Gesetz von Little [P.S96] bezeichnet die durchschnittliche Anzahl in einer Warteschlange wartenden Anfragen mit $N + AT$ und ist auf alle stabilen und konservativen – das heißt es geht keine Anfrage in der Schlange verloren – Warteschlangen-Modelle anwendbar.

Gewöhnlicherweise können komplexe Client-Server Szenarien nicht mit einer einzigen Warteschlange modelliert werden. Vielmehr werden eine Reihe von Schlangen zu einem komplexeren System oder Netzwerk zusammengeschaltet und in einem Graph wie in Abbildung 6.10 als Knoten dargestellt. Solche Netzwerke werden als *offen* bezeichnet, falls neue Anfragen das System von außen betreten und nach außen wieder verlasen können. Laut dem Theorem von Jackson [P.S96] kann in einem offenen System ein Großteil der Komplexität der Interaktionen vernachlässigt werden, selbst dann, wenn die Ankunftsverteilung nicht mehr exponentiell ist, weil sie vom Rest des Netzwerkes abhängt.

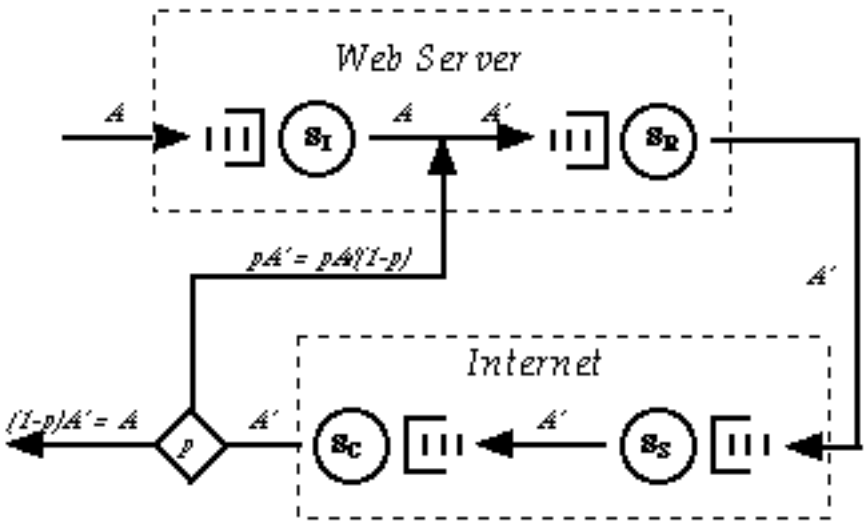

Abbildung 6.10 Warteschlangen-Modell eines Web Servers

Warteschlangen-Modellierung eines Servers Im folgenden wird eine einfache, abstrakte Modellierung eines Web Servers vorgestellt, einem einfachen, über das Internet verbundenen Client-Server System. Das in Abbildung 6.10 abgebildete offene Warteschlangen-Netzwerk ist ein weithin anwendbares, einfaches Modell, das alle Hard- und Software-Details abstrahiert, aber dennoch detailliert genug ist, um aussagekräftige Ergebnisse in Hinsicht auf Server- und Netzwerk-Geschwindigkeit zu liefern. Weitere Vereinfachungen sind die Vernachlässigung von Protokolldetails von HTTP und TCP/IP sowie die Beschränkung auf statische Web Inhalte.

Das in Abbildung 6.10 abgebildete Modell besteht aus vier Knoten (die wiederum aus einer Warteschlange bestehen), von denen zwei den Server selbst und die übrigen zwei das Weitverkehrsnetz modellieren. Anfragen nach Dateien erreichen den Server vom Netzwerk mit der Ankunftsrate A. Der Knoten S_I führt die Initialisierung des Servers aus und gibt die Anfrage an den Knoten S_R zur Bearbeitung weiter. Nach der Bearbeitung der Anfrage wird das Ergebnis an den Knoten S_S übergeben, der die Transferrate (zum Beispiel die 1,5 MBit einer T1 Leitung) des Netzwerkes modelliert. Die Daten gelangen nun über das Netz zum Knoten S_C, der den Client modelliert. Falls die Anfrage nicht vollständig beantwortet wurde, müssen weitere Datenblöcke vom Knoten S_R versendet werden, sonst ist die Anfrage bearbeitet und verläßt das Modell über Verzweigung p. Weitere Anfragen des Clients gelangen nun unter Umgehung der Initialisierung direkt zum Knoten S_R.

Verzweigung p ist wahrscheinlichkeitsverteilt: bei durchschnittlicher Dateigröße von F und Blockgröße von B beträgt die Wahrscheinlichkeit einer vollständigen Bearbeitung $p = \frac{B}{F}$. Die Anzahl der Anfragen an den Knoten S_R ergibt sich aus der Summe neuer Aufträge und der Anzahl der von S_C nach S_R zurückfließenden Anforderungen. Damit ist die Anzahl der Antworten, die ein stabiles System verlassen, gleich der Zahl der Anforderungen.

Als Jackson Netzwerk betrachtet, ergibt sich die Antwortzeit des Servers zu

$$T = \frac{F}{C} + \frac{I}{1 - AI} + \frac{F}{S - AF} + \frac{F(B + RY)}{BR_A F(B + RY)} \tag{6.4.2}$$

mit folgenden Parametern:

A – Die Ankunftsrate der Anfragen an das System: Die durchschnittliche Anzahl der Anfragen an den Server in einer Sekunde.

F – Die durchschnittliche Dateigröße einer Antwort auf eine Anfrage durch den Server.

B – Die Blockgröße des Servers: Die Größe der Dateifragmente, die über das Netzwerk verschickt werden und meist der Blockgröße des Server-Dateisystems entsprechen, spielt bei der allgemeinen Server-Leistung eine untergeordnete Rolle.

I, Y, R – Die Initialisierungszeit, die statische und die dynamische Bearbeitungszeit des Servers: I repräsentiert die durchschnittliche Zeit um einmalige Initialisierungsarbeiten durchzuführen. Y modelliert die Bearbeitungszeit eines Blocks unabhängig von der Blockgröße während R die Datenrate (Bytes/Sek.) repräsentiert, mit der der Server Blöcke bearbeiten kann. Die Verarbeitungsgeschwindigkeit heutiger Prozessoren übertrifft die Bandbreite aktueller Netzwerktechnologien meist um ein Vielfaches.

S, C – Die Netzwerkbandbreite des Servers und des Clients: Beide Werte gemeinsam bilden die Übertragungsgeschwindigkeit des Verbindungsnetzwerkes. Typische Werte für S sind 64 KBit (ISDN), 1,5 MBit (T1) und 6 MBit (T3). C repräsentiert die oft weitaus niedrigere Durchschnittsgeschwindigkeit, mit der der Client die Server-Antworten empfängt. Bei einer kürzlichen Untersuchung [P.S96] ergab sich ein Mittelwert von 707 KBit/Sek.

Modell-Vereinfachungen Mit Hilfe des angegebenen Warteschlangen-Modells lassen sich die Antwortzeiten eines Web-Servers in Abhängigkeit verschiedener Parameter vorhersagen. Zur Vereinfachung der Vorhersage müssen aber zunächst einige für die Simulation weniger wichtige Parameter identifiziert werden, um das Modell zu vereinfachen. Die Blockgröße B des Servers ist vernachlässigbar und wurde fortan auf 2000 Bytes gesetzt. Weiterhin hat die Netzwerk-Geschwindigkeit des Clients C wenig Einfluß auf die Antwortzeit und gar keinen Einfluß auf die Maximalkapazität. Da dieser Wert vom Server sowieso nicht beeinflußt werden kann, wurde er bei den folgenden Studien auf einen Durchschnittswert (707 KBit) gesetzt. Der Einfluß der Initialisierungs- und statischen Bearbeitungszeit, I und Y, kann über die dynamische Bearbeitungszeit R mitmodelliert werden, das heißt $I = Y = 0$ und R repräsentiert die Server-Geschwindigkeit. Die übrigen Modellparameter, F, R und S beeinflußen die Reaktionsgeschwindigkeit und Maximalkapazität sehr stark und können nicht vernachlässigt werden.

6.4.4 Ergebnisse

Modell-Ergebnisse Bei der in Abbildung 6.9 abgebildeten Simulation der Reaktionszeiten T und Maximalkapazitäten M werden einige interessante Ergebnisse offensichtlich.

Während die Auswirkung von F auf T und M stets signifikant ist, hängen die Auswirkungen von R und S davon ab, ob die Systemleistung von der Netzbandbreite oder der Verarbeitungsgeschwindigkeit des Servers begrenzt ist. Weil moderne Server Daten mit 10 MBit und mehr, also deutlich schneller als die typischen Client-Geschwindigkeiten (von 28 KBit über IDSN bis T1 und T3), liefern können, ist fast immer das Netzwerk der begrenzende Faktor. In diesem Fall hängen T und M ausschließlich von der Netzwerkgeschwindigkeit S und der durchschnittlichen Dateigröße F ab. Die Server-Geschwindigkeit R ist dagegen nicht von Belang.

Nur sehr populäre Web-Sites verfügen über so hohe Netzwerkbandbreiten, daß der Server zum Flaschenhals wird. In diesen Fällen hängt die Antwortzeit T und Maximalkapazität M nur von R und F ab, während S ignoriert werden kann. Weiterhin hat die durchschnittliche Dateigröße einen signifikanten Einfluß auf die Antwortzeit T und, in besonderen Maß, auf die Maximalkapazität M. Es ist naheliegend, daß die Erhöhung von F zu niedrigeren Maximalkapazitäten führt. Überraschend ist jedoch, daß dies nicht linear und vorhersehbar geschieht, sondern sehr sprunghaft: schon eine sehr geringe Vergrösserung von F kann zu einer starken Verringerung von M und damit zu einem Überlastungszustand führen.

Wie in Abbildung 6.9 illustriert, steigt die Antwortzeit eines (Web) Servers mit zunehmender Last kaum wahrnehmbar bis zu einem Schwellwert an. Danach strebt die Reaktionszeit sehr schnell und asymptotisch gegen Unendlich. Die Asymptote definiert die Kapazitätsgrenze von Servern und widerspricht gängigen Annahmen, daß die maximale Verarbeitungskapazität von Servern nicht beschränkt ist, das heißt, daß alle Anforderungen letztendlich bearbeitet werden, nur mit immer deutlicherer Verzögerung.

