SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY —
INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Bereitstellung von Forschungsdaten über eine Rechencloud: Eine Fallstudie zur Strömungsimulation des Wiedereintrittsvorgangs einer Apollo-artigen Raumkapsel

Fabian Nowak

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Bereitstellung von Forschungsdaten über eine Rechencloud: Eine Fallstudie zur Strömungsimulation des Wiedereintrittsvorgangs einer Apollo-artigen Raumkapsel

Enabling Access to Research Data via a Compute Cloud: A Case Study on Re-entry Flow Simulation of an Apollo-like Space Capsule

Autor:	Fabian Nowak
Prüfer:	Prof. Dr. Hans-Joachim Bungartz
Betreuer:	Apl. Prof. Dr.-Ing. habil. Christian Stemmer
	Friedrich Ulrich, M.Sc.
Abgabedatum:	15.01.2024

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, 15.01.2024 Fabian Nowak

Abstract

Cloud-Computing hat in den letzten Jahren ein enormes Wachstum erfahren. Es ermöglicht, Online-Dienste anzubieten, ohne selbst die Hardware und Infrastruktur verwalten zu müssen. Das Ziel dieser Bachelorarbeit ist es, die Compute-Cloud des Leibniz-Rechenzentrums (LRZ) zu verwenden, um darauf eine interaktive Visualisierungslösung für Simulationsergebnisse des Lehrstuhls für Aerodynamik und Strömungsmechanik umzusetzen. Mitglieder einer Forschungsgruppe sollen einzelne Verzeichnisse teilen können, die auf dem Data-Science-Storage (DSS) des LRZ liegen. Dann soll externen Forschenden ohne LRZ-Zugang ermöglicht werden, über den Webbrowser per Zugangslink die Datensätze zu betrachten. Zunächst wird das DSS per NFS-Mount mit der Cloud-Instanz verbunden, um direkten Zugriff auf die Simulationsergebnisse zu erlangen. Für die browserbasierte Remote-Visualisierung wird das ParaView Visualizer Tool verwendet, das auf Kitwares ParaView-Web Framework basiert und eine ParaView-ähnliche Browseranwendung bereitstellt.
Als erster Schritt wurde die Beispielkonfiguration aus der Dokumentations des ParaView Visualizers umgesetzt, bei der unter Verwendung von Kitwares wslink Python-Bibliothek bereits automatisiert bedarfsabhängig Visualizer-Instanzen gestartet werden können. Dabei wird der Apache HTTP Server als Reverse Proxy eingesetzt. In der Beispielkonfiguration wurde eine potenzielle Sicherheitslücke gefunden und anschließend behoben. Danach wurde die Konfiguration erweitert, um es mehreren Mitgliedern des DSS-Containers zu ermöglichen, eigenständig einzelne Verzeichnisse zu teilen. Dazu musste ein separates Tool für den Apache-Server entwickelt werden, um die Reverse-Proxy-Funktionalität zu erweitern. Außerdem wurde ein Python-Skript entwickelt, über das die DSS-Nutzer eigenständig Verzeichnisse teilen und verwalten können. Das Skript passt dabei die erweiterte Konfiguration so an, dass das gewünschte Verzeichnis geteilt wird.
Im Anhang findet sich eine Installationsanleitung.

Abstract

Cloud-Computing has experienced tremendous growth in recent years. It enables the provision of online services without the need to manage the underlying hardware and infrastructure. In this thesis, the Compute-Cloud of the Leibniz Supercomputing Centre (LRZ) is used to set up an interactive visualization solution for simulation results from the Chair of Aerodynamics and Fluid Mechanics. Members of the research group shall be able to share selected directories containing the datasets which are stored on the LRZ's Data Science Storage (DSS). External researchers without LRZ credentials shall then be able to view the datasets directly in their web browser using an access link.
First, the DSS is connected to the cloud instance via an NFS mount to allow direct access to the simulation results. For implementing the browser-based remote visualization, ParaView Visualizer was used. It is based on Kitware's ParaViewWeb Framework and provides a ParaView-like web application.
As a first step, the example configuration from the documentation of ParaView Visualizer was implemented. There, Kitware's wslink Python library is used to automatically start Visualizer instances on demand. Furthermore, an Apache HTTP Server was installed as a reverse proxy. During the project, a potential security vulnerability was found in the example configuration and subsequently fixed. Then, the configuration was extended to allow multiple members of a DSS container to independently share some of their own directories. This required the development of a separate program to extend the Apache Server's reverse proxy capabilities. Additionally, a Python script was developed which can be used by the DSS members to manage their shared directories. This script automatically modifies the extended configuration accordingly.
An installation guide is available in the appendix.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Ein wesentlicher Bestandteil der Wissenschaft ist die Kooperation zwischen verschiedenen Forschungsgruppen und damit verbunden der Austausch von Forschungsergebnissen und Forschungsdaten. Eine erneute Datenanalyse durch andere Forschungsteams hilft dabei, die Ergebnisse zu verifizieren. Dies ist eine wesentliche Komponente des Forschungsprozesses. Genauso können alternative Interpretationsansätze zum wissenschaftlichen Fortschritt beitragen [1]. Zudem lässt sich feststellen, dass Veröffentlichungen mit offen zugängliche Daten anschließend öfter zitiert werden (bis zu 30 %)[2]. Außerdem können durch die Verwendung bereits vorhandener Daten die Ressourcen, die für eine erneute Datenerhebung notwendig sind eingespart werden. Gerade für Computational-Fluid-Dynamics (CFD)-Simulationen wird viel Rechenzeit an High-Performance-Computing-Clustern aufgewendet. Entsprechend wichtig ist es, die Ergebnisse möglichst einfach verbreiten zu können.

1.2. High-Performance-Computing-Simulationen

Der Lehrstuhl für Aerodynamik und Strömungsmechanik der Technischen Universität München beschäftigt sich mit vielen verschiedenen Forschungsschwerpunkten, darunter auch der Strömungssimulation. Strömungssimulationen sind zum einen eine Alternative zum Windkanal und lassen sich in Situationen verwenden, in denen sich ein experimenteller Versuchsaufbau schwierig gestaltet. Sie erlauben diverse physikalischen Phänomene und chemische Effekte genauer zu analysieren, die zum Beispiel bei Strömungen mit Hyperschallgeschwindigkeit auftreten.
Eine Einführung zu Hypersonischen Strömungen findet sich bei J. Anderson [3].
Hyperschallgeschwindigkeit (ab Mach > 5) und die damit verbundene hypersonische Strömung findet man unter anderem beim Wiedereintritt von Raumkapseln. Die physikalischen und chemischen Effekte unterscheiden sich dabei stark von dem, was sich bei Überschallgeschwindigkeit (Mach > 1) und darunter beobachten lässt: Die hohe Geschwindigkeit bei hypersonischer Strömung im Vergleich zum Überschall führt dazu, dass die resultierende Schockwelle typischerweise verhältnismäßig nahe am eintretenden Körper liegt. Durch Wechselwirkungen von Druck, Dichte und Temperatur wird die viskose Grenzschicht um den Körper dicker als bei langsameren Geschwindigkeiten. Das führt dazu, dass der Körper aus aerodynamischer Sicht um einiges Dicker wirkt als er in Wirklichkeit ist. Durch die hohe Geschwindigkeit der Strömung entstehen hohe Temperaturen (Beispielsweise 11000K beim Wiedereintritt der Apollo Raumkapsel mit Mach 36). Diese Temperaturen sorgen

wiederum dafür, dass die Gase Sauerstoff und Stickstoff in der Atmosphäre nicht mehr in molekularer, sondern in atomarer Form vorliegen, bei höheren Temperaturen in Form von ionisiertem Plasma. Dies nimmt letztendlich wieder Einfluss auf die physikalischen Eigenschaften der Strömung. Die damit folgenden Effekte auf Temperatur, Reibungswiderstand, Auftrieb, etc. sind zum Beispiel bei der Entwicklung von Raumkapseln, Raketen und sonstigen Hyperschallflugkörpern von großer Bedeutung. Die Effekte müssen dabei in der Wahl der Konstruktionsmaterialien oder auch der Form des Flugkörpers mit einbezogen werden (vgl. [3]).

Die Durchführung von CFD-Simulationen erfolgt auf High-Performance-Computing (HPC) Systemen (siehe Abschnitt 2.1). Typische HPC Systeme zeichnen sich verglichen mit Personal Computern in erster Linie durch das Ausmaß an Parallelisierung aus, also der Fähigkeit, viele Aufgaben gleichzeitig durchzuführen. Diese Parallelisierung findet sich auf allen Hardwareebenen wieder: auf Instruktionsebene durch die Verwendung von SIMD-Instruktionen (Single-Instruction-Multiple-Data). Damit sind CPU-Instruktionen gemeint, die innerhalb dieser einen Instruktion dieselbe Operation auf mehrere Datenströme gleichzeitig anwenden, typischerweise die Elemente eines Vektors. Ein Beispiel ist die AVX (Advanced Vector Extensions) Befehlssatzerweiterung, mit der in einem Takt dank einer Registerbreite von bis zu 512 Bit bis zu 16 Fließkommaoperationen (32-bit, einfache Genauigkeit) gleichzeitig durchgeführt werden können. Die Verwendung von SIMD setzt allerdings voraus, dass der auszuführende zugrundeliegende Algorithmus auch entsprechend vektorisierbar ist [4]. Auf CPU-Ebene enthalten Prozessoren von einem HPC System eine große Anzahl Prozessorkerne, von denen jeder eine separate Aufgabe mit separaten Instruktionen durchführen kann. Auch Moderne PCs für Endverbraucher unterstützen heutzutage bereits in begrenztem Umfang SIMD-Befehle und besitzen mehrere Rechenkerne, wobei die Kernanzahl bei in HPC Systemen verwendeten Prozessoren meist höher ist, häufig sogar um ein Vielfaches. Für HPC Systeme werden viele solcher Computer zu einem großen System verbunden. Die einzelnen Computer werden Nodes oder Knoten genannt. Bei den leistungsstärksten Systemen kommen dabei Stand 2023 mehr als 9000 Nodes zusammen [5]. Jede dieser Nodes besteht aus einem, oder auch mehreren (z.B. Summit Supercomputer [6]) solcher CPUs. Jede Node besitzt ihren eigenen Arbeitsspeicher. Bei modernen Systemen kommen jeweils noch zusätzlich mehrere GPUs dazu (z.B. Frontier Supercomputer [5]). Die Nodes werden anschließend durch besondere durchsatzstarke Netzwerkverbindungen verknüpft. Aufgrund der stark parallelen Natur der HPC Systeme sind sie in erster Linie für Berechnungen geeignet, die sich angemessen parallelisieren lassen. Typische Anwendungsgebiete sind beispielsweise graphenbasierte Probleme, Probleme aus der linearen Algebra, oder auch Lösungen partieller Differenzialgleichungen. Unter letzteres fallen auch die Navier-Stokes-Gleichungen, die in der Strömungsmechanik relevant sind [4].

Trotz all dieser Leistungsoptimierungen kann eine hochauflösende Simulation mithilfe einer parallelisierbaren Lösungssoftware für die Navier-Stokes-Gleichungen auf dem CoolMUC-2 Cluster des LRZ (siehe Abschnitt 2.1) mehrere Tage an Rechenzeit beanspruchen, wobei die resultierenden Simulationsergebnisse mehrere Gigabyte an Speicherplatz belegen.

1.3. Cloud-Computing

Cloud-Computing ist eine moderne Methode, IT-Dienste, IT-Anwendungen und Daten bereitzustellen und stellt eine Alternative zu on-premise (also vor Ort eingerichteten) Rechenzentren dar. Dabei werden die Komponenten, die eine typische Datencenterinfrastruktur ausmachen, also Netzwerk-, Speicher- und Rechenressourcen abstrahiert und virtualisiert. Durch diese Abstraktion kann ein Cloudanbieter einem Kunden ganz nach Bedarf beliebige Mengen an Ressourcen aus einem großen, geteilten Pool unkompliziert, schnell und mit minimalem Verwaltungsaufwand bereitstellen. Der Kunde kann seinen Fokus dabei auf die Anwendungsschicht legen, während die Hardware und Infrastruktur, die dem Rechenzentrum zugrunde liegt, verborgen bleibt und alleinig vom Anbieter verwaltet wird. Neben der damit verbundenen einfachen bedarfsabhängigen Skalierung liegt ein weiterer Vorteil des Cloud-Computing darin, dass durch die gemeinsame Nutzung der Infrastruktur die Kosten pro Kunde reduziert werden können. Die Flexibilität in der Ressourcennutzung erlaubt zudem, dass als Kunde nur für die tatsächlich benötigten bzw. angeforderten Ressourcen gezahlt werden muss (vgl. [7]).
Diese Vorteile haben zu einem immensen Wachstum von Cloud-Computing-Diensten geführt. So ist der Cloud-Computing Markt von 145 Milliarden US-Dollar in 2017 auf 478,32 Milliarden US-Dollar in 2022 gestiegen [8].
Bei Cloud-Computing-Angeboten wird vom US-amerikanischen National Institute of Standards and Technology (NIST) zwischen drei verschiedenen Arten unterschieden, wobei in den verschiedenen Modellen immer mehr Verwaltungsaufwand und Kontrolle an den Kunden abgegeben wird [7, 9, 10]:

1. Beim „Software as a Service“-Modell (SaaS) stellt der Cloudanbieter eine Anwendung über das Internet für den Endnutzer bereit, auf die dann über den Webbrowser oder ein Programm zugegriffen wird. Der Nutzer hat dabei keinerlei Kontrolle über jegliche Aspekte der zugrundeliegenden Cloud-Infrastruktur.