Wenn sich die Server-Last der Asymptote nähert, können sehr geringe Lasterhöhungen den Server in einen verklemmungsähnlichen Zustand (*Deadlock*) versetzen, in dem immer mehr Anforderungen mit immer geringerer Geschwindigkeit bearbeitet werden, so daß keine oder nur sehr wenige Antworten generiert werden können. Beispielsweise verkrafte eine Server problemlos n Anfragen pro Tag mit Reaktionszeiten, die 50% unter nicht mehr akzeptablen Bedingungen liegen. Die Server-Last steige weiterhin um 2% (gemessen in n) pro Tag. Nach üblicher Auffassung dürfte es annähernd ein Jahr dauern, bis die Reaktionszeiten des Servern nicht mehr akzeptabel sind. Falls jedoch der Server bereits nahe an seiner Kapazitätsgrenze ist, können die Reaktionszeiten bereits nach wenigen Tagen sprunghaft ansteigen, so sehr, daß der Eindruck einer Verklemmung entsteht.

Vermeidung von Verklemmungen Extrem lange Reaktionszeiten, die dem Benutzer wie Verklemmungen erscheinen, treten immer dann auf, wenn Anfragen gleich schnell oder schneller eintreffen, als sie beantwortet werden können. Der einzige Weg, die Überlastung zu vermeiden, ist, die Aufnahme neuer Aufträge in die Warteschlange zu verzögern oder zu beenden. Das Problem hierbei ist jedoch, daß die wenigsten Server die Ankunftsrate A überwachen, so daß dieser Parameter unbekannt ist. Allerings entspricht die Ankunftsrate der Anzahl offener Netzwerkverbindungen, die als Indikator für eine Überlastsituation eingesetzt werden kann.

Falls ein (Web-) Server sich einem definierten Schwellwert offener Verbindungen nähert, sollten neu eintreffende Anfragen mit einer entsprechenden Fehlermeldung (*HTTP: 501 come back later*) beantwortet werden und die Anforderungen in der Warte-

schlange priorisiert abgearbeitet werden. Der Client-Browser sollte als Reaktion auf diese Fehlermeldung nach einigen Sekunden eine erneute Anfrage absetzen, wenn der Server weniger ausgelastet ist. Leider unterstützen derzeit die wenigsten Server diese Meldung und kein aktueller Browser eine automatische Wiederholung der Anforderung.
Die Mehrheit heutiger UNIX Server unterstützt eine hohe Zahl gleichzeitig offener Netzwerkverbindungen und ist daher für Verklemmungen sehr anfällig, ganz im Gegensatz zur Macintosh- und Windows-Plattformen, die die Anzahl gleichzeitiger Verbindungen stark beschränken. Eine Server-Verklemmung aufgrund hoher Last ist auf diesen Plattformen also ausgeschlossen.
Eine Verklemmung läßt sich also auf UNIX-Plattformen nur vermeiden, falls die Anzahl offener Netzwerkverbindungen entweder über das Betriebssystem oder die Server-Software beschränkt wird. Unter den Macintosh- und Windows-Betriebssystemen ist dies oft bereits standardmäßig der Fall.

Verbesserung der Server-Leistung Von den in vorangegangenen Abschnitt diskutierten Möglichkeiten, die Server-Leistung unter hoher Last zu verbessern, ist die Erhöhung der Netzwerkbandbreite die beste Verbesserungsmöglichkeit, falls die Bandbreite den begrenzende Faktor des Gesamtsystems darstellt. Stellt sich die Verarbeitungsgeschwindkeit des Servers als Engpaß des Systems dar, existieren mehrere Möglichkeiten der Verbesserung der Leistung.

Die Ergebnisse der Modellierung mehrerer geclusterter Server brachte ebenfalls erstaunliche Ergebnisse. Hierzu muß zunächst dem Warteschlangen-Modell eine Auswahlmöglichkeit hinzugefügt werden. Eingehende Anfragen werden nun mit der Wahrscheinlichkeit p an den Knoten 1 und der Wahrscheinlichkeit $(1-p)$ an den Knoten 2 gerichtet. Wieder muß nach dem begrenzenden Faktor des Gesamtsystems gefragt werden.
Wie zu erwarten, beschleunigt eine Erhöhung der verfügbaren Bandbreite den Server am meisten, falls dies den Engpaß des Systems darstellt. Die Erhöhung der Server-Geschwindigkeit sowie das Hinzufügen eines zweiten, identischen Servers ist nur marginal meßbar. Es zeigt sich, daß die Lastverteilung eine dominante Rolle bei der Optimierung der Leistung spielte. Ein Zwei-Server-Cluster, bei dem ein Server schwächer als der andere ist, erreicht einen Verklemmungs-Zustand früher als der schnellere Server allein.
Ist die Gesamtleistung durch die Server-Geschwindigkeit begrenzt, ist naturgemäß die Erhöhung der Server-Leistung die effizienteste Leistungssteigerung. Noch vor der Addition eines identischen, zweiten Servers rangiert die Erhöhung der Netzwerkbandbreite, solange sich die Ankunftsrate noch unter der Maximalkapazität bewegt. Wie auch schon im vorhergehenden Fall reduziert eine zweiter, schwächerer Server die Gesamtleistung.

6.4.5 Client-Server-Tests als Locust-Anwendung

Aus der Erkenntnis, daß die Leistung von Web-Servern praktisch immer von der verfügbaren Netzwerk-Geschwindigkeit begrenzt wird, ergeben sich für diese Arbeit zwei Folgerungen.
Zunächst kann festgehalten werden, daß einer möglichen Überlast des Locust-Servers sehr oft durch bloße Erhöhung der Bandbreite der Serveranbindung begegnet werden kann, falls die Netzwerkbandbreite den Engpaß des Gesamtsystems darstellt. Selbst wenn die maxi-

male Verarbeitungskapazität des Locust-Servers erreicht ist, kann dessen Leistung durch Clustering mit weiteren, gleich starken Locust-Servern dennoch weiter erhöht werden. Hierzu werden überlastete (Teil-)Märkte mit ihrem Locust-Server in zwei oder mehr neue Teilmärkte mit eigenen Serverdiensten aufgeteilt und über einen hierarchisch übergeordneten Locust-Server (der die Funktion eines inversen Caches übernimmt) gekoppelt. Eine tiefer gehende Beschäftigung mit der Thematik der serverseitigen Lastverteilung ist im Rahmen dieser Arbeit demnach nicht notwendig. Die clientseitige Lastverteilung erfolgt, wie in Kapitel beschrieben, durch die mehrfache Vergabe von Jobs an Web-Terminals (*Eager Scheduling*).

Zweitens verbleibt zu bemerken, daß der für die Server-Leistung so wichtige Parameter Netzwerkgeschwindigkeit bei der Simulation von Server-Last völlig außen vor bleibt. Das Testen von Client-Server-Umgebungen durch das Aufzeichnen und Abspielen von Lastprofilen kommt aber auch aus anderen Gründen nicht an reale Tests heran. Als Alternative bieten sich daher realistische Testbedingungen, die mit Hilfe einer Locust-Anwendung realisiert werden können. Solche Anwendungen richten teilnehmende *Web*-Terminals und dedizierte *Processors* koordiniert auf den zu testenden Serverdienst, um die relevanten Parameter seiner Kapazität, wie maximale Ankunftsrate und Maximalkapazität, zu ermitteln. Ein hoher Multiplikationsfaktor der Locust-Infrastruktur sorgt dabei für eine hohe Zahl an Teilnehmern und damit hohe reale Last während die Lokalität innerhalb eines Teilmarkts verwendet werden kann, falls eine zeitlich und örtlich sehr feine Koordination beim Testen benötigt wird, die sich über den gesamten Locust-Markt nicht aufrecht erhalten läßt. Insgesamt bieten sich eine Fülle neuartiger Anwendungen in diesem Bereich.

6.5 Web Object Computing Anwendungen

Verwendet man Java-basierte Web-Computing-Infrastrukturen als Substrat zur Ausführung von in Java-Applets eingebetteten CORBA [Gro92] Objekten, entsteht eine Programmiermodell, das als *Distributed Object Computing* (DOC) bezeichnet wird. DOC wird als die objektorientierte nächste Generation des noch Datei-orientierten World Wide Webs (WWW) angesehen. Als Substrat für DOC wird eine Web-Computing Infrastruktur wie Locust als Transport- und Verteilungssystem für verteilte CORBA Objekte verwendet, die über HTTP ge-tunneltes IIOP (*Internet Inter ORB Protocol*) miteinander in Verbindung stehen. Zur Abschöpfung ungenutzter, brachliegender lokaler Rechenleistung unterstützt Locust wie im folgenden Abschnitt beschrieben neben Java Applets und Beans auch in Java Applets eingebettete CORBA Objekte [May99d].

6.5.1 Das Object Web

The next shift catalyzed by the Web will be the adoption of enterprise systems based on distributed objects and IIOP (Internet Inter-Orb Protocol). IIOP will manage the communication between the object components that power the system. Users will be pointing and clicking at object available on IIOP-enabled servers. We expect to distribute 20 million IIOP clients over the next 12 months and millions of IIOP-based servers over the next couple of years. We'll put the platform out there so that people can start developing for it.
Marc Andreesen, Netscape Inc. [OH97]

Die Evolution des World Wide Webs

Phase Eins Die erste Phase des WWW wurde zu Anfang der neunziger Jahre von Tim Berners-Lee eingeleitet, der die Protokolle und Technologien vorschlug, die ein unidirektionales Medium zur Veröffentlichung statischer elektronischer Dokumente bestimmten. Diese auf dem Hyper Text Transport Protokoll (HTTP) aufsetzende Client-Server *Middleware* etablierte das World Wide Web als riesigen URL-basierten (*Uniform Resource Locator*) Datei-Server, um statische HTML Seiten zur Verfügung zu stellen, wie in Abbildung 6.11 illustriert.

Phase Zwei In seiner zweiten Phase entwickelte sich das WWW Mitte der Neunziger mit Hilfe einer dreischichtigen Client-Server-Architektur in ein interaktiveres Medium als bisher. Das sogenannte *Common Gateway Interface* (CGI) [Rob] erlaubte Web-Servern spezifische serverseitige Anwendungen in einem eigenen Prozeß auszuführen, um dynamisch HTML Seiten, weitere Programme oder Daten zu manipulieren. Eingaben wurden, wie in Abbildung 6.12 angedeutet, hauptsächlich mit Hilfe von HTML Formularen an die serverseitigen Anwendungen zugestellt. In gewissem Sinn stellt CGI die einzige allgegenwärtige Standard-Schnittstelle zum Zugriff auf entfernte Rechenressourcen im WWW dar, mit einer ähnlichen Abstraktion und Homogenität anderer Standarddienste wie Speicherplatz (über den HTTP Dämon `httpd`), Email- oder Namensdienste (über `sendmail` und `bind`), auf die mit der allgegenwärtigen Client-Software, dem Web Browser, zugegriffen wird.

CGI ist ein langsames und statusloses – aber Plattform-unabhängiges – Modell, so daß es frühzeitig verschiedene Anstrengungen gab, diese Einschränkungen zu überwinden. Beispiele für derartige leistungsfähige, jedoch zumeist proprietäre Erweiterungen und APIs (*Applications Programmers Interface*) sind Cookies, FastCGI [Mar00], VBScript [Mic99b] und Active Server Pages (ASP) [Mic00a] von Microsoft, WebObjects von Next (jetzt Apple) [App00], Java Servlets von Sun [AG96], JavaScript von Netscape [Fla98] und Dynamic HTML (DHTML). Hauptziel dieser Technologien ist die Überwindung der Statuslosigkeit von HTTP und CGI, um statische Seiten und Skripte in Objekte verwandeln zu können. Der schwerwiegendste Nachteil solcher Ansätze, ist, daß der Web Server, oftmals wiederum in einem eigenen Prozeß, zwischen den Client- und den Server-Instanzen dieser Objekte, mittels HTTP vermitteln muß. Client Anwendungen können Server-Instanzen bzw. -Methoden nicht direkt manipulieren bzw. aufrufen. Es wird daher ein abstrakteres Modell der Web-Architektur benötigt, das *Object Web*.

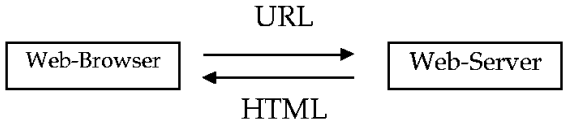

Abbildung 6.11 Erste statische und Datei-zentrierte Phase des Webs

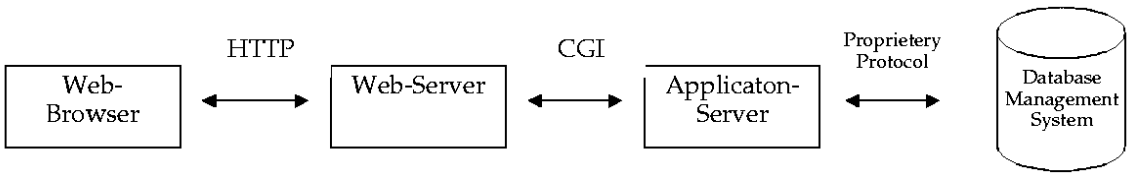

Abbildung 6.12 Interaktive 3-Schichten Web-Architektur der Phase 2

Phase Drei Allgemein wird als nächste Phase, die den Umgang mit dem WWW entscheidend verändern wird, das *Object Web* [OH97] angesehen. *Object Request Broker* (ORBs) stehen bereit, um den CGI Flaschenhals zu lösen, da sie entwickelt wurden, um Grenzen, die durch verschiedene Plattformen und Programmiersprachen von Client-Server-Architekturen entstanden sind, zu überbrücken, wie zum Beispiel die allgegenwärtigen Client-Browser sowie die Web- und Applikations-Server. CORBA bietet weitaus mehr Vorteile als nur die inter-operable Multi-Plattform-ORBs. Es stellt darüber hinaus eine Vielzahl verteilter Dienste wie zur Verfügung wie dynamische Entdeckung, Selbsprüfung, globale Objektreferenzen und Namensdienste, *Garbage Collection* sowie grundlegende Sicherheitsaspekte, die die Grundlage sehr interaktiver, komplexer, verteilter Web-Anwendungen bilden.

Object Web Architektur Zur Benutzung von DOC Anwendung ist clientseitig ein ORB notwendig, was die Verbreitung und Anwendbarkeit solcher Anwendungen scheinbar stark einschränkt. Tatsächlich jedoch ist eine Großteil der für DOC notwendigen Infrastruktur bereits mit Auslieferung des Communicator 4 gelegt. Dieser Browser wurde von Netscape mit dem VisiBroker CORBA/Java ORB von Visigenic ausgeliefert [OH97], so daß nun ungefähr ein Drittel aller Web-Browser der vierten Generation bereit für das Object Web sind. Auf Serverseite steht der Enterprise Server von Netscape zur Verfügung, der seit Version 3.0 zusammen mit C++- und Java-basierten CORBA ORBs ausgeliefert wird.

Ein typisches Szenario interagierender Web-Objekte könnte, wie in Abbildung 6.13 illustriert, folgendermaßen aussehen:

1. Der Browser fordert (via HTTP) eine HTML Seite an, die ein Java-Applet oder -Bean enthält

2. Der HTTP Server liefert die angeforderte Seite und das enthaltene Applet (via HTTP) zurück

3. Der Browser führt das Applet aus

4. Das Applet kontaktiert den Web-Objekt *Adapter* – entweder ein CORBA Server, der IIOP Anforderungen entgegen nimmt, oder ein ORB – um CORBA Server-Objekte der Server-Anwendung zu instantiieren, die wiederum (via IIOP) Datenbanken (DBMS) oder HTML Seiten manipulieren. Der Zugriff und die Ausführung der Server-Objekte erfolgt über im Applet enthaltenen IDL-generierte Client-*Stubs*[1].

[1]Falls das IIOP Protokoll nicht durch HTTP ge-tunnelt wird, ist zur Kommunikation eine IIOP-sensibler *Fire-Wall* notwendig

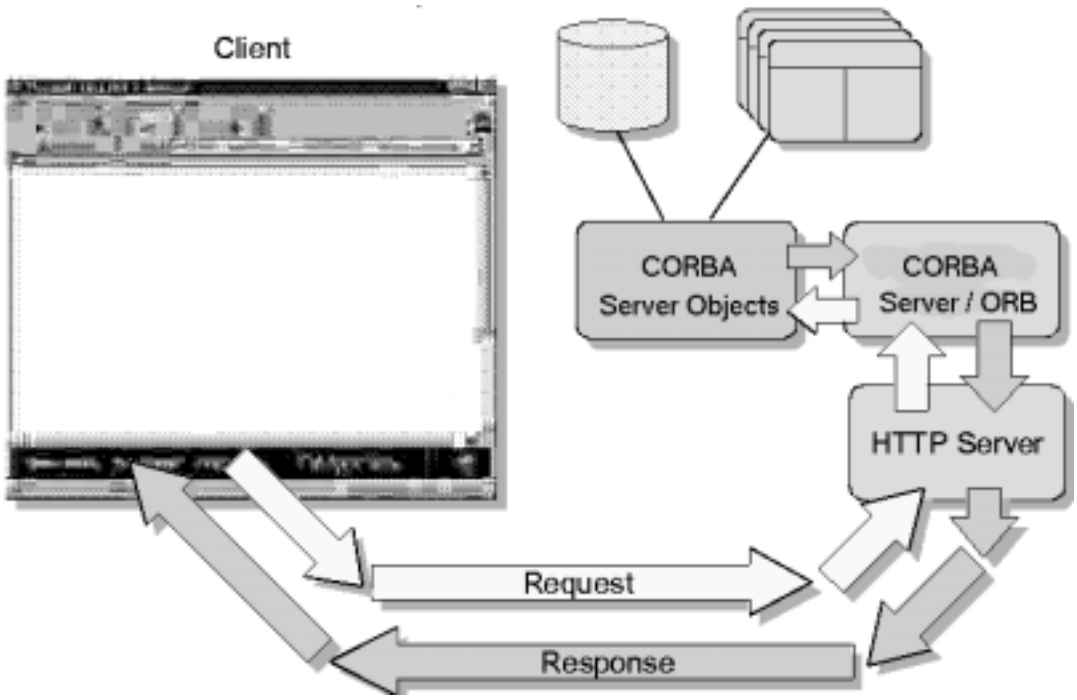

Abbildung 6.13 Interaktion im Object Web

Zusätzlich kann das Server-Objekt auch noch die nächste HTML-Seite des Clients generieren, falls das Java-Applet keine Server-Interaktion unterstützt oder das Server-Objekt sein Eingaben von einem HTML-Formular erwartet.

Vorteile des Object Web CORBA/IIOP lindert den Client-Server-*Overhead* beachtlich, der mit dem klassischen CGI-Flaschenhals einhergeht. Der Client kann über IIOP typisierte Parameter anstelle bloßer *Strings* an den Server übergeben, kann direkt Methoden anstelle eines neuen Prozesses auf dem Server ausführen und schließlich mit persistenten Objekten interagieren anstatt fortlaufend serverseitige Prozesse zu erzeugen. CORBA/IIOP ist im Gegensatz zum Instanzen-zentrierten CGI Modell, das sich auf einzelne, spezifische Anwendungen auf einem bestimmten Server konzentriert, ein hoch-skalierendes verteiltes Rahmenwerk. Dem Client-Applet steht es frei, jeden Server zu kontaktieren, der einen bestimmten Dienst anbietet, anstatt nur eines Einzigen und der ORB ist in der Lage, diese Anfragen auf eine Gruppe von Servern gleichmäßig zu verteilen anstatt einen Einzigen mit Anfragen zu belegen.

Web-Computing mit verteilten Objekten Die Migration von dateizentrierter Client-Server-Interaktion mit CGI, ASP und anderen Erweiterungen des statuslosen HTTP Protokolls in Richtung auf objektorientiertes, verteiltes Client-Server Rechnen, eröffnet Java und Web-Computing-Infrastrukturen wie Locust ganz neue Anwendungsfelder. Anwendungen für solche Rahmenwerke waren in der Vergangenheit auf parallele grob-granulare oder *Brute-Force* Anwendungen mit einem *Master-Slave*-Programmiermodell beschränkt. Hierbei zerlegt der *Master* die Aufgabe in Teilprobleme, die er an die *Slaves* verteilt und

deren Berechnungsfortschritt er verwaltet. Ein weiteres Problem des Web-Computings waren die Sicherheitseinschränkungen der Java *Virtual Machine*, die Netzwerkverbindungen von Applets auf den originären Web-Server beschränken. Selbst wenn sich das Applet auf die Kommunikation mit seinem Web-Server beschränkt, gibt es immer noch genügend Untiefen in Form von *Fire-Walls* und veralteten JVMs, die manche Netzwerkoperationen nicht zulassen, die eine reibungslose Kommunikation behindern.

Verteilte Objekte befreien den Programmierer und Benutzer von Web-Computing-Infrastrukturen von den genannten Problemen und bieten stattdessen einen abstrakten Baukasten zur Konstruktion kooperierender Web-Anwendungen. Wenn zunehmend mehr Software- und Hardware-*Fire-Walls* IIOP-durchlässig werden und Webobjekte sich innerhalb ihres Namensraumes kontaktieren und ansprechen lassen können, ist eine Web-Computing-Infrastruktur wie Locust nicht mehr für die Implementierung eigener Kommunikkations- und Tunnelpolitiken verantwortlich, um Restriktionen von JVMs und *Fire-Walls* zu umgehen. Sie kann sich dann ausschließlich um die Unterstützung von objektorientierter Web-Anwendungen der ersten Generation kümmern, deren Webobjekte mit Hilfe der Locust Infrastruktur "ausgeliefert" werden, als erster Schritt in Richtung auf ein Web Object Computing (WOC).