2. Bei „Platform as a Service“-Angeboten (PaaS) stellt der Anbieter eine Plattform bereit, häufig inklusive Laufzeitumgebungen, auf der ein Cloudkunde selbst entwickelte Anwendungen bereitstellen lassen kann, sofern sie von der Plattform unterstützt werden bzw. in einer Programmiersprache entwickelt wurden, die vom Cloudanbieter unterstützt wird. Der Kunde muss dabei nicht die zugrundeliegende Infrastruktur, inklusive Betriebssystem verwalten und sich häufig nicht um die Ressourcenzuweisung kümmern. Stattdessen kann die Ressourcennutzung automatisch durch den Anbieter skaliert werden.

3. Bei „Infastructure as a Service“-Modellen (IaaS) stellt der Cloudanbieter in erster Linie die (virtualisierten) Hardwareressourcen (CPU, Speicher, etc.) im gebuchten Umfang bereit, auf denen dann beliebige Software, darunter auch Betriebssysteme installiert werden kann. Auch ausgewählte Netzwerkkomponenten können häufig konfiguriert werden.

Insbesondere kleinere und mittelgroße Unternehmen können von Cloudangeboten profitieren, da die Kosten für Hardware und Infrastruktur, die im Fall eines traditionellen Rechenzentrums im Voraus anfallen, vermieden werden [11]. Dazu kommt die Schwierigkeit, die erforderlichen Ressourcen im Voraus abzuschätzen, was entweder zu schlechter Verfügbarkeit der Anwendung oder im gegenteiligen Fall zu unnötigen Ausgaben führt, während bei einer Cloudnutzung die gebuchten Ressourcen schnell angepasst werden können. Auch Schwankungen im Ressourcenbedarf z.B. durch schwankende Nutzerzahlen können in dieser Weise einfach abgefangen werden [7].

1.4. Ziel dieser Arbeit

Die im vorherigen Abschnitt erwähnten Simulationsergebnisse des Lehrstuhls für Aerodynamik und Strömungsmechanik sollen Forschenden außerhalb der TUM zur Verfügung gestellt werden. Dabei soll Nutzen aus den technologischen Fortschritten im Cloud-Computing und in der Remote Visualisierung gezogen werden, um einfachen Datenzugriff bzw. Datenaustausch von Simulationsergebnissen zu ermöglichen.
Als Cloud-Infrastruktur soll die Compute-Cloud des LRZ verwendet werden, auf der dem Lehrstuhl bereits eine angemietete Compute-Cloud-Instanz zur Verfügung steht. Die zu teilenden Simulationsergebnisse liegen auf dem Data-Science-Storage (DSS, siehe Abschnitt 2.1) des LRZ.
Konkretes Ziel ist es, auf der Cloud-Instanz einen ParaView-basierten Visualisierungsdienst einzurichten, der Mitgliedern einer Forschungsgruppe mit LRZ-Zugang ermöglicht, ihre Ergebnisse mit externen Forschern zu teilen. Letztere sollen dann die Ergebnisse *interaktiv*, wenn möglich direkt vom Browser aus visualisieren können, ohne dass der gesamte Datensatz heruntergeladen werden muss. Somit wird insbesondere eine erste oberflächliche Evaluation vereinfacht und wissenschaftliche Kollaboration gefördert. Der Dienst soll direkt an den Data-Science-Storage des LRZ (Abschnitt 2.1) angebunden sein, auf dem die Simulationsergebnisse gespeichert sind, damit ein Kopieren oder Herunterladen potenziell großer Datenmengen durch die LRZ-Nutzer vermieden werden kann. Die LRZ-Nutzer sollen dann verschiedene Verzeichnisse ihrer Wahl auf dem DSS auswählen können, die separat voneinander veröffentlicht werden. Zugriff auf die Verzeichnisse soll über Zugriffslinks erfolgen, wobei jedes geteilte Verzeichnis über einen individuellen Link erreichbar ist. Die Arbeit beantwortet dabei in erster Linie die Frage der Machbarkeit und behandelt die damit einhergehende schrittweise technische Umsetzung.

In Kapitel 2 der Arbeit wird zunächst in Abschnitt 2.1 die LRZ Infrastruktur erläutert, auf der der Dienst bereitgestellt werden soll. Darauffolgend werden in Abschnitt 2.2 die Grundlagen der Remote-Visualisierung erklärt, wobei ein besonderer Fokus auf die Umsetzung durch die ParaView-Software gelegt wird. In den darauffolgenden Abschnitten werden noch weitere verwendete Werkzeuge und Software beschrieben.
In Kapitel 3 soll dann erläutert werden, wie die Funktionsweise des Dienstes konkret umgesetzt wurde. Als aller Erstes wird in Abschnitt 3.1 erläutert, wie von der Compute-Cloud auf DSS zugegriffen werden kann, und welche Einschränkungen damit einhergehen. Dann

wird in Abschnitt 3.2 zunächst die Dokumentation des ParaView Visualizers herangezogen. Die darin beschriebene Konfiguration wird repliziert und gleichzeitig die Funktionsweise und das Zusammenspiel der eingesetzten Komponenten analysiert. Das Ergebnis ist ein Visualisierungsdienst, mit dem ein einzelnes Verzeichnis bereitgestellt wird. Schließlich wird in Abschnitt 3.4 dieses Ergebnis herangezogen und so erweitert, dass die LRZ-Nutzer selbstständig beliebig viele Verzeichnisse teilen können und somit die zuvor definierten Ziele erreicht werden.

2. Methoden

2.1. LRZ Infrastruktur

Das Angebot des Leibniz-Rechenzentrums (LRZ) umfasst eine große Palette an Diensten, besonders im HPC Bereich.

SuperMUC-NG

Unter anderem betreibt das LRZ den Höchstleistungsrechner SuperMUC-NG. Dieser Supercomputer ist Stand November 2023 auf Platz 40 der TOP500 Liste [12], die die 500 leistungsstärksten Supercomputer der Welt auflistet. Er setzt sich aus 6480 Nodes zusammen, mit einer Gesamtzahl von 311040 Rechenkernen und einer Rechenleistung von 19476 Pflop/s (floating-point operations per second, dt. Gleitkommazahloperationen pro Sekunde). Als Dateisystem werden unter anderem ein besonders schnelles, paralleles High Performance Dateisystem verwendet mit einem theoretischen Durchsatz von 500 GB/s. Zudem ist der SuperMUC-NG an das DSS (siehe Abschnitt Data-Science-Storage) mit einer Geschwindigkeit von 70 GB/s angebunden [13].

Linux Cluster

Neben dem SuperMUC-NG gibt es auch noch zwei weniger leistungsstarke Segmente des Linux Clusters: CoolMUC-2 und CoolMUC-3. CoolMUC-2 besteht aus 812 Nodes mit jeweils 28 Prozessorkernen, insgesamt also 22736 Kerne die eine theoretische Spitzenleistung von 1400 Tflop/s erzielen. CoolMUC-3 besteht hingegen aus nur 148 Nodes, dafür mit jeweils 64 Prozessorkernen. Dafür wird in CoolMUC-3 zusätzlich spezieller Arbeitsspeicher mit besonders hoher Bandbreite verwendet (460 GB/s vs. 80 GB/s gewöhnlich). Beide CoolMUC Cluster sind zudem an das DSS angebunden. Neben dem DSS als Dateisystem gibt es noch ein Dateisystem für die Home-Verzeichnisse der Nutzer, sowie zwei vergleichsweise schnelle „scratch“-Dateisysteme mit Kapazitäten von 1400 TB bzw. 3100 TB für kurzlebige Daten und Zwischenergebnisse. Daten auf den Scratch-Systemen werden regelmäßig wieder gelöscht (vgl. [14], [15]).
Auf dem CoolMUC-2 Cluster wurde unter anderem die in Abschnitt 1.2 beschriebenen Eintrittssimulationen durchgeführt.

Data Science Storage (DSS)

Als Speicherlösung für die Langzeitspeicherung von Simulationsdaten und sonstigen Forschungsergebnissen bietet das LRZ das Data Science Storage. Das DSS erlaubt das

Tabelle 2.1.: Kostenloses LRZ Compute Cloud Kontingent [17]

Ressource	Limit	Beschreibung
Instanzen	4	Anzahl an VMs
CPU-Kerne	10	Gesamtzahl an Verfügbaren vCPUs über alle VMs hinweg
RAM	Unbegrenzt	Insgesamt verfügbarer RAM
Speicherplatz	200 GB	Insgesamt verfügbarer Blockspeicher

Speichern von riesigen Mengen an Daten zu Forschungszwecken. Verwaltet wird der Speicher in Form von DSS-Containern, die jeweils einem Forschungsprojekt zugehörig sind. Jedes Projekt hat einen oder mehrere Datenkuratoren, diese Rolle wird typischerweise vom Hauptforscher eines Forschungsprojekts übernommen. Die Datenkuratoren verwalten die Container und können Zugriffsrechte, Speicherlimits für einzelne Nutzer und weiteres festlegen. Ein weiteres Merkmal des DSS ist die Verbindung zu einem Großteil der LRZ Dienste, darunter SuperMUC-NG, CoolMUC-2 und -3 und die LRZ Compute Cloud (siehe Abschnitt LRZ Compute Cloud) [16].

LRZ Compute Cloud

Die LRZ Compute Cloud ist ein Dienst des LRZ für Kunden, die eigene virtuelle Maschinen erstellen und nutzen möchten. Der Lehrstuhl für Aerodynamik und Strömungsmechanik stellt für diese Bachelorarbeit ein Compute Cloud Projekt mit dem Standardkontingent aus Tabelle 2.1 zur Verfügung.
Bei der Einrichtung einer VM hat man die Auswahl aus verschiedenen Varianten in Bezug auf die Größe. Abgesehen von den in Tabelle 2.2 gelisteten Varianten gibt es noch weitere, die jedoch nur auf Anfrage unter bestimmten Bedingungen verfügbar sind, darunter welche mit GPUs oder deutlich größeren Mengen RAM und mehr vCPUs. Mit dem Standardkontingent lässt sich somit beispielsweise eine einzelne lrz.xlarge VM betreiben oder 2× lrz.large und 1× lrz.medium.
Für die Auswahl des Betriebssystems werden vom LRZ Images von diversen populären Linux-Distributionen bereitgestellt, darunter CentOS, Fedora und Ubuntu. Alternativ hat man die Möglichkeit ISO-Dateien beliebiger Distributionen hochzuladen. In diesem Projekt wurde eine Ubuntu-Installation vorgenommen.
Für die Netzwerkebene stellt das LRZ konfigurierbare Security Groups bei. Diese funktionieren wie eine externe Firewall zur VM, und bestimmen, ob und welche ein- und ausgehenden Verbindungen hergestellt werden dürfen. Das LRZ stellt standardmäßig zwei vordefinierte Netzwerke bereit, zu denen man die VM verbinden kann. Eins für Zugang vom Internet und eins für Zugang vom Münchner Wissenschaftsnetz (MWN) aus. Um von einem persönlichen Computer außerhalb des MWN in das MWN zu gelangen wird ein VPN bereitgestellt. Um die VM über eine fixe IP-Adresse erreichbar zu machen, werden Floating IPs verwendet: Es gibt zwei Pools an Floating IPs, aus denen man der VM MWN-weite bzw. öffentlich erreichbare Adressen zuweisen kann. Solch eine Floating

Tabelle 2.2.: Auswahl an verfügbaren VM Flavors der LRZ Compute Cloud. CPU Overcommit bezeichnet das Verhältnis von virtuellen CPUs zu physischen CPUs. Ein Wert von 8 bedeutet, dass ein physischer CPU bis zu 8 vCPUs bearbeitet [17].

Name	vCPUs	RAM	maximaler CPU Overcommit
lrz.tiny	1	1.12 GB	8
lrz.xsmall	1	2.25 GB	8
lrz.small	1	4.5 GiB	8
lrz.xmedium	2	4.5 GB	8
lrz.medium	2	9 GiB	8
lrz.large	4	18 GiB	8
lrz.xlarge	10	45 GiB	8

IP bleibt einer VM zugewiesen, auch während sie Heruntergefahren ist. Sie kann auch nach Belieben an andere VMs innerhalb eines Compute-Cloud-Projekts vergeben werden. Erst wenn die IP-Adresse explizit freigegeben wird, wird sie in den Pool an Floating IPs zurückgegeben (vgl. [17]).
Außerdem ist es möglich, von einer Compute Cloud VM aus auf die Dateien in einem DSS-Container zuzugreifen. Dazu wird das Network File System (NFS) Protokoll verwendet, um den DSS-Container in der VM zu mounten (siehe Abschnitt 3.1).
Da die LRZ Compute-Cloud nur die Cloud-Infrastruktur bereitstellt, aber ansonsten dem Nutzer die Kontrolle über die Verwendung der Cloud-Instanzen überlässt, folgt sie dem Infrastructure-as-a-Service (IaaS)-Modell (vgl. Abschnitt 1.3).

2.2. ParaView Remote Rendering

2.2.1. ParaView

ParaView ist ein Open-Source Programm zur Visualisierung von Datensätzen, entwickelt vom Unternehmen Kitware. Es ist optimiert auf parallele Datenverarbeitung, kann also die Rechenarbeit auf viele Prozessorkerne, oder im Extremfall sogar auf mehrere Computer verteilen.
ParaView kann als kompiliertes Programm auf der ParaView Homepage [18] heruntergeladen werden. Beim Download der Software für Linux-Systeme werden verschiedene Varianten angeboten. Zunächst gibt es die Standardversion, die in erster Linie für die Verwendung als grafische Desktopanwendung gedacht ist. Gleichzeitig gibt es auf der Downloadseite für Version 5.11 noch zwei verschiedene Varianten für den Betrieb auf einem Server ohne Grafischer Benutzeroberfläche: Eine `egl`-Variante und eine `osmesa`-Variante. Beide Varianten benötigen etwas weniger Speicherplatz, da die Komponente für die grafische Benutzeroberfläche nicht enthalten ist. Der Unterschied in den zwei Varianten ist die verwendete Grafikschnittstelle.