Der erste Fortschritt des WOC wäre sicher die Verbreitung neuer Programmiermodelle neben dem allgegenwärtigen *Master-Slave*-Modell, wie zum Beispiel dem *Work-Pile*- und *Pipeline*-Modell, zusammen mit einer Migration vom dateizentrierten zum verteilten Rechnen im WWW, die mit einer Neuorientierung des WWW vom Informations-Pool zum Medium für verteilte Anwendungen einhergeht. Wir erwarten eine Verschiebung der Serverobjekte weg von den Web-Servern, unter Umständen auf jeden beliebigen Knoten des Namensraums. Die Konzentration auf einzelne Server war in der Vergangenheit notwendig, weil die Client-Server-Interaktion auf Dateien ohne eigenen Kontext und Zustand basierte. Diese Dateien konnten nicht vom Web-Server entfernt werden, der über diesen Kontext und Zustand verfügte. Mit Web-Objekten, die ihren Kontext und Zustand mit sich führen, wird der Objektabstand im WOC irrelevant. Beispielsweise könnte ein Serverobjekt, auf das über einer Web-Seite zugegriffen wird, im Browser des Benutzers oder aber im Browser eines anderen Benutzers, der gerade auf diese Web-Seite zugreift, ablaufen.

6.5.2 WOC Anwendungen

Automatische Code-Auslieferung Wie wir gesehen haben, müssen WOC Anwendungen in der Regel erst mit Hilfe von Java Applets initialisiert werden, bis sowohl server- als auch clientseitig CORBA Objekte vorhanden sind, die ausschließlich über IIOP kommunizieren können. Dies erfordert das Laden eines Applets samt weiterer umfangreicher Klassen, um zusätzliche Funktionalität, wie zum Beispiel eine grafische Benutzungsoberfläche, bereitzustellen. Um die Ladezeit solcher Klassen zu reduzieren, wird der *Update*-Mechanismus der Locust-Infrastruktur eingesetzt. Das Applet und alle zusätzlichen Klassen werden über den *Update*-Server angefordert und verwaltet. Damit ist sicher gestellt, daß stets die aktuellste Version vom am geografisch nächsten Locust-Server angefordert wird. *Update*-Server überprüfen selbständig die Gültigkeit des zwischengepeicherten Codes und fordern gegebenenfalls eine aktuelle Version an, so daß auf dem Hauptserver neu eingespielter Code automatisch an Teilmärkte propagiert wird. Eine Hierarchie von *Update*-Servern reduziert Latenz und notwendige Bandbreite zur Code-Auslieferung weiter.

Neuartige CORBA-Dienste Weitere DOC-Anwendungen für die Locust-Infrastruktur sind neuartige CORBA-Dienste, die ihre Eigenschaften (Multiplikation, Lokalität und Rekursivität) ausnutzen. Als Beispiel für eine Anwendung sei ein spezieller CORBA-Namensdienst angeführt, der rekursiv gekoppelte Domänen – den Locust-Teilmärkten – mit einem globalen Namensraum austattet. Der Locust-Namensdienst erweitert den herkömmlichen CORBA-Namensdienst mit impliziter Latenz-Sensivität und Lastverteilung, das heißt, der Namensdienst stellt automatisch die geografisch nächstgelegene Instanz, bzw. diejenige mit der geringsten Latenz, für einen bestimten Dienst zur Verfügung. Dadurch wird ein transparenter und simpler Lastverteilungsmechanimus implementiert, der auf einer topologisch günstigen Initialplazierung beruht. Darüberhinaus stellt der Locust-Namensdienst *globale* Namen innerhalb der angeschlossenen Domänen (Teilmärkte) zur Verfügung, die in der CORBA-Spezifikation nicht enthalten sind.

6.6 Zusammenfassung

Dieses Kapitel hat die vielen Anwendungsfelder – auch außerhalb paralleler Anwendungen – aufgezeigt und soweit möglich evaluiert, die sich aus der universellen Konzeption des generischen Locust-Modells ergeben. Den Schwerpunkt dieses Kapitels bildeten grobgranulare Anwendungen aus dem verteilten Hochleistungsrechnen wie die parallele RC5 Entschlüsselung und der parallele Raytracing-Algorithmus. Ihre erfolgreiche Evaluierung bilden die Basis für den Nachweis der Leistungsfähigkeit der Locust-Modells und seiner Implementierung. Doch auch mögliche Anwendungen für der Infrastruktur aus dem Bereich des verteilten Hochdurchsatzrechnens, die mit ausgearbeiteten Implementierungsplänen vorgestellt wurden, zeigen den breiten Einsatzbereichs von Locust.

Die Leistungsbetrachtungen wurden mit Hilfe der implementierten grobgranularen parallelen Anwendungen in einer LAN-Umgebung durchgeführt. Sowohl die RC5-Dechiffrierung als auch die Raytracing-Anwendungen bieten in einem weit nutzbaren Bereich der Zahl der Teilnehmer skalierbare Leistung. Nur am Anfang und Ende des Leistungsspektrums machen sich der zusätzliche Aufwand der Locust-Infrastruktur sowie fehlende Hierachie-Ebenen zur Reduzierung der Verwaltungs- und Transaktionskosten bemerkbar. Die erzielbare Leistung ist innerhalb des nutzbaren Bereichs durch die verfügbare Netzwerkbandbreite ins LAN bzw. WAN beschränkt, so daß höhere Teilnehmerzahlen und akkumulierte Rechenleistung ohne weitere Replikations- und Lastverteilungsmechanismen nur durch Erhöhung der verfügbaren Bandbreite erreicht werden können.

Neben den konkreten Anwendungs-Realisierungen wurde die vielseitige Anwendbarkeit von Locust durch eine Reihen weiterer, detailliert vorgestellter Applikationen aus den Bereichen des *Cachings* von Web-Dokumenten und des Tests von *Client-Server*-Umgebungen sowie des *Distributed Object Computings* nachgewiesen. Die konkrete Beschreibung dieser Anwendungen unterstreicht die allgemeine Anwendbarkeit der Locust-Infrastruktur, die sich als Ausführungsumgebung für Anwendungen empfiehlt, die von seinen Grundprinzipipien, der Multiplikation, Lokalität und Rekursivität, profitieren können.

Kapitel 7

Zusammenfassung und Ausblick

Im abschließenden Kapitel werden zunächst die Kernpunkte der Arbeit zusammengefaßt und einer kritischen Bewertung unterworfen. Die Ergebnisse dieser Evaluierung sind Grundlage weitergehender Fragestellungen und zukünftigen Forschungsarbeiten.

7.1 Zusammenfassung

Das exponentielle Wachstum von Weitverkehrsnetzen insbesondere des Internet hat zu einer drastischen Vergrößerung des Pools weltweit verfügbarer, aber sehr verteilter und heterogener Ressourcen geführt.

Web- und Meta-Computing Drei verwandte Ansätze, die Mittelpunkt des zweiten Kapitels sind, stehen zum uniformen Zugriff auf solche Ressourcen zur Verfügung: Meta-Computing-, Web-Computing- und Rechennetz-Infrastrukturen (*Computational Grids*). Besonderer Beitrag dieses Kapitels zur vorliegenden Arbeit ist die Erarbeitung einer präzisen, aufeinander aufbauenden Nomenklatur, um Ordnung in die inkonsistent verwendeten Begriffe zu bringen. Die beiden erstgenannten Ansätze unternehmen den Versuch, höhere Rechenleistung zu erzielen und dabei die Verteilung soweit wie möglich oder technisch sinnvoll zu verbergen. Rechen-Netze hingegen stellen mit einer allgegenwärtigen Infrastruktur zur logischen Verknüpfung verteilter und heterogener Rechen-Ressourcen das Fundament eines verteilten Rechensystems zur Verfügung. Meta-Computing-Systeme sind ohne Ausnahme durch die Verwendung dedizierter Ressourcen und geschlossener Schnittstellen zur Systemsoftware gekennzeichnet, das heißt sie erzielen die Erhöhung der Rechenleistung durch Erhöhung der *Qualität* der beteiligten Komponenten im Hinblick auf Leistungsfähigkeit und Zuverlässigkeit. Dies schränkt die Zahl der Teilnehmer, die diese Ansprüche erfüllen, deutlich ein. Ganz im Gegensatz dazu steigern Web-Computing-Infrastrukturen die Rechenleistung durch die *Quantität* der beteiligten Komponenten, auch wenn diese unter Umständen von minderer Qualität sind. Web-Computing Systeme richten sich mit Hilfe weniger effizienter, aber offener und allgegenwärtiger Protokolle und Programmiermodelle des *World Wide Web* (WWW) wie Java, Browser und HTTP an gelegentliche Nutzer mit opportunistischen Ressourcen. Insofern sind zur Sammlung freier Ressourcen Technologien und Protokolle des Web-Computings besser geeignet als solche des Meta-Computings. Anhand des erarbeiteten Definitionen werden die Meta-Computing-Systeme ATLAS und Charlotte, das prototypische Rechen-Netz von Foster und Kesselman (*The Grid* [FK98b]) mit ihrer Testumgebung Legion sowie die Web-Computing-Infrastrukturen Javelin, SuperWeb vorgestellt und klassifiziert. Eine einge-

hende Untersuchung im Hinblick auf ihre Fähigkeit, opportunistische Ressourcen sammeln und verwerten zu können, fördert einen allen System gemeinsamen Mangel zutage, solche Ressourcen anonymer Teilnehmer für Standard-Anwendungen, sogenannten *Legacy*-Applikationen einsetzen zu können. Obwohl zumindest ein Teil der Infrastrukturen ihr Leistungspotential aus einer hohen Teilnehmerzahl schöpft, ist die Architektur einiger Web-basierten Umgebungen nicht auf hohe Skalierbarkeit ausgelegt. Alle Umgebungen erfordern die Installation von Software und Registrierung von Benutzern bevor eine Teilnahme am System möglich ist, Erfordernisse, die mit Sicherheit *nicht* eine hoch-skalierende Leistung bieten können.

Elektronische Märkte Nach einer allgemeinen Einführung in traditionelle prioritätenbasierte Ressourcen-Zuteilungsverfahren mit ihren Stärken und Schwächen insbesondere in offenen, verteilten Systemen, wurden im Kapitel 3 marktbasierte Verfahren zur gleichzeitigen Allokation mehrerer Ressourcen eingeführt. Der Hauptbeitrag dieses Kapitels zur ganzen Arbeit ist die Identifizierung und Behandlung der Besonderheiten beim Handel mit zeitkritischen und opportunistischen Ressourcen mit sofortiger[1] Abschreibung. Es folgt eine theoretische Abschätzung der Komplexität und Leistungsfähigkeit dieser Verfahren im Hinblick auf die Anforderung an den Handel von Rechenleistung. Anschließend folgt ein Überblick über konkrete Implementierung von Marktsystemen, sogenannten elektronischen Märkten, wie Enterprise und Spawn und hybriden Web-Computing-Infrastrukturen wie JavaMarket und Popcorn. Echte elektronische Märkte als hinsichtlich Teilnehmern und Ressourcen geschlossene Systeme zeichnen sie dabei durchgehend durch niedrige Transaktionskosten aus. Hybrid-Systeme, die einen zusätzlich Markt zur Verfügung stellen, können aufgrund ihrer Offenheit auch einen sehr dynamischen Pool von Ressourcen verwalten und erkaufen diesen Vorteil durch höhere Transaktionskosten. Eine Sonderrolle nimmt hier das Rechenleistungs-Handelssystem ReGTime ein, das nur einen Marktplatz anbietet, und den Großteil der Transaktionen und ihrer Kosten seinen Benutzern am Marktplatz vorbei aufbürdet. Mit Ausnahme von Spawn ist keines der Systeme hoch skalierbar und allen Markttypen gemeinsam ist das Fehlen von Mechanismen zur Schaffung homogener (Teil-)Märkte und die Berücksichtigung der Lokalität zur Reduzierung der Verwaltungs- und damit der Transaktionskosten, wie zum Beispiel anonyme und automatische Markteilnahme.

Modellierung eines Ressourcenmarkts Aufbauend auf den bisher erarbeiteten Grundlagen wurde in Kapitel 4 die Modellierung eines Marktes für opportune Ressourcen in Weitverkehrsnetzen vorgestellt. Als Zielvorgaben dienten die oben aufgeführten Defizite existierender Rechen-Infrastrukturen und Marktsysteme. Anforderungen zur Erfüllung dieser Vorgaben sind asymmetrische Modellierung der Marktteilnehmer orientiert an den Ressourcenanbietern, transparente Systemteilnahme, marktbasierte Ressourcenzuteilung, dezentrale, hierarchische Verwaltung über Zwischenhändler sowie die Bereitstellung von Anreizen zur Marktteilnahme. Aus diesen Anforderungen wurden drei Säulen abgeleitet, die ein funktionierendes Ressourcen-Handelssystem tragen können, nämlich die Ausnutzung von multiplikativen Effekten, die Berücksichtigung von Lokalität und die Anwendung von Rekursivität.

[1] unendlich hoher

Als Ergebnis dieses Anforderungskatalogs wurde das Ressourcen-Handelsmodell Locust (LOw cost Computing Utilizing Skimmed idle Time) mit folgenden Haupteigenschaften entworfen:

Ausnutzung multiplikativer Effekte – Durch Ausnutzung multiplikativer Effekte wird die große Zahl laienhafter Teilnehmer und ihre effiziente Verwaltung beherrschbar gemacht. Als Multiplikatoren werden Zwischenhändler eingesetzt, die viele Rechner oder Benutzer verwalten und dadurch die Reichweite des Ressourcen-sammelnden Locust-Clients vervielfachen. Sie sind für die Ressourcensammlung und -abrechnung der ihnen unterstellten Rechner und Nutzer verantwortlich. Der durch mehrere Ebenen solcher hierarchisch organisierten Zwischenhändler erreichbare exponentielle Multiplikationsfaktor reduziert den hierfür notwendigen Verwaltungs- und Serveraufwand um ein Vielfaches.

Berücksichtigung der Lokalität – Die hierarchisch organisierten Zwischenhändler und ihre Teilmärkte werden dezentral mit Hilfe marktbasierter Zuteilungsverfahren verwaltet, die auf lokalen Informationen beruhen. Lokale Politiken zur Sammlung, zum Handel und zur Abrechnung von Ressourcen, die Besonderheiten ihrer Teilmärkte hinsichtlich Transaktionskosten, Sicherheitsanforderungen sowie Teilnahmebedingungen und -anreizen berücksichtigen, haben dabei stets zum Ziel, die Kosten zur Verwaltung und Aufrechterhaltung des Ressourcenhandels so weit wie möglich zu reduzieren.

Anwendung von Rekursivität – Zum Aufbau und zur Verwaltung hierarchisch organisierter Zwischenhändler und Teilmärkte findet das Rekursivitätsprinzip Anwendung. Inhomogene Märkte oder überlastete Serverdienste werden rekursiv hinsichtlich einer Eigenschaft in kleinere und homogenere Einheiten aufgeteilt bis die entstandenen Teilmärkte hinsichtlich dieser Eigenschaft homogen geworden sind und ihre Serverdienste sich in einem Gleichgewichtszustand befinden. Die alten und neuen Märkte werden hierarchisch gekoppelt und dezentral von den untergeordneten Teilmärkten und Serverdiensten aus verwaltet.

Das Locust zugrundeliegende Marktteilnehmer-Modell ist inhärent asymmetrisch und in allen Teilaspekten zugunsten der zahlenmäßig gegenüber Nachfragern im Vorteil liegenden Anbietern von brachliegender Rechenleistung ausgerichtet. Die Anwendung der Rekursivität manifestiert sich in der hierarchischen Organisation der Zwischenhändler, die jeweils einen Teilmarkt verwalten, der bezüglich einer oder mehreren Eigenschaften homogen ist. Diese Homogenität wird ausgenutzt, um die Transaktionskosten und die Zutrittsbarrieren zur Marktteilnahme an lokale Gegebenheiten anzupassen und auf möglichst niedriges Niveau zu senken, da in homogenen Teilmärkten die Anforderungen an Sicherheit, Buchführung oder (finanziellen) Anreizen ungleich niedriger als in einem inhomogenem Markt untereinander fremder Teilnehmer.

Architektur und Implementierung Die Architektur und Implementierung des vorgestellten Ressourcen-Handelsmodells Locust war Gegenstand des 5. Kapitels. Aufbauend auf einer Reihe von notwendigen Entwurfszielen wie Skalierbarkeit, Portabilität, Wiederverwendbarkeit und Geschwindigkeit wurden Methoden und Verfahren evaluiert und zur

Verwendung ausgewählt, die geeignet sind, diese Ziele im Hinblick auf die drei Säulen des Locust-Modells – Multiplikation, Lokalität und Rekursivität – zu erreichen. Die ausgewählten Methoden bildeten das Fundament der Architektur der drei Hauptkomponenten, die diese drei Säulen umsetzen und implementieren, der Locust-Server, der Locust-Client und der Locust-Markt. Der Locust-Server als vielfädiger Server-Dienst hat die Aufgabe, eine Vervielfachung der Reichweite (Multiplikation) möglicher Anwendungen zu unterstützen und zu verwalten. Die Multiplikation wird zum Beispiel durch sehr leichtgewichtige und robuste Kommunikationsprotokolle und durch die Unterstützung mehrerer Steuerflüsse und mehrerer replizierter Server zur Lastverteilung bereitgestellt. Die Architektur des Locust-Clients ist dagegen in erster Linie darauf ausgerichtet, sich effizient und dynamisch an lokale Besonderheiten (Lokalität) anzupassen und diese zur Reduzierung der Transaktionskosten auszunutzen. Zur Lokalisierung gehören beispielsweise die Erkennung, ob der Client innerhalb eines Browsers oder eigenständig gestartet wurde, in welcher hierarchischen Ebene der Teilmärkte er sich befindet oder spezielle anwendungsabhängige Maßnahmen. Der Locust-Markt schließlich ist für Zusammenführung von Ressourcenangebot und -nachfrage zuständig. Mit Hilfe von vier Serverdiensten – für die Teilnehmerverwaltung, die marktliche Koordination, die Ereignisbehandlung sowie die Softwareverteilung – implementiert Locust einen elektronischen Markt für opportune und dedizierte Ressourcen. Maschinen können mittels gewünschter Anforderungen und Kriterien für eine Reservierung freigegeben oder erworben werden und verteilte oder sequentielle *Legacy*-Applikationen in Java für die Reservierungsdauer ausgeführt werden.

Anwendungen und Leistungsfähigkeit Kapitel 6 beschäftigte sich mit Anwendungen zur Nutzung der Locust Infrastruktur sowie der Messung und Bewertung ihrer Leistungsfähigkeit. Auf Grundlage verschiedener Anwendungsklassen, die den Grundprinzipien des Locust-Modells – Ausnutzung der Multiplikation, Lokalität und Rekursivität – folgen, werden mögliche und im Rahmen dieser Arbeit implementierte Applikationen vorgestellt. Das Kapitel beginnt zunächst mit der Beschreibung der der im Rahmen dieser Arbeit implementierten und evaluierten Locust-Anwendungen aud dem verteilten Hochleistungsrechnen, der parallelen RC5-Dechiffrierung und der parallelen Raytracing Anwendung. Ihre erfolgreiche Evaluierung bilden die Basis für den Nachweis der Leistungsfähigkeit der Locust-Modells und seiner Implementierung. Es folgen detaillierte Konzepte weiterer verteilter Hochdurchsatz-Anwendungen aus dem *Web-Caching*, dem *Client-Server*-Test sowie des *Distributed Object Computing*, die mit ausgearbeiteten Implementierungsplänen versehen sind und die allgemeine Anwendbarkeit des Locust-Modells untermauern. Insgesamt erfolgt damit der Nachweis der Nützlichkeit und Leistungsfähigkeit des Systems.

Die Leistungsbetrachtungen wurden mit Hilfe der implementierten grobgranularen parallelen Anwendungen in LAN- und WAN-Umgebungen durchgeführt. Sowohl die RC5-Dechiffrierung als auch die Raytracing-Anwendungen bieten in einem weit nutzbaren Bereich der Zahl der Teilnehmer skalierbare Leistung. Nur am Anfang und Ende des Leistungsspektrums machen sich der zusätzliche Aufwand der Locust-Infrastruktur sowie fehlende Hierarchieebenen zur Reduzierung der Verwaltungs- und Transaktionskosten bemerkbar. Die erzielbare Leistung ist innerhalb des nutzbaren Bereichs durch die verfügbare Netzwerkbandbreite ins LAN bzw. WAN beschränkt, so daß höhere Teilnehmerzahlen und

akkumulierte Rechenleistung ohne weitere Replikations- und Lastverteilungsmechanismen durch bloße Erhöhung der verfügbaren Serverbandbreite erreicht werden können. Neben den konkreten Anwendungs-Realisierungen wurde die vielseitige Anwendbarkeit von Locust durch eine Reihe weiterer, detailliert vorgestellter Applikationen aus den Bereichen des *Distributed Object Computings*, des *Cachings* von Web-Dokumenten und des Tests von *Client-Server*-Umgebungen nachgewiesen. Die konkrete Beschreibung dieser Anwendungen unterstreicht die allgemeine Anwendbarkeit der Locust-Infrastruktur, die sich als Ausführungsumgebung für Anwendungen empfiehlt, die von ihren Grundprinzipien, der Multiplikation, der Lokalität und der Rekursivität, profitieren können.