EGL ist eine Spezifikation für eine Grafikschnittstelle, die als Verbindungsstück zwischen dem Desktopfenstersystem und der OpenGL Grafikschnittstelle agiert [19]. Eine konkrete Implementierung der EGL und OpenGL Spezifikationen wird durch den Grafiktreiber des Systems bereitgestellt. Ob dabei Offscreen-Rendering, also Rendering ohne eine grafische Benutzeroberfläche unterstützt wird, ist letztendlich abhängig vom verwendeten Grafiktreiber. NVIDIA unterstützt mit EGL Eye das Offscreen-Rendering auf Linux-Systemen [20]. Für die Verwendung der EGL-Variante ist somit Voraussetzung, dass ein dedizierter Grafikprozessor inklusive Treiber im System installiert ist. Dafür kann dann für die Grafikberechnungen der Visualisierung die schnellere Grafikhardware verwendet werden.
OSMesa ist eine alternative Grafikschnittstelle, die für die Verwendung auf Systemen ohne Grafikhardware und ohne grafische Oberfläche geeignet ist und ist Teil der Mesa 3D Grafikbibliothek. Mesa 3D ist wiederum eine Open Source Implementierung der OpenGL Grafikschnittstelle und ist dazu in der Lage softwarebasiertes Rendering durchzuführen [21]. Dabei werden die notwendigen Berechnungen vom Hauptprozessor übernommen. Da ein Hauptprozessor, anders als die Recheneinheiten von Grafikprozessoren jedoch nicht speziell für die Grafikberechnungen entwickelt wurde, ist softwarebasiertes Rendering üblicherweise langsamer.
Im Rahmen dieser Arbeit steht das Standardkontingent für LRZ-Cloud-Instanzen zur Verfügung, das keinen Grafikprozessor beinhaltet. Deshalb muss auf die OSMesa-Variante zurückgegriffen werden. Das soll aber kein größeres Problem darstellen, da der Fokus mehr auf der Machbarkeit und Umsetzung und weniger auf der letztendlichen Performance liegen soll. Insbesondere kann das Setup in Zukunft jederzeit auf einer anderen Cloud-Instanz *mit* Grafikprozessor durchgeführt werden.
Insgesamt bringt ParaView diverse Features mit sich, die von Nutzen für das Ziel dieser Arbeit sind, darunter auch eine Unterstützung für viele verschiedene Dateiformate. Remote-Rendering ist das im Rahmen dieser Arbeit wichtigste Feature von ParaView, und wird im folgenden genauer betrachtet.

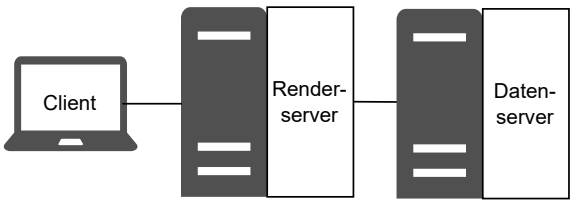
2.2.2. Remote-Visualisierung mit ParaView

ParaViews Rendering-Architektur lässt sich in drei Komponenten unterteilen: Datenserver, Renderserver und Client (vgl. Abbildung 2.1a und [22]): Der Datenserver nimmt den zu visualisierenden Datensatz und wendet die verschiedenen nutzergewählten Filter auf die Daten an. Das Ergebnis dieses Prozesses wird daraufhin an den Renderserver weitergegeben. Bei Anwendung des „Slice“-Filters auf ein dreidimensionales Objekt beispielsweise wäre das Ergebnis eine Scheibe aus dem inneren des Objektes. Aufgabe des Renderservers ist es, dann aus diesen Daten ein Bild zu generieren und dabei das Objekt zum Beispiel anhand enthaltener Messwerte einzufärben. Der Client nimmt das generierte Bild und zeigt es dem Nutzer an. Im Fall von interaktiver Anwendung gibt er alle weiteren Nutzeraktionen (Kamera rotieren, Filter anpassen, Farben anpassen) wieder an Datenserver und Renderserver zurück, damit diese zusammen ein neues Bild generieren können (z.B. aus anderer Perspektive).

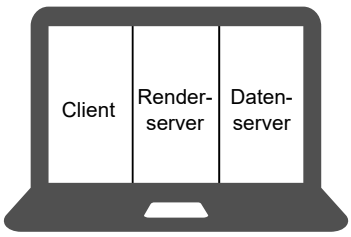
Bei simpler Anwendung von ParaView als Desktopprogramm mit einem lokal gespeichertem Datensatz laufen Datenserver, Renderingserver und Client alle kombiniert auf dem Desktopcomputer. Der Nutzer kann sich dann interaktiv durch die Szene bewegen, die Kamera drehen oder Filter anpassen und alle notwendigen Berechnungen laufen lokal ab.
Im Gegensatz dazu ist es auch möglich den Client auf dem eigenen Desktop auszuführen und Datenserver und Renderserver hingegen auf separater Hardware laufen zu lassen (vgl. Abbildung 2.1c). Die Funktionalität bleibt die gleiche, die Nutzeraktionen im Client und die resultierenden Bilder müssen dafür jedoch über das Netzwerk gesendet werden: Nutzeraktionen vom Client zu Renderserver und Datenserver, die Bilder von Renderserver zu Client. Als Folge bringen alle Formen der Interaktion auch eine Latenz mit sich und die Flüssigkeit des Bildes während einer Kamerabewegung ist durch Latenz und Netzwerkgeschwindigkeit limitiert.
Das Setup aus 2.1c kann jedoch dann nützlich sein, wenn der Datensatz sehr groß ist, da dann nur die resultierenden Bilder gesendet werden müssen, ohne dass der Client jemals den vollen Datensatz erhält. Ein genauer, beispielhafter Vergleich der benötigten Datenmenge ist im Anhang in Abschnitt A.1 beschrieben. Auch wenn der Computer, wo der Client läuft und auf dem der Datensatz betrachtet werden soll, schwache Hardware oder keinen dedizierten Grafikprozessor enthält, und der physische Renderserver hingegen verhältnismäßig leistungsstark ist, kann das Setup sinnvoll sein.
Eine weitere Konfiguration sieht wie folgt aus: Sowohl Client als auch der Rendervorgang werden auf ein und demselben Computer ausgeführt, und nur der Datenserver ist separat (vgl. Abbildung 2.1d). Das kann nützlich sein, wenn der Datensatz besonders groß ist, man aber nur an einem Bruchteil der Daten, beispielsweise einem kleinen Bereich oder einem Querschnitt, interessiert ist, der Client Computer einen angemessen schnellen Grafikprozessor enthält *und* der entfernte Server keine dedizierte Grafikhardware besitzt und sonst auf Softwarerendering zurückgreifen müsste. Nach Anwendung eines entsprechenden ParaView Filters ist die Datenmenge, die letztendlich an den Client gesendet werden muss im Idealfall um ein vielfaches kleiner. Das Rendering wird dabei vom Client übernommen. Als Nebeneffekt verhindert man dabei zusätzlich noch negative Auswirkungen der Netzwerklatenz beim interaktiven Betrachten der Szene, da keine Bilder gestreamt werden müssen.

2.3. ParaView Visualizer

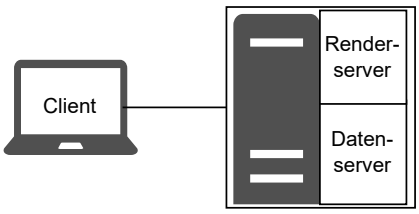
ParaView stellt zudem das *ParaViewWeb*-Framework für Datenverarbeitung und -visualisierung im Browser bereit. Auf diesem Framework aufbauend wurde von Kitware die ParaView Visualizer Webanwendung entwickelt. Sie stellt im Browser eine Benutzeroberfläche vergleichbar mit der ParaView Desktopanwendung bereit. Dabei wird ein ParaView Backend für Datenverarbeitung und Rendering (vgl. Abschnitt 2.2.2) verwendet [23]. Nach Aufruf der Website läuft alle weitere Kommunikation (Bilderstrom, Mausinteraktionen, etc.) über eine WebSocket Verbindung.
ParaView Visualizer stellt Browserseitig verschiedene Konfigurationsoptionen bereit, um



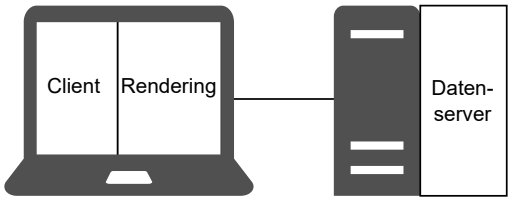
(a) Remote-Rendering mit getrennten Daten- und Renderservern



(b) ParaView als Desktopanwendung



(c) Remote-Rendering mit Daten- und Renderserver auf derselben Hardware



(d) Nur ein Remote-Datenserver, Rendering wird vom Client übernommen

Abbildung 2.1.: Verschiedene Aufteilungsmöglichkeiten der ParaView Komponenten

die Remote Rendering Erfahrung anzupassen. Darunter befinden sich Einstellungen wie maximale Bildrate, Bildqualität oder Bildmaße während Interaktionen. ParaView Visualizer verwendet zum Senden der Einzelbilder das JPEG Format zur Komprimierung. Mit der Bildqualitätseinstellung lässt sich der Qualitätsparameter für die JPEG-Kompression anpassen. Die Einstellung bezieht sich jedoch nur auf die Zwischenbilder innerhalb einer Interaktion, also während beispielsweise mit der Maus die Kamera gedreht wird. Sobald die Kamera still steht, wird ein abschließendes Bild mit hoher Qualität gesendet. Dadurch wird die gesendete Datenmenge während einer Interaktion reduziert, wodurch sich wiederum die Kamera aus Nutzersicht flüssiger und somit auch präziser bewegen lässt, besonders bei langsameren Netzwerkverbindungen.

Das gleiche Ziel verfolgt auch die Einstellung der Bildmaße: In einem Bereich von 0-100 % lässt sich bestimmen, wie hoch und breit die generierten Bilder während einer Interaktion im Vergleich zu Inaktivität sein sollen. Auch hier kann die gesendete Datenmenge drastisch reduziert werden, zusätzlich wird weniger Rechenleistung für die Bildgenerierung benötigt, da weniger Pixel berechnet werden müssen. Das ist bei großen Datensätzen und besonders bei weniger leistungsstarken Renderservern hilfreich. Letztendlich lassen sich dadurch mehr Einzelbilder in der gleichen Zeit erstellen, sodass Kamerabewegungen auch hier wiederum flüssiger wirken. Der Qualitätsverlust ist in beiden Fällen typischerweise akzeptabel, da jedes Einzelbild innerhalb einer Interaktion sowieso nur für einen Bruchteil einer Sekunde angezeigt wird.

Außerdem lässt sich Remote Rendering deaktivieren, woraufhin die gefilterten Daten vollständig an den Browser gesendet werden und der Browser für das Rendering verwendet wird, die resultierende Konfiguration ist vergleichbar mit der letzten in Abschnitt 2.2.2

beschriebenen bzw. in Abbildung 2.1d dargestellten Konfiguration. Diese Funktion ist vor allem bei kleinen Datensätzen sinnvoll, da in so einem Fall die Datenmenge aller Einzelbilder einer Interaktion schnell die Größe des gesamten Datensatzes übersteigen würde. Entsprechend gibt es noch ein weiters Optionsfeld, in dem sich ein Limit in Megabytes angeben lässt, unter dem Remote Rendering automatisch deaktiviert wird.
Beim Starten des Visualizer-Backends über die Kommandozeile gibt es auch noch einige Optionen: Zunächst lässt sich die Netzwerkadresse (IP-Adresse und Portnummer) festlegen, auf der die Webanwendung bereitgestellt wird und die WebSocket Kommunikation abläuft. Außerdem muss ein Verzeichnis angegeben werden, in dem die Dateien liegen, die angezeigt werden sollen, sowie optional ein standardmäßig geöffneter Datensatz. Auch die maximalen Maße der Einzelbilder lassen sich beschränken. Unterhalb dieser Beschränkung wird immer die aktuelle Auflösung des Browserfensters des Clients verwendet.
Eine Einschränkung des ParaView Visualizers ist, dass ein Visualizer-Backend nur mit einem einzelnen Client gleichzeitig kommunizieren kann und nach einer Verbindungstrennung sich selbst beendet. Somit ist weitere Software nötig um Visualizer-Backends nach Bedarf zu starten, sobald sich weitere Clients verbinden möchten.

2.4. Apache-Server

Der Apache-HTTP-Server ist ein Open-Source HTTP-Server entwickelt von der Apache Software Foundation für diverse Betriebssysteme, darunter UNIX und Windows. Er ist neben seiner Alternative nginx einer der meistgenutzten Webserver im Internet. Laut netcrafts „November 2023 Web Server Survey“, ist nginx momentan knapper Marktführer mit einem Marktanteil von 22,83 % aller Websites, ganz knapp vor Apache mit 22,74 % Marktanteil [24]. Der Apache-Server bringt jedoch durch seinen großen Modulkatalog einige Features mit sich, die in diesem Projekt benötigt werden, insbesondere das *rewrite*-Modul [25].
In diesem Projekt soll er in erster Linie als Reverse Proxy eingesetzt werden. Ein Reverse Proxy ist ein Server, der eingehende Anfragen, sowie die korrespondierenden Antworten transparent an andere Server weiterleitet. Das ist beispielsweise nützlich, wenn mehrere verschiedene Services auf einer Adresse (im Sinne von IP und Port) erreichbar sein sollen, wobei typischerweise anhand des URL-Pfads ermittelt wird, für welchen Service eine eingehende Anfrage bestimmt ist. Wie in Abschnitt 2.3 beschrieben, sollen mehrere ParaView Visualizer Instanzen gleichzeitig laufen und verschiedene Nutzer auf ein und derselben Adresse bedient werden. Dabei ist es Aufgabe des Reverse Proxys, die Kommunikation jeweils an die passende ParaView Visualizer Instanz weiterzuleiten.

2.5. wslink Launcher

Die von Kitware entwickelte *wslink* Python-Bibliothek vereinfacht die Kommunikation zwischen einem Python Server und einem JavaScript Client über eine WebSocket-Verbindung, und wird unter anderem von Kitware selbst in ihrem ParaView Web Framework verwen-

Tabelle 2.3.: ParaView Visualizer Launcher API [27].

URL-Pfad	Methode	Anfragenbody	Antwortbody
/paraview	POST	`{sessionManagerURL: 'http://host:port/paraview', application: 'visualizer'}`	`{sessionManagerURL: 'http://host:port/paraview', application: 'visualizer', id: '12345', sessionURL: 'ws://host:port/ws?id=12345'}`
/paraview/12345	GET	-	`{sessionManagerURL: 'http://host:port/paraview', application: 'visualizer'}`
/paraview/12345	DELETE	-	`{sessionManagerURL: 'http://host:port/paraview', application: 'visualizer'}`

det [26]. Gleichzeitig enthält die Bibliothek unter anderem eine Implementierung eines *Launchers* für die ParaView Visualizer Instanzen. Der Launcher ist out-of-the-box bereits an den von ParaView Web verwendeten Kommunikationsablauf angepasst und unterstützt somit bereits einen simplen Multi-User Anwendungsfall.
Die vom Launcher bereitgestellte API ist in Tabelle 2.3 dargestellt [27]. Sobald im Browser die Website geladen wurde, wird von der ParaView Visualizer Anwendung eine POST-Anfrage an den `/paraview`-Pfad gesendet. Der Launcher startet dann das ParaView Visualizer Backend und gibt in `sessionURL` eine URL zurück, unter der der WebSocket Server des gerade gestarteten Backends erreichbar ist. Das Frontend baut dann mit dieser Adresse eine WebSocket-Verbindung auf.
Der wslink Launcher lässt sich über eine JSON-Datei konfigurieren, die genaue Konfiguration wird in Abschnitt 3.2 beschrieben.

3. Data Sharing & Remote Rendering auf der LRZ Compute Cloud

3.1. Mounten eines DSS-Containers auf der LRZ-Compute-Cloud

Der erste Schritt auf dem Weg zur Umsetzung ist zunächst, die LRZ-Cloud mit dem DSS-Container zu verbinden, unter Verwendung des NFS-Protokolls.
Das NFS (Network File System) Protokoll ermöglicht einem Client auf Dateien über ein Netzwerk zuzugreifen, so als ob sie Teil des lokalen Dateisystems wären. In der Konfiguration des NFS-Servers müssen dabei IP-Adressen von Clients festgelegt werden, die auf freigegebene Dateien zugreifen dürfen. Aus diesem Grund wird auch in der LRZ-Dokumentation gewarnt, dass der DSS-Container nur für statische IP-Adressen freigegeben werden soll, bei denen der zugrundeliegende Client vertrauenswürdig ist. Die Floating-IP-Adressen der LRZ-Compute-Cloud erfüllen diese Anforderung, sofern sie nicht wieder freigegeben werden. Im Fall, dass die Floating-IP-Adresse doch irgendwann freigegeben werden soll, sollte davor DSS-seitig der NFS-Export an diese Adresse wieder aufgehoben werden.
Um den DSS-Container auf der Compute-Cloud mounten zu können muss also als Erstes der Compute-Cloud-Instanz eine Floating-IP zugewiesen werden. Anschließend muss ein Datenkurator des DSS-Projekts über die DSS-Weboberfläche den NFS-Export an die IP-Adresse aktivieren. Dabei hat man die Wahl, ob dem Client nur Lesezugriff, oder Lese- und Schreibzugriff ermöglicht werden soll. Für die Zwecke dieser Arbeit ist Lesezugriff ausreichend. Ist das geschehen, wird dem Datenkurator in der Weboberfläche ein Mountpfad angezeigt, der vom Client zum mounten des Dateisystems verwendet werden kann. Dafür muss zunächst auf der Compute-Cloud-Instanz ein leeres Verzeichnis, z.b. `/dss` angelegt werden, in dann das DSS-Dateisystem eingehängt werden kann. Der Befehl dafür lautet:

```
mount -t nfs -o ro,rsize=1048576,wsize=1048576,hard,tcp,bg,timeo=600,vers=3
    ↪ <mount-pfad> /dss
```

Hinter `-t` wird die Art des Dateisystems angegeben, hinter `-o` eine Liste von Optionen. Für die hier gezeigten Optionen wurden die empfohlenen Werte aus der Dokumentation [28] übernommen und zusätzlich `ro` eingefügt. `rsize` und `wsize` beispielsweise bestimmen die Maximalgröße in Bytes der einzelnen Lese- bzw. Schreibpakete. Je größer dieser Wert, desto weniger Pakete müssen gesendet werden, wodurch der Overhead reduziert und somit die Performance erhöht werden kann. Der angegebene Wert entspricht hierbei 1 MiB,

dem maximal zulässigen Wert. `ro` legt Cloud-Instanz-seitig erneut read-only Zugriff fest. Selbst wenn der Datenkurator Schreibzugriff erlaubt hätte, könnten somit trotzdem keine Schreibzugriffe durchgeführt werden. Um das DSS-Dateisystem über Systemstarts hinweg zu mounten muss noch zusätzlich ein passender Eintrag in `/etc/fstab` angelegt werden:

```
<mount-path>  /dss  nfs  ro,rsize=1048576,wsize=1048576,hard,tcp,bg,timeo=600,vers
    ↪ =3 0 0
```

Nachdem der DSS-Container erfolgreich gemountet wurde, wird man feststellen, dass grundsätzlich überhaupt kein Zugriff auf die eingehängten Daten möglich ist. Jeder Versuch, das `/dss`-Verzeichnis zu betreten, wird mit der Meldung „Permission denied“ abgelehnt. Das liegt daran, dass NFS erwartet, dass alle Linux-Nutzerkonten auf allen verbundenen Systemen die gleichen Nutzer-IDs (UID) haben.
Wenn also eine Datei auf dem DSS-Dateisystem beispielsweise einem DSS-Nutzer mit der Kennung „abc12xyz“ und der UID 12345432 gehört, dann erlaubt der NFS-Server einem Nutzer auf dem Client nur dann Zugriff auf die Datei, wenn der Nutzer auf der Client-Seite „abc12xyz“ heißt und 12345432 als UID hat. Deswegen wird dem Standardnutzer (UID 1000 auf Ubuntu) der Zugriff verweigert. Normalerweise könnte der Root-User unabhängig von den gesetzten Dateiattributen trotzdem auf Dateien zugreifen, das LRZ verwendet jedoch beim NFS-Export die `root_squash` Option von NFS [29]. Mit dieser wird festgelegt, dass alle Zugriffe von Root (UID 0) so umgeschrieben werden, als kämen sie von einem anderen Nutzer (standardmäßig UID 65534) ohne besondere Privilegien. Somit kann auch Root nicht direkt auf die Dateien zugreifen.
Jeder Nutzer auf dem DSS-Container der Forschungsgruppe besitzt einen eigenen Ordner. Damit auf diese Ordner zugegriffen werden kann, muss für jeden DSS-Nutzer auch eine Kopie des Nutzers mit identischer UID auf der Compute-Cloud angelegt werden [28, 29]. Eine weitere Implikation ist, dass auch das Visualisierungsprogramm selbst zunächst nicht auf die Nutzerdaten zugreifen kann, weswegen es (z.B. mittels `sudo -u`) explizit mit den Rechten genau des Nutzers aufgerufen werden muss, auf dessen Dateien zugegriffen werden soll. Wie genau das Rechtemanagement letztendlich umgesetzt wurde, wird in Abschnitt 3.4.4 beschrieben.

3.2. Ausgangspunkt

Kitware beschreibt in der Dokumentation von ParaView Web [30] bereits ein simples Multi-User-Setup für eine ParaView Web Anwendung. *Multi-User* dabei in dem Sinne, dass sich mehrere Nutzer gleichzeitig über den Browser die Datensätze anzeigen lassen können. Bevor das eigentliche Ziel dieser Arbeit umgesetzt wird, also die Funktion, dass mehrere Nutzer des LRZ jeweils eigene Datensätze separat von den anderen Nutzern teilen können, soll zunächst ein ParaView Visualizer Setup umgesetzt werden, das auf Kitwares Dokumentation basiert. In der Dokumentation wird der Apache-Server aus Abschnitt 2.4 als Reverse Proxy verwendet (vgl. [31]), und der wslink Launcher aus Abschnitt 2.5 als Launcher für die einzelnen Visualisierungsprozesse im Backend (vgl. [32]). Die Interaktion

zwischen den Komponenten funktioniert dann wie folgt, unterstützend sind die einzelnen Schritte auch noch in Abbildung 3.1 dargestellt.

1. Der Apache-Server hört auf Port 80 (Standard HTTP-Port) auf einkommende Anfragen und ist vom Internet erreichbar. Auf dem gleichen Server läuft ein wslink-Launcher, der beispielsweise auf Port 9000 auf Anfragen hört. Anders als der Apache-Server ist dieser nicht von außerhalb erreichbar.

2. Der Nutzer ruft die Website beispielsweise auf `http://example.com/` auf.

3. Der Apache-Server ist so konfiguriert, dass bei Anfragen auf das Wurzelverzeichnis `/` die HTML-, JavaScript- und CSS-Dateien bereitgestellt werden, die das Browser-Frontend ausmachen.

4. Sobald der Browser diese Ressourcen geladen hat und damit beginnt, den JavaScript Code auszuführen, wird von ParaView Visualizer eine POST-Anfrage an `/paraview/` gesendet, mit `{application: "visualizer", ...}` als JSON-formatiertem Rumpfteil.

5. Anfragen an den `/paraview/` Pfad leitet der Apache-Server hingegen in seiner Funktion als Reverse Proxy an die Launcher-Instanz weiter, die lokal auf Port 9000 läuft, also an `http://localhost:9000/paraview/`. Die POST-Anfrage aus Schritt 4 erreicht somit den Launcher.

6. Der Launcher generiert daraufhin eine neue Sitzungs-ID und sucht nach einem freien Port in einem definierten Bereich. Anschließend sucht er in seiner Konfigurationsdatei nach einer registrierten Anwendung namens `"visualizer"` (aus dem POST-Rumpf), und führt den zugehörigen Startbefehl in einem neuen Prozess aus. Dabei wird der gefundene freie Port dem Programm als Befehlsargument übergeben. Somit wurde eine ParaView Visualizer-Instanz gestartet, die über den Port erreichbar ist.

7. Die Sitzungs-ID wird zusammen mit Host (der Hostname ist bei diesem Setup immer `localhost`) und Port in eine neue Zeile einer Datei (`proxy.txt`) geschrieben. Diese Information ist später für den Apache-Server relevant.

8. Sobald die Visualizer-Instanz bereit für eingehende Anfragen ist, antwortet der Launcher auf die POST-Anfrage mit einem JSON-Dokument. Dieses ist vergleichbar mit der Antwort aus Zeile 1 der Tabelle 2.3 und enthält unter anderem Sitzungs-ID (`"id"`) und Sitzungs-URL (`"sessionURL"`). In dem Sitzungs-URL-Feld ist die Adresse angegeben, unter der die Anwendung, die in Schritt 6 gestartet wurde, per WebSocket erreichbar ist. Im Fall einer Sitzungs-ID mit dem Wert 12345 lautet die Adresse beispielsweise `ws://example.com/ws?sessionId=12345`.

9. Die Browseranwendung empfängt diese Antwort und startet den WebSocket Verbindungsaufbau mit der Sitzungs-URL als Ziel.

10. Die WebSocket Verbindung erreicht zunächst den Apache-Server, der aber noch einen weiteren Konfigurationseintrag mit einer komplexeren Reverse Proxy Funktionalität

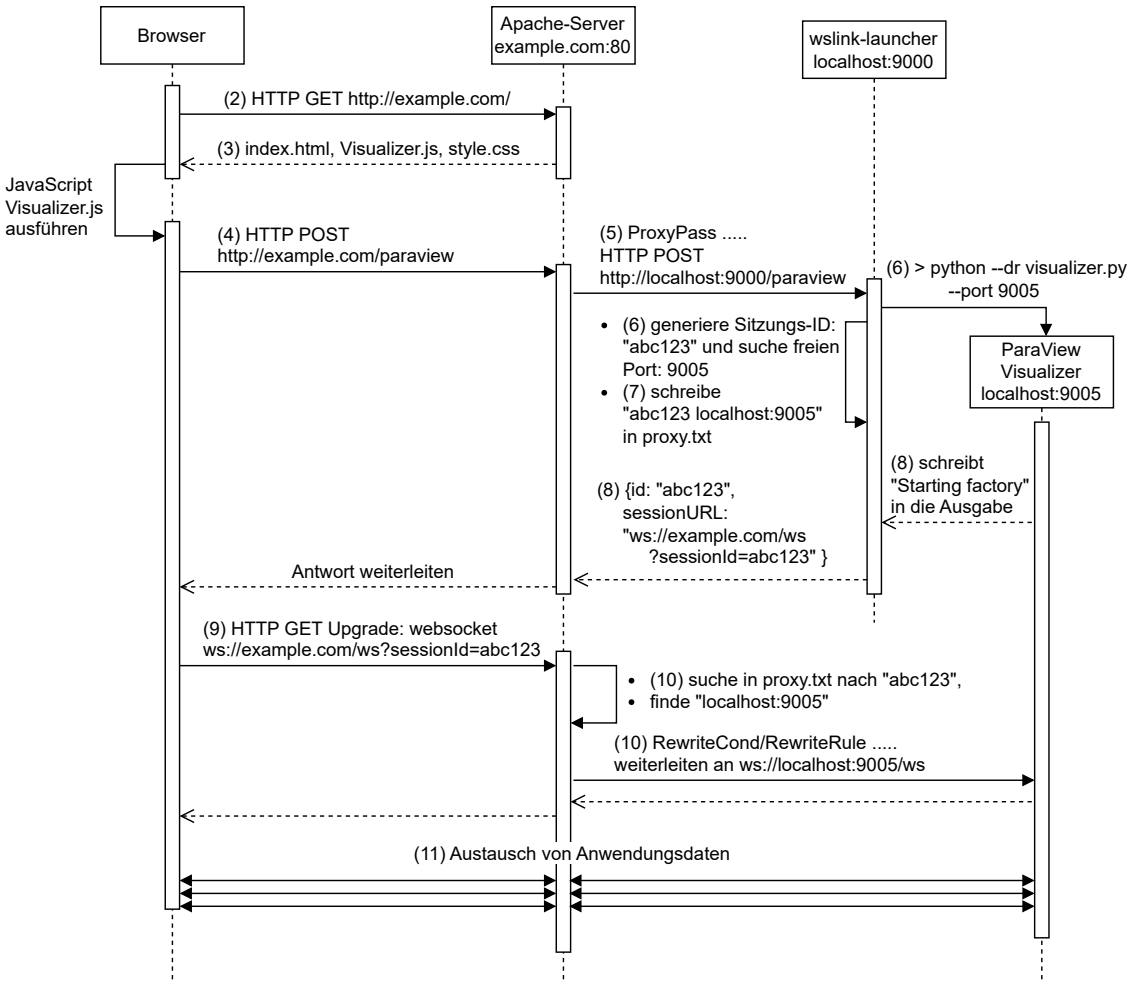


Abbildung 3.1.: Der in Abschnitt 3.2 beschriebene Kommunikationsablauf bei Aufruf der Website. Zahlen in Klammern beziehen sich auf den entsprechenden Schritt in 3.2

hat. Sobald eine Anfrage unter dem `/ws` Pfad eingeht, extrahiert Apache den Wert des `sessionId` Anfrageparameters aus der URL (12345 im Beispiel) und sucht sich in der `proxy.txt` Datei aus Schritt 7 die korrespondierende Host-Port-Kombination. Die WebSocket Verbindung wird dann genau an diese Visualizer-Instanz weitergeleitet.