7.2 Ausblick

Der erste Wunsch, der den Entwicklern einer Software von den Benutzern herangetragen wird, ist meist die Einführung weiterer Abstraktionsebenen in das Modell oder seine Implementierung zur Vereinfachung unangenehmer technischer Details. Im Fall der vorliegenden Arbeit bieten sich zwei mögliche Ansatzpunkte an, die Abstraktion der Unzuverlässigkeit von Ressourcen und der Schritt von bloßen Ressourcen hin zu abstrakten Diensten.
Der in Locust zur Anwendung kommende *Eager Scheduling*-Algorithmus zeigt bereits ohne eine solche Abstraktion ein robustes Verhalten gegenüber Teil- und Komplettausfällen der verwendeten opportunistischen Ressourcen, eignet sich jedoch nur für die Klasse von Anwendungen, deren Teilaufgaben sich ohne Einfluß auf das Ergebnis unabhängig voneinander und beliebig oft wiederholen lassen. Durch den Einsatz einer Softwareschicht, die transparent die Verfügbarkeit und Qualität seiner Ressourcen überwacht und unter Umständen auch garantiert (*Quality of Service*), könnte Locust für weitere Anwendungsklassen geöffnet werden, die die genannten Anforderungen nicht erfüllen.
Führt man die Garantierung von Dienstqualitäten konsequent weiter, gelangt man zu einem Punkt, an dem die Benutzer eines Web-Computing-Systems nicht mehr mit den bloßen Ressourcen in Berührung kommen und stattdessen nur mehr mit Diensten interagieren. Dies können Basisdienste wie Namens-, Makler- und Verwaltungsdienste sein, oder aber höherwertige Anwendungen wie mathematische oder numerische Bibliotheken, die bereits lokal zur Verfügung stehen oder aber von der Infrastruktur dynamisch bereit gestellt werden. Werden durch eine Infrastruktur nur noch Dienste zur Verfügung gestellt, ist der Schritt zum *Application Service Providing* (ASP) nicht mehr weit, allerdings aufgrund der darunterliegenden opportunistischen Ressourcen mit geringeren Dienstgarantien als bei herkömmlichen ASP-Anbietern. Dieser Nachteil kann jedoch bei geeigneten Diensten durch einen sehr günstigen Preis wettgemacht werden, da auf einen Großteil des sonst üblichen Maschinenparks verzichtet werden kann.

Trotz der guten Ergebnisse innerhalb der Testumgebungen beinhaltet ein Schritt vom Prototyp zu einem *Real-World*-Einsatz ein nicht unerhebliches Risiko. Zunächst einmal sind die Tests von der universitären Umgebung in möglichst reale Szenerien zu übertragen. Da sich opportunistische Ressourcen in dedizierten Testumgebungen nur schwer nachbilden lassen, sind hierzu reale Tests in WANs sowie dem Internet selbst mit bekanntem Aufwand und Nachteilen in Kauf zu nehmen. Solche Tests mit den von ihnen verursachten Kosten lassen sich nur schwer von reinen Forschungsprojekten durchführen. Technische Voraussetzung hierfür ist die Anpassung der Locust-Kommunikationsprotokolle an

Sicherheitsrestriktionen von Inter-Netzwerken (*Firewall Awareness*). Solche Verbesserungen könnten eine offizielle Zertifizierung von Locust zur Signatur von Applets oder der Einsatz von TCP-Verbindungen und HTTP-Tunneling zur Überwindung von *Fire-Walls* sein.

Desweiteren hängt die Akzeptanz und damit der Erfolg des Locust-Modells von der Erfüllung der Wünsche dreier Gruppen von Teilnehmern ab. Ob dieser Spagat gelingt, der mit großer Sicherheit gleichbedeutend mit dem Durchbruch und damit auch der Profitabilität des Locust-Modells ist, läßt sich innerhalb eines universitären Inkubators auch mit exzessiven Tests nicht mit Sicherheit vorhersagen. Es bleibt daher weiteren Forschungsarbeiten und Marktforschungen vorbehalten, diese Unsicherheiten zum Gegenstand weiterer Untersuchungen zu machen. Insbesondere muß jeder langfristig funktionierende Anreiz zur Teilnahme an Locust eine finanzielle Komponente enthalten, ein Umstand, der sich kaum innerhalb universitärer Forschung untersuchen läßt. Technische Voraussetzung für monetäre Anreize zur Marktteilnahme wäre eine Software-Komponente (*Accounting*) zur Führung, Verwaltung und Auszahlung von Multiplikatorkonten. Wirtschaftliche Voraussetzung für einen solchen Schritt wären risikobereite Geldgeber zur Initialisierung eines funktionierenden und stabilen Geld- und Güterkreislaufs.

Es ergibt sich daher als naheliegendster Schritt für weitere Arbeiten der Versuch der (teilweisen) kommerziellen Verwertung des vorgestellten Ressourcenhandels-Modells. Die bloße Simulation von Marktmechanismen in Zusammenhang mit der Sammlung und Nutzung opportunistischer Ressourcen in Weitverkehrsnetzen allein macht wenig Sinn.

Insgesamt liefert diese Arbeit einen grundlegenden Beitrag zur Thematik des Handels und Zugriffs auf entfernte, heterogene und opportunistische Ressourcen in Weitverkehrsnetzen. Sie schließt die klaffende Lücke zwischen der geringen Leistungsfähigkeit und Transparenz einerseits, die durch den simplen Zugriff auf entfernte Ressourcen über das *Common Gateway Interface* (CGI) erreichbar ist, und der extrem hohen Leistungsfähigkeit – aber auch Kosten – dedizierter Rechenzentren und Meta-Computing-Umgebungen andererseits. Es verbleibt abzuwarten, wie die anvisierten Benutzergruppen die Locust-Infrastruktur annehmen, um abschätzen zu können, ob sich das Internet selbst – im Gegensatz zu geschlossenen, dedizierten Meta-Computing-Systemen, die dieselben Protokolle und Schnittstellen verwenden – als Ausführungsumgebung für verteilte und parallele Anwendungen eignet.

Literaturverzeichnis

[AAB98a] *Yair Amir, Baruch Awerbuch und R.Sean Borgstrom.* A Cost-Benefit Framework for Online Management of a Metacomputing System. In „ICE 98“, Charleston, USA (1998).

[AAB98b] *Yair Amir, Baruch Awerbuch und Ryan S. Borgstrom.* The Java Market: Transforming the Internet into a Metacomputer. Technical report cnds-98-1, Center for Networking and Distributed Szstems, Johns Hopkins University, Baltimore, USA (Januar 1998).

[AG96] *Ken Arnold und James Gosling.* „The Java Programming Language, The Java Series“. Addison-Wesley, Reading, USA (Mai 1996).

[AISS97] *Albert D. Alexandrow, Maximilian Ibel, Klaus E. Schauser und Chris J. Scheiman.* SuperWeb: Research Issues in Java-Based Global Computing. *Concurrency: Practice and Experience* (Juni 1997).

[App00] *Apple.* Apple WebObjects Webpage (2000). www: http://www.apple.com/webobjects/.

[Bac98] *Martin Backschat.* Dynasty II: A Novel Economic-based Approach to Dynamic Load Distribution in Large Heterogenous Workstation Networks. Vortrag, (1998). Doktorandenkolloquium, Institut für Informatik, Technische Universiät München.

[BBB96a] *J. Eric Baldeschwieler, Robert D. Blumofe und Eric A. Brewer.* ATLAS: An Infrastructure for Global Computing. In „Proceedings of the 7th ACM SIGOPS Eurpean Workshop on System Support for Worldwide Applications“ (1996).

[BBB96b] *J. Eric Baldeschwieler, Robert D. Blumofe und Eric A. Brewer.* Charlotte: Metacomputing on the Web. In „Proceedings of the 9th Conference on Parallel and Distributed Computing Systems“ (1996).

[BBC+96] *D. Bhatia, V. Burzevski, M. Camuseva, W. Furmanski G. Fox und G. Premchandran.* Webflow. In „Workshop on java for computational science and engineering workshop“, Syracuse University (Dezember 1996).

[BCCM] *Graydon Barz, Samuel S. Chiu, Jean Pascal Crametz und Ross Mayfield.* Pricing Bandwidth Derivatives. Bericht. www: www.ratexchange.com.

[BD96] *Lewis B. und Berg D.J.* „Threads Primer - A Guide to Multithreaded Programming". SunSoft Press, Mountain View (1996).

[BDJ96] *Nichols B., Buttlar D. und Proulx Farell J.* „Pthreads Programming". O'Reilly & Associates, Inc. (1996).

[BKKK98] *Arash Baratloo, Mehmet Karaul, Holger Karl und Zvi M. Kedem.* An Infrastructure for Network Computing with Java Applets. In „ACM Workshop on Java for Science and Engineering Computation" (Juni 1998).

[BN96] *Kilicarslan N. Bakircioglu N.* Teilportierung von FASTEST nach Fortran90. Fortgeschrittenenpraktikum, Institut für Informatik, Technische Universität München, München (1996).

[Bog94] *N. R. Bogan.* Economic Allocation of Computation Time with Computation Markets. Master thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, Cambridge, Massachusetts (Mai 1994).

[BR96] *R. Baldwin und R. Rivest.* The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms (Oktober 1996).

[Bro98] *John Browning.* A Nasdaq for Bandwidth. *Wired Online* (April 1998).

[BT94] *B.C.Neumann und T.Ts'o.* Kerberos:An Authentification Service for Computer Networks. In „Proceedings of the IEEE Communications", Band 32(9), Seiten 33–39 (November 1994).

[CCN+97] *Bernd O. Christiansen, Peter Cappello, Mihai F. Ionescuand Michael O. Neary, Klaus E. Schauser und Daniel Wu.* Javelin: Internet-Based Parallel Computing Using Java. *ACM Workshop on Java for Science and Engineering Computation* (Juni 1997).

[CD98] *Henri Casanove und Jack Dongarra.* Using agent-based software for scientific computing in the NetSolve system. *Parallel Computing* **24**(12-13), 1777–1790 (November 1998).

[CDL+96] *K. Chandy, B. Dimitrov, H. Le, J. Mandleson, M. Richardson, A. Rifkin, P. Sivilotti, W. Tanaka und L. Weisman.* A world-wide distributed system using Java and the Internet. In „Proceedings of the fifth IEEE international symposium on high performance distributed computing", Syracuse, NY, USA (August 1996).

[Cha95] *R. Chan.* Implementation of Farringdon's Web Computer. Final year computer science project, London (1995).

[CMU98] *CMU*. Netbill Project Homepage (1998). www: http://www.ini.cmu.edu/netbill/.

[Com97] *Compaq*. Millicent Webpage (1997). www: http://www.millicent.digital.com/.

[Cyb98] *Cybercash*. Cybercash Webpage (1998). www: http://www.cybercash.com.

[Dec98] *Peter Decker*. Wash and go - Webmaster gegen Konzerne (Dezember 1998). www: *http://www.webhits.de/webhits/wash_and_go.htm*.

[DFC+96] *Blumofe Robert D., Joerg Christopher F., Kuszmaul Bradley C., Leiserson Charles E., Randall Keith H. und Zhou Yuli*. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing* **1**(37), 55–69 (August 1996).

[DFP+96] *T. DeFanti, I. Foster, M. Papka, R. Stevens und T. Kuhfuss*. Overview of the I-WAY: Wide Area Visual Supercomputing. *IJSA* **10**(2), 123–130 (1996).

[dis] *distributed.net*. Bovine RC5 Cracking Effort. www: http://www.distributed.net.

[dM] *Frederico Inacio de Moraes*. Java RayTrace Code. www: http://www.dcc.unicamp.br/chico/raytracing/raytracing.html.

[DS] *Court Demas und Martin Sweitzer*. Computation Markets Whitepaper. www: http://www.computationsmarkets.com.

[DW97] *D.Garcia und W.Watson*. ServerNet II. In „Proceeedings Parallel Computer Routing and Communications Workshop“. Springer (1997).

[EDR90] *Mohr Eric, Kranz David und Halstead Robert*. Lazy task creation: A technique for increasing the granularity of parallel programs. *IEEE Transactions on Parallel and Distributed Systems* (1990).

[Far96] *Jonathan Farringdon*. A Web Computer. research note rn-1996-58, (1996).

[FF97] *G. Fox und W. Furmanski*. Computing on the web - new approaches to parallel processing - petaop and exaop performance. *IEEE Internet Computing* (Januar 1997).

[FK97] *Ian Foster und Carl Kesselman*. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications* **11**(2), 115–128 (1997).

[FK98a] *Zabatta Fabian und Ying Kevin*. A Thread Performance Comparison: Windows NT and Solaris on A Symmetric Multiprocessor. In „Proceedings of the Second Windows NT symposium“ (August 3-4 1998).

[FK98b] *Ian Foster und Carl Kesselman* (Herausgeber). „The Grid: Blueprint for a New Computing Infrastructure“. Morgan-Kaufmann (Juli 1998).

[Fla96a] *David Flanagan*. „Java Examples in a Nutshell“. O'Reilly & Associates, Inc. (1996).

[Fla96b] *David Flanagan*. „Java in a Nutshell“. O'Reilly & Associates, Inc. (1996).

[Fla98] *David Flanagan*. „JavaScript“. O'Reilly & Associates, Inc. (1998).

[Fra99] *Wolfgang Franke*. The Role of Computer- and Software Giants in the Creation of Efficient Internet Payment Systems. In „Bezahlen im Internet 99“. Technische Universität Chemnitz (1999).

[Fü98] *Michael Fürsich*. Konzept und prototypische Realisierung eines Netzwerk-Computers auf Basis von Einchip-PC's. Master thesis, Institut für Informatik, Technische Universität München, München, Germany (1998).

[GD83] *Herausgeber Günther Drosdowski*. „Duden - Deutsches Universalwörterbuch“. Dudenverlag (1983).

[Gib97] *W. Wayt Gibbs*. World Wide Widgets. *Scientific American* (Mai 1997). www: http://www.sciam.com/0597issue/0597cyber.html.

[GK94] *David Gelernter und David Kaminsky*. Supercomputing out of Recycled Garbage: Preliminary Experience with Piranha. In „Proceedings of the 1992 ACM International Conference on Supercomputing“ (November 1994).

[GMA+95] *Steve Glassman, Mark Manasse, Martn Abadi, Paul Gauthier und Patrick Sobalvarro*. The Millicent Protocol for Inexpensive Electronic Commerce (Oktober 1995). www: http://http.cs.berkeley.edu/ gauthier/millicent/millicent.html.

[GPO98] *Pavel Gladychev, Ahmed Patel und Donal O'Mahony*. Cracking RC5 with Java Applets. In „ACM Workshop on Java for Science and Engineering Computation“ (Juni 1998).

[Gra95] *John R. Graham*. „Solaris 2.x: Internal & Architecture“. McGraw-Hill, Inc. (1995).

[Gro92] *OMG (Object Management Group)*. The Object Management Architecture Guide. Bericht, (1992).

[Gro97a] *OMG (Object Management Group)*. A Discussion of the Object Management Architecture. Bericht, (Januar 1997).

[Gro97b] *OMG (Object Management Group)*. CORBAservices: Common Object Services Specification. Bericht, (November 1997).

[Gro98] *OMG (Object Management Group).* The Common Object Request Broker: Architecture and Specification — Revision 2.2. Bericht, (Februar 1998).

[GW94] *Andrew S. Grimshaw und Wm. A. Wolf.* Legion: The next logical step softward the world-wide virtual computer. Technical Report CS-94-21, University of Virginia (Juni 1994).

[Hal85] *R.H.Jr. Halstead.* Multilisp: A language for concurrent symbolic computation. In „ACM Trans. on Programming Languages and Systems", Band 4, Seiten 501–538 (1985).

[Her98] *Amir Herzberg.* Charging for Online Content. *D-Lib Magazine* (Januar 1998).

[Hew] *Greg Hewgill.* RC5 and Java. www: http://www.hewgill.com.

[HSRV96] *H.Schulzrinne, S.Casner, R.Frederick und V.Jacobson.* RTP: A Transport Protocol for real-time Applications. Bericht, (1996).

[HU97] *A. Huber und T. Ungerer.* ReGTime - eine neue Serviceleistung im elektronischen Markt. In „Workshop Ökonomie und Modellierung Elektronischer Märkte im Rahmen der 4. Deutschen Tagung Wissensbasierte Systeme, XPS-97" (1997).

[HYW98] *Lu Honghui, Hu Y.Charlie und Zwaenepoel Willy.* OpenMP on Networks of Workstations. In „Proceedings of the Tenth High Performance Networking and Computing Conference" (November 7-13 1998).

[Int] *Mercury Interactive.* The Mercury Interactive Homepage. www: http://www.mercury-interactive.com/.

[JA97] *J.Gray und A.Reuter.* „Transaction Processing: Concepts and Techniques". Morgan Kaufmann Publishers (1997).

[JE98] *Narlikar Girija J. und Blellocj Guy E.* Pthreads for Dynamic and Irregular Parallelism. In „Proceedings of the Tenth High Performance Networking and Computing Conference" (November 7-13 1998).

[JS97] *J.Carreira und J.G. Silva.* Implementing Tuple Space with Threads. In „Proceeding of the EuroPDS'97" (1997).

[Kri98] *Mark Kriegsman.* An open Letter to Web Surfers and Webmasters (Dezember 1998). www: http://www.clearway.com/AdScreen/.

[Lam] *Hampel & Lambert.* OpenIPO: Open Initial Public Offering. www: http://www.openipo.com.

[Los99] *Fabian Loschek*. Entwicklung und Implementierung einer Schlüsselverwaltung zur Dechiffrierung des RC5 Algorithmus. Systementwicklungsprojekt, Institut für Informatik, Technische Universität München, München, Germany (1999).

[LSD+93] *L.Zhang, S.Deering, D.Estrin, S.Shenker und D.Zappala*. RSVP: A new Resource Reservation Protocol. **7(5)**, 8–18 (1993).

[MA98] *Ed. Michael Atallah*. „Handbook on Algorithms and Theory of Computation". CRC Press (November 1998).

[Mac99] *Macromedia*. Macromedia Flash Webpage (1999). www: http://www.macromedia.com/software/flash/.

[Man99a] *Michael Mann*. Mediated Reality: University of Toronto RWM Project. *Linux Journal* Seiten 50–57 (März 1999).

[Man99b] *Michael Mann*. University of Toronto WearComp Project. *Linux Journal* Seiten 10–19 (Februar 1999).

[Mar00] *Open Market*. FastCGI Webpage (2000). www: http://www.fastcgi.com.

[May] *Michael May*. The Locust Homepage. www: http://wwwbode.in.tum.de/ maym/locust/.

[may97] Internet Keyed Payment Protocols (IKP) (Oktober 1997). www: http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/iKP.html.

[May99a] *Michael May*. Distributed RC5 Decrypting as a Consumer for Idle-time Brokerage. In „Workshop on Distributed Computing on the Web (DCW 99)", Rostock (Juni 1999).

[May99b] *Michael May*. Idle Computing Resources as Micro-Currencies - Bartering CPU Time for Online Content. In „Proceedings of the Webnet World Conference 99 (WebNet'99)", Honolulu, Hawaii, USA (Oktober 1999).

[May99c] *Michael May*. Locust - A Brokerage System for Accessing Idle Ressources for Web-Computing. In „Proceedings of the 25th EUROMICRO Conference (EUROMICRO '99)", Mailand, Italien (September 1999).

[May99d] *Michael May*. Towards the Object Web: Combining Web computing and Distributed Object. In „Proceedings of the SoftCom'99", Split (Oktober 1999).

[MD88] *M.S. Miller und K.E. Drexler*. „Market and Computation: Agoric Open Systems". North-Holland (1988).

[Men82] *H. Mendelson*. Market Behaviour in a Clearing House. *Econometrica 47, pp. 61-74* (1982).

[MFGH88] *T.W. Malone, R.E. Fikes, K.R. Grant und M.T. Howard*. Enterprise: A Market-like Task Scheduler for Distributed computing. In „The Ecology of Computation, B.A. Huberman,Ed.“, Amsterdam (1988). North-Holland.

[Mic96] *Sun Microsystems*. NC Reference Profile 1 (Juli 1996). www: *http* : *//www.nc.ihost.com/nc_ref_profile.html*.

[Mic97] *Jochen Michels*. Finanzmanagement für die Datenverarbeitung (1997). www: http://home.t-online.de/JMichels/.