11. Schließlich werden über diese Verbindung, wie in Abschnitt 2.2.2 beschrieben, die Nutzerinteraktionen in die eine Richtung, die Visualisierungsergebnisse in die andere gesendet.

```
1 import os
2 from wslink import launcher
3
4 # Löst Probleme die beim Lesen der Programmausgabe der Visualizer-Instanzen
      auftreten
5 os.environ["PYTHONUNBUFFERED"] = "1"
6
7 launcher.start()
```

Listing 3.1: Das Wrapper-Skript, das den Launcher aus der wslink-Bibliothek startet

3.2.1. Beispielkonfiguration des wslink-Launchers

Da der wslink-Launcher ein Teil einer Python-Bibliothek ist, muss zunächst ein Wrapper-Skript geschrieben werden, das die Bibliothek verwendet und die Launcher-Komponente ausführt (siehe Listing 3.1). Dieses wslink-Launcher Programm erwartet einen Dateipfad zu einer Konfigurationsdatei als Programmargument:

```
$ python ./path/to/launcher.py ./path/to/config.json
```

Zunächst wurde eine Konfigurationsdatei für den wslink-Launcher verfasst. Dabei wurde sich am Beispiel aus der ParaViewWeb Dokumentation orientiert. Das Ergebnis ist in Listing 3.2 dargestellt. **Diese Konfiguration, sowie das Beispiel aus der Dokumentation besitzen eine potenzielle Sicherheitslücke, und sollten so nicht direkt verwendet werden. Für genaueres, siehe Abschnitt 3.3.** Die einzelnen Einstellungen haben dabei folgende Bedeutungen:

- Der Launcher unterstützt die Verwendung von Platzhaltern in der Form `${<Bezeichner>}`. Die einzusetzenden Werte können dabei einerseits über das `properties` Objekt in Zeilen 17–23 festgelegt werden, andererseits gibt es auch vordefinierte Platzhalter, deren Werte vom wslink-Launcher selbst dynamisch bestimmt werden.
- In Zeilen 3 und 4 wird bestimmt, auf welchem Netzwerkinterface und welchem Port der Launcher auf Anfragen hören soll. Als Port kann ein beliebiger freier Port gewählt werden, im Host-Feld wird `localhost` bzw. `127.0.0.1` angegeben, da der Launcher sowieso nur vom lokal laufenden Apache-Server angesprochen wird.
- Mit `proxy_file` in Zeile 5 wird der Dateipfad für die Datei festgelegt, in die der Launcher die (Session-ID, Host, Port)-Tupel schreiben soll (vgl. Schritt 7).
- `endpoint` gibt an, an welchem Pfad der Launcher die Anfragen aus Tabelle 2.3 bzw. Schritt 4 erwartet.
- Bei `sessionURL` in Zeile 7 wird die URL angegeben, unter der eine einzelne gestartete Visualisierungsinstanz vom Browser aus erreichbar sein wird, damit der Launcher diese Information als Antwort auf die anfängliche POST-Anfrage senden kann. Der `${id}` Platzhalter ist ein Beispiel für einen von wslink bereitgestellten Platzhalter. Er

```
{
    "configuration": {
        "host": "localhost",
        "port": 9000,
        "proxy_file": "/srv/launcher/proxy.txt"
        "endpoint": "paraview",
        "sessionURL": "ws://example.com/ws?sessionId=${id}",
        "fields": [],
        "timeout": 5,
        "log_dir": "/var/log/launcher"
    },

    "resources": [
        {"host": "localhost", "port_range": [9001, 9003]}
    ],

    "properties": {
        "python_exec": "/opt/paraview/bin/pvpython",
        "visualizer_exec":
           "/opt/paraview/share/paraview-5.11/web/visualizer/server/pv-visualizer.py",
        "data_dir": "/srv/simulationfiles/",
        "default_file": "examplefile.dat"
    },

    "apps": {
        "visualizer": {
            "cmd": ["${python_exec}", "--dr", "${visualizer_exec}", "--port",
                "${port}", "--data", "${data_dir}", "--load-file",
                "${default_file}"],
            "ready_line": "Starting factory"
        }
    }
}
```

Listing 3.2: Eine Beispielkonfiguration für den wslink-Launcher, angelehnt an das Beispiel der ParaViewWeb Dokumentation, für ein einfaches Multi-User-Setup wie in Abschnitt 3.2 beschrieben. **Die Konfiguration besitzt in dieser Form jedoch eine potenzielle Sicherheitslücke und sollte so nicht verwendet werden.**

wird hierbei individuell für jede Instanz mit der Sitzungs-ID ersetzt, die in Schritt 6 generiert wurde.

- In Zeile 8 können Bezeichner von Platzhaltern angegeben werden. Die Werte der dort gelisteten Platzhalter werden daraufhin in der Antwort auf die POST-Anfrage verwendet, und ergänzen die Standardmäßig enthaltenen Attribute (`sessionManagerURL`, `id` und `sessionURL`). Es können sowohl Platzhalter aus dem `properties` Block verwendet werden, als auch Platzhalter, die vom Launcher bereitgestellt werden.

- `timeout` (Zeile 9) legt fest, wie viele Sekunden nach Start des Visualisierungsprozesses darauf gewartet werden soll, dass dieser bereit ist Anfragen anzunehmen. Ist die Zeit verstrichen, ohne dass der Visualisierungsprozess rechtzeitig bereit für Anfragen war, wird dem Browser eine Fehlermeldung als Antwort auf die POST-Anfrage übermittelt.

- `log_dir` (Zeile 10) gibt das Verzeichnis an, in dem die Logdateien des Launchers und der einzelnen Visualisierungsinstanzen gespeichert werden sollen.

- Wenn der Launcher einen neuen Prozess starten soll, muss er sich eine freie Portnummer suchen, auf der die Anwendung anschließend erreichbar sein soll. Mit `resources` in Zeilen 13–15 können Portnummernbereiche angegeben werden, die für die Instanzen verwendet werden sollen. Der Launcher merkt sich, welche Instanz unter welcher Portnummer erreichbar ist und vergibt noch verfügbare Portnummern an neue Instanzen. Verfügbar bedeutet in diesem Fall nur, dass der Launcher diese Portnummer momentan nicht vergeben hat. Im Umkehrschluss bedeutet das, dass nicht geprüft wird, ob ein anderes Programm diesen Port verwendet. Falls das der Fall ist, kann die neu gestartete Visualisierungsinstanz den Socket nicht an den zugewiesenen Port binden und wird somit beim Start fehlschlagen.

- Unter dem `apps` Konfigurationspunkt ab Zeile 25 werden die Anwendungen festgelegt, die der Launcher starten soll. Bei eingehender POST-Anfrage zum Starten einer Anwendung nimmt der Launcher den Wert aus dem `application` Feld des Anfragenbodys (vgl. Tabelle 2.3) und sucht nach einem identisch benannten Eintrag in der Konfiguration. Daraufhin wird der Befehl ausgeführt, der in `cmd` festgelegt wurde. Er wird dabei als Liste von Programmargumenten festgelegt, wobei das erste Element der Liste die auszuführende Programmdatei ist.

  Im Beispiel werden die Platzhalter aus dem `properties` Block verwendet, sowie ein vom Launcher bereitgestellter `${port}` Platzhalter. Letzterer wird dabei mit der Portnummer ersetzt, die im vorherigen Punkt ermittelten wurde, sodass jede gestartete Instanz ihren Socket an einen anderen Port bindet.

  Einige Werte im Startbefehl sind von der Systemumgebung abhängig, beispielsweise die Dateipfade von diversen Programmen oder die Verzeichnispfade mit den Simulationsdaten. Die Verwendung von Platzhaltern ist hierbei sinnvoll, da dadurch diese

Werte lokalisiert an einem Punkt in der Konfigurationsdatei sind und gleichzeitig getrennt sind von der Struktur des Befehls. Wenn also ein solches Multi-User-Setup auf einem anderen System eingerichtet werden soll, müssen nur die Werte im `properties` Block angepasst werden.

3.2.2. Beispielkonfiguration des Apache-Servers

Im Folgenden soll eine beispielhafte Konfiguration des Apache-Servers beschrieben werden, die auf der Kitware Dokumentation basiert. Zusammen mit der Launcher-Konfiguration aus 3.2.1 realisiert sie das Verhalten, das in den Schritten 3, 4 und 10 aus Abschnitt 3.2 beschrieben wurde.
Konfigurationsdateien sind bei Apache-Servern unter Debian (und Derivaten wie Ubuntu) typischerweise so strukturiert, dass das Hinzufügen und Entfernen bzw. das Aktivieren und Deaktivieren von einzelnen Modulen und Einstellungsgruppen möglichst flexibel durchgeführt werden kann. Dazu wird folgende Ordnerstruktur verwendet:

```
/etc/apache2/
|-- apache2.conf
|-- ports.conf
|-- mods-available/
|       |-- *.load
|       `-- *.conf
|-- mods-enabled/
|-- conf-available/
|       `-- *.conf
|-- conf-enabled/
|-- sites-available/
|       `-- *.conf
`-- sites-enabled/
```

`apache2.conf` ist die Hauptkonfigurationsdatei, die alle anderen Dateien mittels `Include`-Direktiven zusammenführt. In `ports.conf` wird mittels `Listens`-Direktiven festgelegt auf welchen Portnummern grundsätzlich auf eingehende Anfragen gehört werden soll. Für das HTTP Protokoll wird dabei standardmäßig der Port 80 verwendet. Welcher Port auch immer hier angegeben wird, am Ende muss im LRZ-Webinterface ein Security Group Eintrag angelegt werden, der eingehende Verbindungen auf diesem Port zulässt. `mods-available` ist ein Verzeichnis, in dem für alle verfügbaren Module jeweils eine *`modulname`*`.load` Datei liegt. Diese Dateien enthalten wiederum übereinstimmende `LoadModule`-Direktiven mit Dateipfaden zu den Binärdateien der jeweiligen Module. Zusätzlich kann für jedes Modul noch über jeweils eine *`modulname`*`.conf` Datei modulspezifische Einstellungen vorgenommen werden. Letztendlich lädt die Hauptkonfigurationsdatei jedoch nicht die Dateien aus `mods-available`, sondern die aus `mods-enabled`. Mittels der Kommandozeilentools `a2enmod` und `a2dismod`, die vom Apache-Server bereitgestellt werden, können einzelne Module aus `mods-available` aktiviert bzw. deaktiviert werden. Dabei wird in `mods-enabled` eine symbolische Verknüpfung auf die `*.load` Datei und die `*.conf` Datei aus dem `mods-available` Verzeichnis erstellt, wodurch die Konfiguration in `apache2.conf` geladen wird.

Apache hat ein „Virtual Host“ Feature, mit dem über ein und denselben Apache-Server unterschiedliche Inhalte bereitgestellt werden können, abhängig von dem Hostnamen der bei der Anfrage verwendet wird. Konfigurationen für die einzelnen Virtual Hosts werden in `sites-available` erstellt und nach dem gleichen Prinzip wie bei den Modulen aktiviert und deaktiviert (die Tools nennen sich analog `a2ensite` und `a2dissite`).
Konfigurationsdateien mit Einstellungen, die den gesamten Apache-Server über Virtual Hosts hinweg betreffen, können in `conf-available` abgelegt werden. Das Prinzip zum Aktivieren und Deaktivieren ist auch hier identisch.
Zunächst wird die Konfiguration als Grundlage verwendet, die nach einer Installation per APT Paketverwaltungssystem auf Ubuntu vorliegt. Um das Verhalten zu erzielen, das beim Multi-User-System erwünscht ist, muss ergänzend eine Konfigurationsdatei (im Beispiel als `paraview-multi-user.conf` benannt) in `conf-available` angelegt und mit dem Inhalt aus Listing 3.3 gefüllt werden. Anschließend muss die Konfigurationsdatei mittels

```
$ a2enconf paraview-multi-user
```

aktiviert werden. Die Konfigurationsdatei selbst ist wie folgt aufgebaut:

- Auch der Apache-Server unterstützt die Verwendung von Variablen in der Konfiguration, ähnlich wie beim wslink Launcher werden die Variablen mittels `${VARIABLENNAME}` eingesetzt. Zunächst werden in den Zeilen 1 und 4 zwei Variablen definiert, die systemabhängige Dateipfade definieren. Der Grund für die Verwendung der Variablen ist der gleiche wie bei den Platzhaltern im wslink Launcher: Dateipfade und andere Werte wie der Hostname, die eventuell von System zu System unterschiedlich sind oder auch beliebig gewählt werden können, befinden sich lokalisiert an einem Punkt und für Anpassungen müssen die eigentlichen Konfigurationsanweisungen nicht modifiziert werden.

  `PARAVIEW_HTML_ROOT` bestimmt dabei den Verzeichnispfad, in dem die Dateien für das Frontend der Anwendung liegen. Diese Dateien sollen dem Browser beim Aufruf der Seite bereitgestellt werden (vgl. Schritt 3). Die relevanten Dateien des ParaView Visualizers liegen vom ParaView Installationsverzeichnis aus gesehen unter folgendem relativen Ordnerpfad: `./share/paraview-5.11/web/visualizer/www/.`
  In der Praxis wurde, um alle für den Visualisierungsservice relevanten Dateien im `/srv` Verzeichnis bereitzustellen, eine symbolische Verknüpfung

  ```
  /srv/visualizer-www --> /opt/paraview/share/paraview-5.11/web/visualizer/www/
  ```

  erstellt, wobei der ParaView Download in `/opt/paraview` installiert wurde.

  `SESSION_PROXY_FILE` enthält den Pfad zu der Datei, in die der wslink-Launcher in Schritt 7 die Sitzungs-ID schreibt. Der hier angegebene Dateipfad muss sich auf dieselbe Datei beziehen, die in der wslink-Konfigurationsdatei (Listing 3.2) unter dem Punkt `proxy_file` angegeben ist.

  In `LAUNCHER_PORT` wird angegeben, auf welcher lokalen Portnummer der wslink-Launcher auf Anfragen wartet. Der Wert hier muss mit `port` der wslink-Konfiguration übereinstimmen.

```
Define PARAVIEW_HTML_ROOT /srv/visualizer-www
Define SESSION_PROXY_FILE /srv/launcher/proxy.txt
Define LAUNCHER_PORT 9000
Define SERVER_NAME 123.111.222.233

# Hostname des Servers
ServerName ${SERVER_NAME}

DocumentRoot ${PARAVIEW_HTML_ROOT}

# Reverse Proxy für wslink Launcher
ProxyPass /paraview http://localhost:${LAUNCHER_PORT}/paraview

# Rewrite setup für ParaViewWeb
RewriteEngine On

# Pfad zur vom wslink Launcher erstellten Mapping Datei (proxy.txt)
RewriteMap session-to-port "txt:${SESSION_PROXY_FILE}"

# Reverse Proxy Regeln für ParaViewWeb
RewriteCond %{QUERY_STRING}  ^sessionId=(.*)$ [NC]
RewriteRule ^/ws$ ws://${session-to-port:%1}/ws [P]

<Directory "${PARAVIEW_HTML_ROOT}">
    AllowOverride None
    Require all granted
</Directory>
```

Listing 3.3: Eine Beispielkonfiguration für den Apache-Server um ein Verhalten, wie in Abschnitt 3.2 beschrieben, umzusetzen.

In SERVER_NAME, bzw. in der ServerName Direktive in Zeile 7 muss der Hostname oder alternativ die IP-Adresse angegeben werden, unter der der Apache-Server vom außerhalb erreichbar ist. Der dort angegebene Wert wird von Apache unter anderem für HTTP-Weiterleitungen verwendet. ParaView Web nutzt zwar momentan keine solchen Redirects, dennoch ist es sinnvoll diesen Wert anzugeben, um auch in Zukunft eine robuste Konfiguration zu haben.

- DocumentRoot in Zeile 9 gibt an, welches Verzeichnis grundsätzlich bereitgestellt wird. Hier wird der Ordnerpfad zu den Frontend-Dateien des ParaView Visualizers angegeben. Für alle Anfragen, die nicht durch eine andere, spezialisiertere Direktive behandelt werden, hängt Apache den URL-Pfad der Anfrage an den Pfad an, der in DocumentRoot angegeben ist, um den lokalen Dateipfad des angefragten Dokuments zu ermitteln.

- Die ProxyPass Anweisung in Zeile 12 wird vom mod_proxy Modul bereitgestellt [33] und realisiert das in Schritt 4 beschriebene Reverse Proxy Verhalten: Jede Anfrage auf dem /paraview Pfad wird an den lokal laufenden wslink-Launcher weitergeleitet. Hierbei wird die oben definierte Portnummer verwendet. Der /paraview-Teil am Ende der Launcheradresse (zweiter ProxyPass Parameter) entspricht hierbei dem endpoint-Wert aus der Launcherkonfiguration.

- RewriteEngine On aktiviert die Rewriting-Engine des mod_rewrite Moduls im aktuellen Konfigurationskontext (in diesem Fall globale Einstellungen). Das Modul ist in [25] dokumentiert. Es ist dazu in der Lage, die URLs eingehender Anfragen nach festgelegten Regeln umzuschreiben. Das Ergebnis dieser Operation kann entweder ein Dateipfad sein, der bereitgestellt werden soll, eine andere URL sein, an die der Browser mittels HTTP-Redirect weitergeleitet werden soll, oder eine URL zu einem anderen Dienst, an den die eingegangene Anfrage weitergeleitet werden soll. Durch Letzteres kann der Apache-Server als Reverse Proxy fungieren.

- Mit RewriteMap wird zunächst eine Abbildungsfunktion für Key-Value-Abfragen definiert. Die Funktion wird in diesem Fall in Zeile 18 session-to-port genannt und verwendet (durch txt: festgelegt) eine Textdatei als Quelle. Darin wird eine Liste mit leerzeichengetrennten Schlüssel-Wert-Paaren erwartet, wobei jedes Paar auf einer eigenen Zeile steht. Eine Datei in genau diesem Format wird vom wslink Launcher für die Sitzungs-IDs produziert. Der Dateipfad der Datei wird hinter dem txt: erwartet. Nach Einsetzen der hier verwendeten SESSION_PROXY_FILE Variable ist somit txt:/srv/launcher/proxy.txt, die Quelle für die Abbildungsfunktion. Dies entspricht dem gleichen Pfad wie in der Konfigurationsdatei des wslink-Launchers (Listing 3.2, Zeile 5)

- Die eigentlichen Rewrite-Regeln werden mit RewriteCond (Zeile 21) und RewriteRule (Zeile 22) festgelegt. RewriteCond definiert zunächst eine Bedingung, die erfüllt werden muss, damit diese URL von der nächsten darauffolgenden RewriteRule umgeschrieben wird. Es können auch mehrere RewriteCond untereinander angegeben werden, die

dann alle erfüllt werden müssen. Eine `RewriteCond`-Anweisung hat folgende Form: `RewriteCond` *`TestString CondPattern`* `[`*`Flags`*`]`. `TestString` ist dabei der Text, der mit der Bedingung in `CondPattern` verglichen werden soll. Primär ist `CondPattern` ein regulärer Ausdruck (Abk. RegEx für *regular Expression*), zusätzlich werden aber auch unter anderem Vergleichsoperatoren für numerische oder alphabetische Vergleiche bereitgestellt, oder auch solche, die die Existenz einer Datei prüfen. Bei den `Flags` gibt es auch einige Optionen: Mit dem `[OR]`-Flag muss im Fall mehrerer `RewriteCond`-Anweisungen die mit `[OR]` versehene *oder* die darauffolgende Bedingung erfüllt werden, anstatt dass wie standardmäßig alle Bedingungen erfüllt sein müssen. Das `[NC]`-Flag gibt an, dass beim Vergleich von `TestString` und `CondPattern` Groß- und Kleinschreibung ignoriert werden soll.

Die darauffolgende `RewriteRule`-Anweisung in der Form
`RewriteRule` *`Pattern Substitution`* `[`*`Flags`*`]`
prüft anschließend, ob der reine Pfadteil der URL, also ohne Hostname und Query-Komponente, mit dem RegEx in `Pattern` übereinstimmt. Wenn ja, wird er mit dem Wert in `Substitution` ersetzt. Falls dieser Wert ein relativer URL-Pfad ist, wird der ursprüngliche Pfadteil ersetzt und die Anfrage passend zum neuen Pfad behandelt. Der `Substitution` Wert darf jedoch auch eine absolute URL zu einem anderen Host sein, in dem Fall wird standardmäßig ein HTTP-Redirect zur Zieladresse durchgeführt. In beiden Fällen wird standardmäßig, falls die Zieladresse keine eigene beinhaltet, die Query-Komponente der eingehenden Anfrage beibehalten.

Wie auch bei `RewriteCond` erlauben eine Reihe von `Flags`, das Standardverhalten anzupassen. Ein `[P]`-Flag sorgt beispielsweise dafür, dass Anfragen anstatt per HTTP-Redirect stattdessen proxymäßig an den angegebenen Host weitergeleitet werden. Ein weiteres Beispiel ist das `[QSD]`-Flag, das die Query-Komponente entfernt, anstatt sie an das Ziel anzuhängen.

Sowohl im `TestString` einer `RewriteCond` als auch im `Substitution`-Teil einer `RewriteRule` Anweisung werden zusätzliche Variablen unterstützt. Sie können z.B. verschiedene Informationen über die Anfrage liefern. So kann `%{QUERY_STRING}` für die Query-Komponente der Anfrage, oder `%{HTTP:`*`ein-header`*`}` für die Werte beliebiger HTTP-Header verwendet werden. Man beachte die `%`-Symbole statt dem `$`-Symbol bei normalen Variablen. Es können auch Gruppierungen aus den regulären Ausdrücken referenziert werden, die in `RewriteCond` oder `RewriteRule` verwendet wurden. Sie werden dann mittels `%1`–`%9` (bei `RewriteCond`) bzw. `$1`–`$9` (bei `RewriteRule`) referenziert.

Besonders wichtig für den hier ausgeführten Anwendungsfall ist die Möglichkeit, die Abbildungsfunktion aufzurufen, die zuvor per `RewriteMap` definiert wurde: Mittels `${`*`map-name`*`:`*`schlüssel`*`}` kann der Wert aus der Abbildung `map-name` verwendet werden, der zu einem `schlüssel` gehört.

- Die Rewrite-Regeln aus Zeilen 21 und 22 führen also bei eingehenden Anfragen folgenden Prozess aus: Zunächst wird in `RewriteCond` überprüft, ob die Query-Komponente der Form `sessionId=(.*)` entspricht, wobei Groß-/Kleinschreibung ignoriert werden

soll. Anstelle `(.*)` dürfen dabei beliebige Zeichen stehen. Durch den Gruppierungsausdruck `(.*)` in der RegEx kann mittels `%1` in darauffolgenden Regeln auf den Wert an dieser Stelle zurückgegriffen werden. Dieser Wert ist dabei der Wert des `sessionId` Query-Parameters, also die Sitzungs-ID. Falls die Bedingung aus `RewriteCond` zutrifft, wird in `RewriteRule` geprüft, ob die Anfrage an den `/ws` Pfad gerichtet ist. Falls ja, wird per `${session-to-port:%1}` aus der Abbildungsfunktion in Zeile 18 der Host abgefragt, der zur Sitzungs-ID (in `%1`) gehört. Die Anfrage wird proxymäßig (Flag `[P]`) an den `/ws`-Pfad dieses Hosts weitergeleitet. Da im Anwendungsfall dieser Arbeit von der eingehenden Anfrage erwartet wird, dass sie eine WebSocket Verbindung aufbaut, wird auch bei der Weiterleitung `ws://` als Protokoll festgelegt.

- Schließlich wird in Zeilen 24 – 27 dem Apache-Server der Zugriff auf die in Zeile 9 bereitgestellten Dateien erlaubt.

Ab diesem Punkt ist es bereits möglich, Dateien aus einem festgelegten Ordner über den Browser zu betrachten. Um *mehreren* Nutzern die Möglichkeit zu geben, selbstständig Ordner bereitzustellen müssen die Anpassungen aus Abschnitt 3.4 vorgenommen werden. Im nächsten Abschnitt soll aber davor eine mögliche Schwachstelle in der Beispielkonfiguration des wslink-Launchers beschrieben werden.

3.3. Mögliche Schwachstelle in der Beispielkonfiguration

3.3.1. Erläuterung der Ursache

Um ein genaueres Verständnis für das exakte Verhalten des wslink-Launchers zu entwickeln, wurde der Quellcode des Launchers im Detail betrachtet. Dabei wurde in der `createSession` Methode der `SessionManager` Klasse ein Verhalten festgestellt, das in keiner Dokumentation des Launchers beschrieben wurde. Der Quellcode des `SessionManagers` ist in Listing 3.4 dargestellt. Im Folgenden sollen Schritt für Schritt die Effekte der einzelnen Anweisungen untersucht werden: Die Methode `createSession` in Zeile 291 wird als Teil von Schritt 6 in Abschnitt 3.2 aufgerufen. Der JSON-Body der POST-Anfrage wird dabei in ein Python Dictionary konvertiert und im `options` Parameter übergeben. `options` enthält also zunächst die Schlüssel-Wert-Paare aus der Anfrage.
Als Erstes wird in Zeile 293 eine ID für die neue Sitzung generiert. Daraufhin wird in Zeile 295 versucht, eine freie Host-Port-Kombination aus dem `resources`-Block der Konfiguration zu ermitteln. Falls keine freie Ressource verfügbar ist, wird abgebrochen und `None` als Fehlerindikator zurückgegeben. Ansonsten werden in Zeilen 300–302 die Einträge `"id"`, `"host"` und `"port"` des `options` Dictionary mit der eben generierten ID, sowie den ermittelten Host und Port Werten überschrieben. Zeilen 303–304 generieren einen zufälligen String und fügen `options` einen `"secret"` Eintrag hinzu, falls noch keiner vorhanden war, der Client also keinen gesendet hat.
Der kritische Teil ist nun in erster Linie in den Zeilen 305–314 zu finden: `replaceVariables` nimmt drei Argumente, `template_str`, `variable_list` und `sanitize`. Die Funktion ist dafür zuständig, die Platzhalter aus der Konfigurationsdatei mit konkreten Werten zu ersetzen.