[Mic98a] *Microsoft*. PC 97 Hardware Design Guide (1998). www: http://www.microsoft.com/hwdev/download/desguid/pc97.zip.

[Mic98b] *Microsoft*. The Distributed Component Object Model - DCOM Architecture. Bericht, (1998).

[Mic99a] *Thierry Michel*. Common Markup for micropayment per-fee-links. W3c working draft 25 august 1999, (August 1999). www: http://www.w3.org/TR/Micropayment-Markup.

[Mic99b] *Microsoft*. Microsoft VBScript Webpage (1999). www: http://msdn.microsoft.com/scripting/default.asp.

[Mic00a] *Microsoft*. Microsoft Active ServerPages Webpage (2000). www: http://www.microsoft.com/Office/intranet/Modules/asp411s3.asp.

[Mic00b] *Microsoft*. Microsoft ActiveXWebpage (2000). www: http://www.microsoft.com/com/tech/ActiveX.asp.

[MMH+98] *Frumkin Michael, Hribar Michelle, Jin Haoqiang, Waheed Abdul und Yan Jerry*. A Comparison of Automatic Parallelization Tools/Compilers on the SGI Origin 2000. In „Proceedings of the Tenth High Performance Networking and Computing Conference“ (November 7-13 1998).

[MT97] *Griebel Michael und Schiekofer Thomas*. An adaptive sparse grid Navier-Stokes solver in 3D based on the finite difference method. In „Proceedings ENUMATH97“ (1997).

[Mue92] *Frank Mueller*. Implementing POSIX Threads under UNIX: Description of Work in Progress. In „Software Engineering Research Forum '92“ (1992).

[NCR+95] *N.J.Boden, D. Cohen, R.E.Felderman, A.E.Kulawik, C.L.Seitz, J.N.Seizovic und W.K.Su*. Myrinet - A Gigbit-per-second Local-Area Network. In „IEEE Micro“, Band 15(1), Seiten 29–36 (1995).

[Net98] *Netbill*. Netbill Webpage (1998). www: http://www.netbill.com.

[OH97] *Robert Orfali und Dan Harkey*. „Client/Server Programming with Java and CORBA“. John Wiley and Sons (1997).

[OHE96] *Robert Orfali, Dan Harkey und J. Edwards*. „The Essential Distributed Objects Survival Guide“. John Wiley and Sons (1996).

[OMG] *OMG Object Management Group*. The CORBA/IIOP Homepage. www: http://www.omg.org/corba/corbiiop.htm.

[OPT97] *Donal O'Mahony, Michael Peirce und Hitesh Tewari*. „Electronic Payment Systems“. Artec House (November 1997).

[PD93] *P.R.McAfee und D.Vincent*. The Declining Price Anomaly. *Journal of Economic Theory* **60**, 191–212 (Juni 1993).

[Pei96] *Michael Peirce*. PayMe: Secure Payment for World Wide Web Services. Computer science project, Dublin, Ireland (1996).

[PO95] *Michael Peirce und Donal O'Mahony*. Scaleable, Secure Cash Payment for WWW Resources with the PayMe Protocol Set. In „4th International World Wide Web Conference“, Boston, USA (Dezember 1995).

[P.S96] *Louis P.Slothouber*. A Model of Web Server Performance. Bericht, StarNine Technologies, Inc. (1996). www: http://www.starnine.com/.

[PSaGSM97] *Hernâni Pedroso, Luis M. Silva, Jo ao Gabriel Silva und Tavares Jose M.* Web-based Metacomputing with JET. In „ACM Workshop for Java for Science and Engineering Computation“, University of Twente, Netherlands (Juni 1997). World occam and Transputer User Group (WoTUG), IOS Press, Netherlands.

[Pub93] *Federal Information Processing Standards Publication*. Data Encyption Standard (DES) (Dezember 1993).

[Ran99] *Appleton Randy*. Understanding a Context Switching Benchmark. *Linux Journal* Seiten 70–71 (Januar 1999).

[Reg98] *Ori Regev*. Economic Oriented CPU Sharing System for the Internet. Master thesis, Institute of Computer Science, Hebrew University of Jerusalem, Jerusalem, Israel (1998).

[Res] *Georgia Tech Research*. WWW Online Survey 1998. www: http://www.giorgiatech-research.com.

[Riv95] *Ronald L. Rivest*. The RC5 Encryption Algorithm. In „Proceedings of the Second International Workshop on Fast Software Encryption“, Seiten 86–96, Leuven, Belgium (Januar 1995). Springer.

[RN98] *Ori Regev und Noam Nisan*. The POPCORN Market - an Online Market for Computational Ressources. In „Proceedings of the ICE 98“, Charleston, USA (1998).

[Rob] *D. Robinson*. The WWW Common Gateway Interface. Bericht. Internet Draft, Version 1.1.

[Rob54] *R. M. Robinson*. Mersenne and Fermat numbers. In „Proceedings of the American Mathematical Society“ (1954).

[RSW90] *Rustichini, Satterwaithe und Williams*. Convergence to Price-Taking Bahviour in a Simple Market. Bericht, (Dezember 1990).

[RT98] *Blikberg Ragnhild und Sorevik Tor*. Early Experiments with OpenMP on the Origin 2000. In „IPP“ (1998).

[R.W83] *R.Weber*. „Multi Objects Auctions“. New York University Press (1983).

[Sch93] *Alexander Schill*. „DCE - Das OSF Distributed Computing Environment“. Springer (1993).

[Sch96] *Klaus Schauser*. Research Issues in Java-based Global Computing. Vortrag, (Oktober 1996). Lecture Series TU Munich.

[SDB96] *Kleiman S., Shah D. und Smaalders B.* „Programming with Threads“. SunSoft Press (1996).

[Sec] *RSA Data Security*. The RSA Data Security Secret-Key Challenge. www: http://www.rsa.com/rsalabs/97challenge/.

[Ser98] *Clickshare Service*. Clickshare Webpage (Oktober 1998). www: http://www.clickshare.com/clickshare/.

[SET] *SETI*. SETI@home: Search for Extraterrestrial Intelligence at Home. www: http://setiathome.ssl.berkeley.edu/.

[SHW98] *Luis F.G. Sarmenta, Satoshi Hirano und Stephen A. Ward*. Towards Bayanihan: Building an Extensible Framework for Volunteer Computing Using Java. In „ACM Workshop on Java for Science and Engineering Computation“ (Juni 1998).

[SKP97] *Ben Lamine S., P. Kropf und J. Plaice*. Problems of Computing on the Web. In *A. Tentner* (Herausgeber), „High performance computing symposium 97“, Seiten 296 – 301, The Society of Computer Simulation International, Atlanta, GA (April 1997).

[SPaGS97] *Luis M. Silva, Hernâni Pedroso und Jo ao Gabriel Silva*. The Design of JET: A Java Library for Embarrassingly Parallel Applications. In *A. Bakkers* (Herausgeber), „Parallel Programming and Java, Proceedings of WoTUG 20“, Band 50 aus „Concurrent Systems Engineering“, Seiten 210–228, University of Twente, Netherlands (April 1997). World occam and Transputer User Group (WoTUG), IOS Press, Netherlands.

[SSN99] *Lakshminarayanan S., Ghosh S.S. und Balakrishnan N.* Implementation of MPI over HTTP. In „Proceedings of the HPCN 99“ (1999).

[SSS98] *F. Soares, L. M. Silva und J.G. Silva.* How to get Volunteers for Web-based Metacomputing. In „Workshop Distributed Computing on the Web“ (1998).

[Sun97] *Sun Microsystem.* Java Remote Method Invocation Specification — Revision 1.2. Bericht, (Oktober 1997).

[SVC+97] *S.Floyd, V.Jacobson, C.G.Liu, S.McCanne und L.Zhang.* A Reliable Multicast Framework for light-weight Sessions and Application Level Framing. In „ACM/IEEE Transactions on Networking“, Band 5(6), Seiten 784–803 (1997).

[Sve95] *Graupner Sven.* Nichtprozedurale Ablaufformen in imperativen Sprachen, Coroutinen und preemptive Threads in C. Chemnitzer informatik berichte tr-95-07, Fakultät für Informatik, TU Chemnitz-Zwickau (August 1995).

[TDG] *Inc. Trilogy Development Group.* JCM User Feedback: Computation Markets Survey. www: http://www.trilogy.com.

[Tec96] *IEEE Standard Information Technology.* IEEE Standard 1003.1-1996 - POSIX Threads, Portable Operating System Interface (POSIX), Part 1: System Application Program Interface (API) [C Language]. Institute of Electrical and Electronics Engineers, Inc. (1996).

[Tec98] *Ecash Technologies.* Ecash Webpage (1998). www: http://www.ecashtechnologies.com.

[TG98] *Schiekofer Thomas und Zumbusch Gerd.* Software Concepts of a Sparse Grid Finite Difference Code. In *G. Wittum W. Hackbusch* (Herausgeber), „Proceedings of the 14th GAMM-Seminar Kiel on Concepts of Numerical Software, Notes on Numerical Fluid Mechanics“. Vieweg (1998).

[Tho98] *Schiekofer Thomas.* „Die Methode der Finiten Differenzen auf Dünnen Gittern zur adaptiven Multilevel-Lösung partieller Differentialgleichungen“. Dissertation, Institut für angewandte Mathematik, Universität Bonn, Bonn (1998).

[VA98] *Karamcheti Vijay und Chien Andrew A.* A Hierrachical Load-Balancing Framework for Dynamic Multithreaded Computations. In „Proceedings of the Tenth High Performance Networking and Computing Conference“ (November 7-13 1998).

[Val96] *Uresh Valhalla.* „UNIX Internals: The New Frontiers“. Prentice-Hall (1996).

[Van97] *Laurence Vanhelsuwe.* Create your own Supercomputer with Java. *Javaworld* (Januar 1997). www: http://www.javaworld.com/javaworld/jw-01-1997/dampp/.

[Vic61] *W. Vickrey.* Counterspeculations, Auctions and Competitive Sealed Tenders. *Journal of Finance, 16, pp.8-37* (März 1961).

[Wal89] *Carl A. Waldsburger.* A Distributed Computational Economy for Utilizing Idle Ressources. Master thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, Cambridge, Massachusetts (Mai 1989).

[Wal92] *C.A. Waldsburger.* Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering, Vol.18, No.2* (Februar 1992).

[WBA95] *W.T.Strayer, B.J.Dempsey und A.C.Weaver.* „XTP: The Xpress Transport Protocol". Addison-Wesley, MIT Press (1995).

[Zah99] *Annja Zahn.* „Elektronisches Handeln mit Rechenleistung". Dissertation, Institut für Informatik, Universität Augsburg, Augsburg (1999).