- `template_str` soll eine Zeichenkette sein, eventuell mit Platzhaltern.
- `variable_list` ist eine Liste von Dictionaries, wobei jedes Dictionary Platzhalternamen und die zugehörigen Werte enthält.
- `sanitize` ist in der Beispielkonfiguration aus Abschnitt 3.2.1 immer ein leeres Dictionary und wird somit von `replaceVariables` ignoriert.

Beim Aufruf von `replaceVariables` in Zeilen 305–309 werden also folgende Argumente übergeben: Als `template_str`-Argument wird der `sessionURL`-Wert aus der Konfigurationsdatei übergeben, im Beispiel also `ws://example.com/ws?sessionId=${id}`. Als `variable_list` wird eine Liste mit zwei Elementen übergeben.

1. Das `options` Dictionary das vom Client gesendet wurde, und zu diesem Zeitpunkt bereits mit ID, Host und Port gefüllt ist.
2. Der `properties` Block der Konfigurationsdatei in Form eines Python Dictionary

Die Funktion ersetzt dann den „`${id}`“-Teil der Sitzungs-URL mit dem Wert, der in `options["id"]` gespeicherten ist, also der generierten Sitzungs-ID. Danach sind bereits alle Platzhalter aus der Sitzungs-URL ersetzt, sodass keine weiteren Änderungen vorgenommen werden. Dieses Ergebnis wird dann aus der Funktion zurückgegeben. In Zeile 305 wird dem `options` Dictionary ein `"sessionURL"` Eintrag mit dem Ergebnis hinzugefügt, bzw. ein bereits vorhandener Eintrag überschrieben.
`replaceList` in Zeile 310 ist nahezu identisch zu `replaceVariables`, mit dem Unterschied, dass hier `template_str` eine Liste an Zeichenketten ist. Dabei wird `replaceVariables` auf jede dieser Zeichenketten angewendet und die resultierende Liste zurückgegeben. In Zeilen 310–314 werden also als `template_str` die Befehlsargumentliste übergeben und als `variable_list` wie bereits in Zeile 307 eine Liste aus `options` und dem `properties` Block, wobei `options` zu diesem Zeitpunkt zusätzlich einen `sessionURL` Eintrag enthält.
Bei Verwendung der Konfiguration aus Abschnitt 3.2.1 ist die `template_str` Liste also

```
["${python_exec}", "--dr", "${visualizer_exec}, "--port", "${port}",...].
```

Die in `variable_list` übergebene Liste enthält beispielsweise

```
variable_list[0] = { // vom Client gesendet und vom Launcher aufgefüllt
    "application": "visualizer",
    "id": "abc123",
    "host": "localhost",
    "port", 9000,
    "sessionURL": "ws://example.com/ws?sessionID=abc123",
    ...
}
variable_list[1] = { // aus properties block
    "python_exec": "/opt/paraview/bin/pvpython",
    "visualizer_exec": "/opt/.../pv-visualizer.py",
    ...
}
```

Die einzelnen Schritte der Durchführung laufen so ab:

1. Es wird damit angefangen, `replaceVariables` auf die `"${python_exec}"` Zeichenkette anzuwenden. Dabei wird zunächst in `variable_list[0]` ein Eintrag namens `"python_exec"` gesucht. Weil kein solcher Eintrag existiert, wird anschließend in `variable_list[1]` gesucht. Da dort dann ein passender Eintrag vorzufinden ist, wird der `${python_exec}` Platzhalter mit dem korrespondierenden Wert ersetzt.
2. Bei `"--dr"` ist kein Platzhalter vorzufinden, dieses Element wird also nicht verändert.
3. Die `"${visualizer_exec}"` Zeichenkette enthält erneut einen Platzhalter, der dann analog zum Fall von `"${python_exec}"` ersetzt wird.
4. `"--port"` bleibt wiederum unverändert.
5. Im Fall von `"${port}"` wird wieder zunächst in `variable_list[0]` nach einem Eintrag namens `"port"` gesucht, diesmal kann auch sofort einer gefunden werden, sodass der Platzhalter mit dem Wert `9000` ersetzt wird.
6. Analog wird auch mit allen weiteren Elementen verfahren.
7. Das Ergebnis ist schließlich `["/.../pvpython", "--dr", "/.../pv-visualizer.py", "--port", "9000", ...]`.

Dieses Ergebnis wird dann als `"cmd"`-Eintrag in das `options` Dictionary geschrieben, das anschließend aus der Funktion zurückgegeben wird. Der Code, der die Funktion aufgerufen hat verwendet dann die Einträge des Dictionaries, um unter anderem den Prozess zu starten und den Client über die Sitzungs-URL zu informieren.
Problematisch wird das Ganze, wenn ein bösartiger Client im JSON-Objekt der Anfrage neben `application` auch noch `python_exec`, oder `visualizer_exec` angibt, also beispielsweise folgendes Dokument:

```
{
    "application": "visualizer",
    "python_exec": "/path/to/some/other/executable"
}
```

Da zuerst im `options` Dictionary und erst danach in `properties` nach Ersetzungen gesucht wird und zudem `options` den Inhalt des Anfrageobjekts enthält, würde in diesem Fall der `${python_exec}` Platzhalter in Schritt 1 direkt mit `/path/to/some/other/executable` ersetzt werden. Dadurch würde anschließend vom Launcher anstatt ParaViews Pythonversion irgendein anderes, vom bösartigen Client wählbares Programm ausgeführt werden. Genauso kann per Übermittlung des `visualizer_exec` Platzhalters ein beliebiges Python-Skript ausgeführt werden. Wenn der Launcher zusätzlich als Systemdienst mit Root-Rechten ausgeführt wird, wird auch das vom bösartigen Client gewählte Programm mit Root-Rechten ausgeführt. Inwieweit diese Sicherheitslücke letztendlich konkret ausnutzbar ist, um das System zu kompromittieren oder anderweitig Schäden anzurichten, konnte nicht weiter ermittelt werden.

```
283 class SessionManager(object):
284     def __init__(self, config, mapping):
285         self.sessions = {}
286         self.config = config
287         self.resources = ResourceManager(config["resources"])
288         self.mapping = mapping
289         self.sanitize = config["configuration"]["sanitize"]
290 
291     def createSession(self, options):
292         # Assign id and store options
293         id = str(uuid.uuid1())
294 
295         # Assign resource to session
296         host, port = self.resources.getNextResource()
297 
298         # Do we have resources
299         if host:
300             options["id"] = id
301             options["host"] = host
302             options["port"] = port
303             if not "secret" in options:
304                 options["secret"] = generatePassword()
305             options["sessionURL"] = replaceVariables(
306                 self.config["configuration"]["sessionURL"],
307                 [options, self.config["properties"]],
308                 self.sanitize,
309             )
310             options["cmd"] = replaceList(
311                 self.config["apps"][options["application"]]["cmd"],
312                 [options, self.config["properties"]],
313                 self.sanitize,
314             )
315 
                ...

326             return options
327 
328         return None
```

Listing 3.4: Ein Ausschnitt aus der `launcher.py` Datei [34] des wslink-Launchers

```
"configuration": {...},
"resources": [...],
"properties": {}, <== leer
"apps": {
    "visualizer": {
        "cmd": [
            "/opt/paraview/bin/pvpython",
            "--dr",
            "/opt/paraview/share/pavaview-5.11/web/visualizer/server/pv-visualizer
                .py",
            "--port", "${port}",
            "--data", "/srv/simulationfiles",
            "--load-file", "examplefile.dat"
        ],
        "ready_line": "Starting factory"
    }
}
```

Listing 3.5: Die wslink-Konfiguration nach Auflösen des `properties`-Blocks

3.3.2. Lösung

Die einfachste Möglichkeit, dieses Problem zu umgehen, ist der Verzicht auf den `properties`-Block in der Konfigurationsdatei. Wie in Listing 3.5 demonstriert, müssen anstatt die Ordner- und Dateipfade getrennt in `properties` anzugeben und anschließend im Befehl zu referenzieren, die Pfade fest in den Befehl eingebaut werden. Damit wird der Ersetzungsalgorithmus nur für den `${port}`-Platzhalter durchgeführt, dessen Ersetzung nicht durch den Client beeinflussbar ist, da das korrespondierende Element im `options` Dictionary vom Launcher überschrieben wird.

Eine Alternative findet sich noch im `sanitize` Parameter der `replaceVariables`-Funktion. Der Wert, der für diesen Parameter übergeben wird, wird aus dem `sanitize`-Block in der wslink-Launcher-Konfiguration abgeleitet. Ein solcher Block ist beispielhaft in Listing 3.6 gezeigt, und funktioniert wie folgt: In dem Block können für verschiedene Propertynamen (vgl. `eine_property`, `andere_property`) erlaubte Werte festgelegt werden. Wenn dann die `replaceVariables`-Funktion aufgerufen wird, wird darin vor der Variablenersetzung noch überprüft, ob der einzusetzende Wert den Kriterien des `sanitize`-Blocks entspricht. Die Kriterien können dabei auf zwei Varianten festgelegt werden. Entweder über eine Liste an erlaubten Werten (bei `eine_property`) oder einen regulären Ausdruck, den der Propertywert erfüllen muss (bei `andere_property` z.B. darf der Wert keinen Schrägstrich enthalten). Falls die Kriterien nicht erfüllt werden, wird statt dem ursprünglichen Wert der Wert eingesetzt, der im `default`-Feld der Konfiguration festgelegt wurde. Das `sanitize` Feature ist folglich in erster Linie dafür gedacht, einzelne Parameter des Befehls in beschränktem Umfang vom Client bestimmen zu lassen. Wenn damit aber jegliche Modifikation durch den Client

```
"configuration": {
    ...
    "sanitize": {
        "eine_property": {
            "type": "inList",
            "list": [
                "wert A", "wert B", "default-wert"
            ],
            "default": "default-wert"
        },
        "andere_property": {
            "type": "regexp",
            "regexp": "^[^/]*$",
            "default": "wert bei mismatch"
        }
    }
},
"properties": {
    "eine_property": "wert A",
    "andere_property": "anderer Wert"
},
...
```

Listing 3.6: Struktur des `sanitize`-Blocks

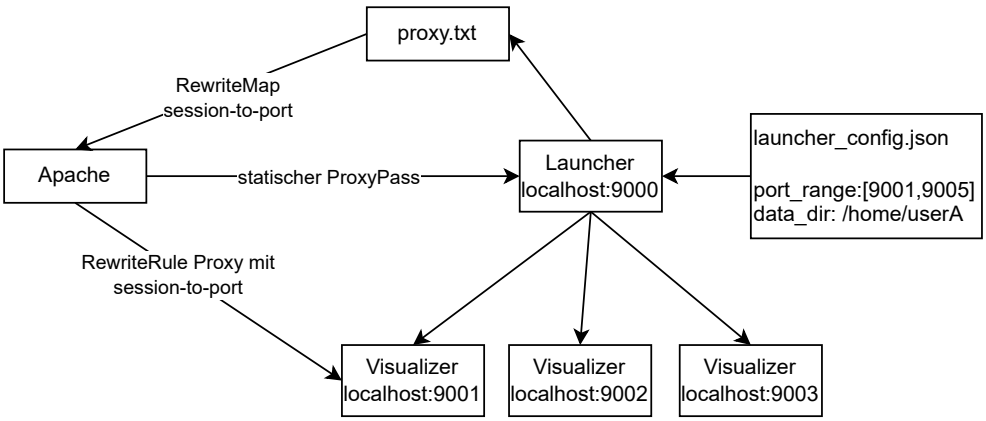


Abbildung 3.2.: Struktur des ursprünglichen Multi-User-Setup als Ergebnis von Abschnitt 3.2

vermieden werden soll, dann muss der Propertywert gleich dreifach angegeben werden: einmal im `properties`-Block, einmal beim `list`-Feld des `sanitize`-Blocks und noch einmal als `default`-Wert im `sanitize`-Block. Damit würde aber die Übersichtlichkeit verloren gehen, die eigentlich das ursprüngliche Ziel bei der Verwendung der Platzhalter war. Somit wurde sich in diesem Anwendungsfall gegen die Verwendung des `sanitize`-Konfigurationsblocks entschieden.

3.4. Anpassungen für mehrere separate Data-Sharing-Projekte

Basierend auf dem Ergebnis aus Abschnitt 3.2 soll nun das eigentliche Ziel dieser Arbeit umgesetzt werden. Nutzer des LRZ sollen von der Compute Cloud aus beliebige Ordner aus ihrem persönlichen DSS Verzeichnis über den ParaView Visualizer online verfügbar machen können und die Freigabe rückgängig machen können. Außerdem sollen sie auflisten können, welche Ordner sie momentan teilen. Zugriff soll dabei über individuelle Zugriffslinks geschehen, ähnlich wie bei der Dateifreigabe über Google Drive, Microsoft OneDrive oder Dropbox. Gleichzeitig soll, wenn möglich, die vorhandene Software, also wslink-Launcher und ParaView Visualizer nicht modifiziert werden. Zwar würden Anpassungen am Code des Launchers auch den gewünschten Effekt erzielen, jedoch müssten jedes Mal, wenn eine neue Version des Launchers zum Einsatz kommt, zum Beispiel durch ein Update, das Sicherheitslücken schließt, diese Anpassungen erneut an der neuen Version vorgenommen werden. Dieser erhöhte Wartungsaufwand ist in diesem Fall nicht erwünscht.
Eine vollständige Schritt-für-Schritt-Anleitung zur Bereitstellung auf der LRZ-Infrastruktur ist im Anhang B verfügbar.
Der Grundgedanke hinter der Umsetzung ist schließlich der folgende: Die Komponenten rechts von Apache in Abbildung 3.2, also der Launcher, die gestarteten Visualizer-Instanzen, die `proxy.txt` Datei für die Sitzung-zu-Port Abbildung und die Konfigurationsdatei des

Launchers werden zu einem Projekt zusammengefasst. Wann immer ein Nutzer einen neuen Ordner teilen möchte, wird diese Struktur automatisch repliziert und mit vom Nutzer festgelegten Parametern (Ordnerpfad, Standarddatei, etc.) konfiguriert. Der Apache-Server bleibt eine einzelne Instanz und leitet alle Anfragen an die gewünschten Projekte weiter. Die neue Struktur ist dann in Abbildung 3.3 dargestellt.
Die Schritte aus Abschnitt 3.2 werden somit wie folgt angepasst:

1. Auf dem Server laufen neben dem Apache-Server noch *mehrere* wslink-Launcher auf verschiedenen Ports, jeder mit eigener Konfigurationsdatei, die jeweils ein anderes Projekt bereitstellen. In einer `launchers.txt` Datei werden Paare aus Projekt-IDs und den localhost Adressen der zugehörigen Launcher gespeichert.

2. Der Nutzer ruft die Website auf, wobei die URL die Projekt-ID beinhaltet. Idealerweise schaut die Adresse in etwa so aus: `http://example.com/?project=<Projekt-ID>`. (Dieses Format ist so nicht direkt umsetzbar, für Details siehe Abschnitt 3.4.5)

4. Die im Browser ausgeführte ParaView Visualizer Anwendung sendet eine POST-Anfrage nicht mehr an `http://example.com/paraview`, sondern an `http://example.com/project/<Projekt-ID>`.

5. Ursprünglich wurde die Anfrage per `ProxyPass` an den einzigen Launcher weitergeleitet. Nun gibt es jedoch mehrere Launcher, die jeweils für ein Projekt zuständig sind. Deshalb muss der Apache-Server bei Anfragen auf dem `/project/<Projekt-ID>` Pfad mittels `RewriteRule` die Projekt-ID aus dem Pfad extrahieren und in der `launchers.txt` Datei nach der korrespondierenden Zeile suchen. Daraus wird dann das Host-Port-Paar ermittelt, unter dem der Launcher für dieses Projekt erreichbar ist (wobei der Host in diesem Setup immer `localhost` ist). Die Umsetzung funktioniert dabei analog zu dem, wie im Ausgangspunkt in Abschnitt 3.2.2 mit den einzelnen Sitzungs-IDs verfahren wird. Die Anfrage wird anschließend lokal an `localhost:<port>/paraview` weitergeleitet und erreicht damit den verantwortlichen Launcher.

7. Anstatt einer einzelnen `proxy.txt`, die die Siztungs-ID-Host-Paare speichert, besitzt jeder Launcher eine eigene `<projekt-id>.proxy.txt`-Datei. Dort schreiben die Launcher die aktuellen Sitzungen ihres jeweiligen Projekts hinein.

8. Auch hier wird nach Start der Visualizer-Instanz mit einem JSON-Dokument geantwortet, das die URL enthält, unter dem das Visualizer-Backend nun erreichbar ist. Der Unterschied besteht hier darin, dass zusätzlich noch die Projekt-ID in die URL eingebunden wird, in etwa so: `http://example.com/ws?project=abc&sessionId=12345`.

10. Wie im Ausganspunkt müssen nun WebSocket-Verbindungen, die auf der URL aus Schritt 8 eingehen, vom Apache-Server an das korrekte Visualizer-Backend weitergeleitet werden. Für diese Multi-Projekt-Situation sind jedoch zwei Schritte an Indirektion notwendig! Zunächst muss anhand der Projekt-ID in der URL die

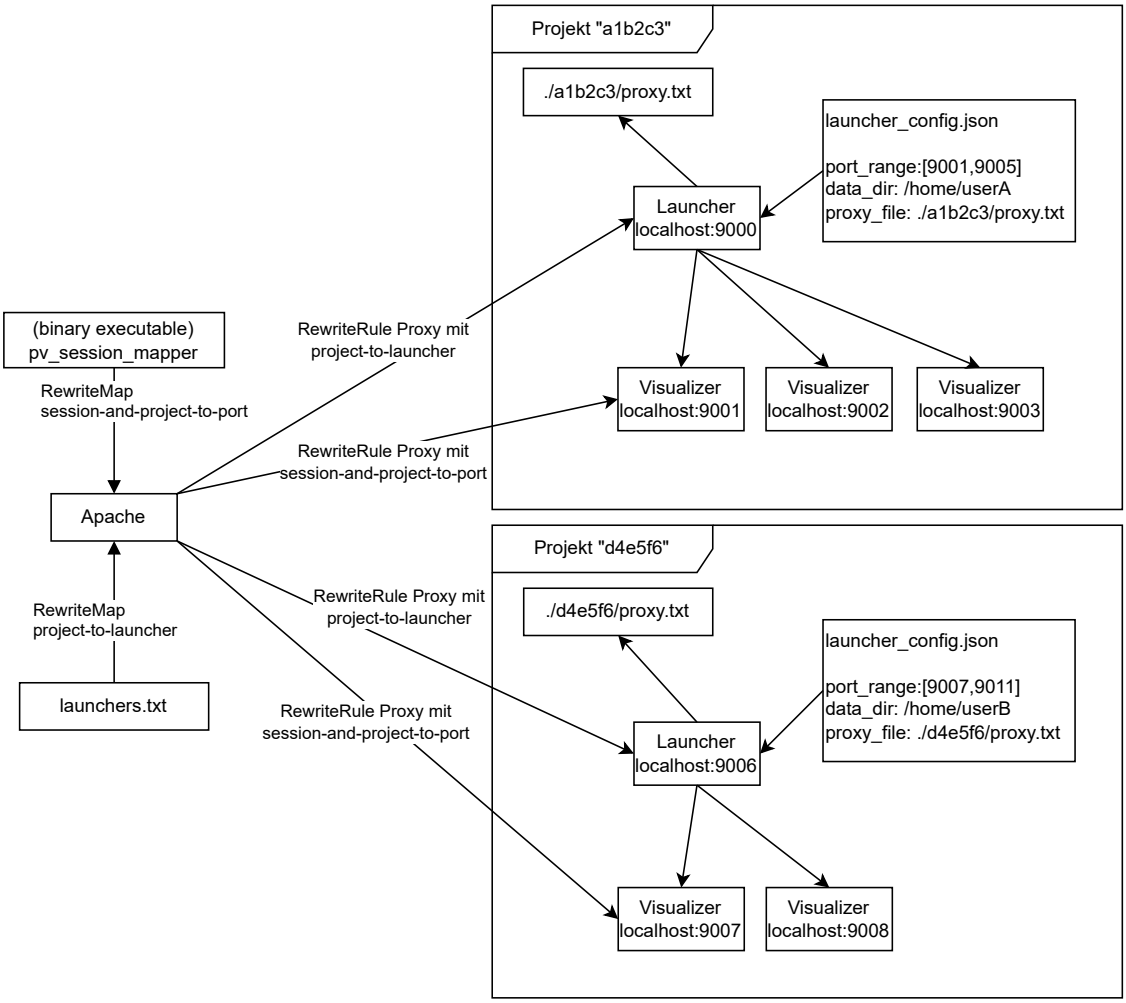


Abbildung 3.3.: Struktur mit mehreren Projekten, alle über den Apache-Server erreichbar. Für jedes vom Nutzer erstellte Projekt wird ein neuer solcher Projekt-Block aufgesetzt und angebunden.

`<id>.proxy.txt`-Datei ermittelt werden, die die Sitzungen des gewünschten Projekts enthält. Anschließend muss darin die Hostadresse des für die Sitzung zuständigen Visualizer-Backends ermittelt werden. Die Umsetzung dieser doppelte Indirektion ist zunächst ein Problem für welches jedoch in Abschnitt 3.4.1 eine Lösung präsentiert wird.

Für einige Schritte wurden Designentscheidungen getroffen, wobei sich gegen die Alternativen entschieden wurde, weil sie unerwünschtes oder inkorrektes Verhalten mit sich bringen könnten.
Eine alternative Umsetzung wäre beispielsweise, dass man in Schritt 7 allen Launchern den Pfad zu ein und derselben `proxy.txt` Datei übergibt, anstatt dass jeder Launcher seine eigene Datei verwaltet. Damit würde die doppelte Indirektion in Schritt 10 vermieden werden, ab Schritt 6 würden sogar überhaupt keine Anpassungen mehr notwendig sein. Die Implementierung des Launchers funktioniert aber so, dass der Launcher zu jedem Zeitpunkt die laufenden Sitzungen selbst im Speicher behält, und jedes Mal, wenn eine Sitzung hinzugefügt oder entfernt wird, diese Liste als ganzes in die Datei geschrieben wird. Die Launcher lesen selbst nie aus der Datei. Im Fall von mehreren Launchern bedeutet das jedoch, dass sobald einer davon eine neue Sitzung startet, die Datei komplett überschrieben wird, samt der Sitzungen die davor von anderen Launchern niedergeschrieben wurden. Somit wären immer nur die Sitzungen des zuletzt aktiven Launchers für den Apache-Server zugänglich, woraufhin dieser seine dynamische Reverse-Proxy Weiterleitung nicht korrekt durchführen kann. Deshalb wurde sich gegen diese Umsetzung entschieden.
Eine weitere Alternative wäre die Verwendung eines einzelnen Launchers für alle Visualizer-Instanzen. Dadurch würde das Problem des vorherigen Ansatzes gelöst werden. Für die Umsetzung könnte im `apps`-Block der Launcher-Konfiguration für jedes Projekt eine eigene „`application`“ mit eigenem Kommandozeilenbefehl erstellt werden, der dann den individuellen Ordnerpfad des Projekts beinhaltet. Diese Einträge könnten dann, anstatt als `"visualizer"`, stattdessen mit ihrer Projekt-ID benannt werden. Schließlich müsste browserseitig nur noch dafür gesorgt werden, dass im `application`-Feld der POST-Anfrage statt `"visualizer"` die Projekt-ID aus der URL gesendet wird. Das wäre sogar ohne Modifikation des Frontend-Codes möglich (vgl. Abschnitt 3.4.5).
Ein erster Unterschied im Verhalten liegt dabei darin, dass bei Verwendung mehrerer Launcher jeder Launcher seinen eigenen Pool an Portnummern zur Verfügung hat, während bei einem einzelnen Launcher alle Projekte sich den Pool an Portnummern teilen. Diese zwei Varianten haben folglich unterschiedliche Implikationen für die Verfügbarkeit. Man nehme an es gibt zwei Projekte A, B, jeweils mit 4 Portnummern zur Verfügung. Wollen nun 8 Personen A betrachten, bekommen nur die ersten 4 eine Visualizer-Instanz zugewiesen, die restlichen landen in einer Warteschlange. Möchte jetzt eine weitere Person B betrachten, ist das möglich, da für B noch 4 Pornummern verfügbar sind. Würden sich hingegen die Projekte den Pool an Portnummern teilen, dann könnten alle ursprünglichen 8 Personen Projekt A betrachten, jedoch kann Projekt B überhaupt nicht mehr betrachtet werden. Welches der beiden Verhalten erwünscht ist, wurde nicht genauer als Ziel festgelegt.
Das Problem bei Verwendung eines einzelnen Launchers liegt jedoch darin, dass in ein

und dieselbe Konfigurationsdatei geschrieben werden muss, und der Launcher somit jedes Mal neu gestartet werden muss, sobald ein Nutzer Anpassungen an einem seiner Projekte vornimmt. Der Launcher schreibt die Sitzungsinformationen zwar in eine Datei, (die der Apache-Server dann für die RewriteRule benutzt) liest selbst jedoch nie aus der Datei, sondern verwendet nur die Daten im Hauptspeicher. Somit gehen alle laufenden Sitzungen verloren, sobald der Launcher neu gestartet wird.
Zudem werden die Visualizer-Instanzen als Unterprozesse gestartet, im Umkehrschluss also beendet, sobald der Launcher für den Neustart beendet wird. Jedes Mal, wenn ein DSS-Nutzer Änderungen an einem Projekt vornimmt, würden alle Sitzungen abgebrochen werden. Da dieses Verhalten unerwünscht ist, wurde sich auch gegen diese Alternative entschieden.

3.4.1. Neue Konfiguration des Apache-Servers

Nun soll eine Konfiguration für den Apache-Server gezeigt werden, die an den neuen Ablauf angepasst ist. Die in Listing 3.7 dargestellte Konfigurationsdatei wurde zunächst unter dem Namen `paraview-multi-project.conf` im `conf-available` Verzeichnis gespeichert. Dann wurde mit

```
$ a2disconf paraview-multi-user
$ a2enconf paraview-multi-project
```

die alte Konfiguration deaktiviert und die neue aktiviert.

- Der erste Unterschied liegt in den am Dateianfang definierten Variablen. Anstatt `SESSION_PROXY_FILE` gibt es nun eine `SESSION_MAPPER_EXEC` Variable, die den Dateipfad zum „Session-Mapper“-Programm angibt. Zusätzlich wurde noch eine `PROJECT_PROXY_FILE` und eine `PROJECTS_LOCATION` Variable hinzugefügt. `PROJECT_PROXY_FILE` gibt den Dateipfad zu einer Datei an, in der für jedes Projekt aufgelistet ist, unter welcher Adresse der zugehörige Launcher erreichbar ist. Das Format sind hierbei, wie von `RewriteMap` erwartet, leerzeichengetrennte Paare von Projekt-IDs und Host-Port-Adressen mit einem Paar pro Zeile. `PROJECTS_LOCATION` ist ein Verzeichnis, in dem für jedes neue Projekt eine `<projekt-id>.proxy.txt` Datei erstellt wird, die jeweils die Aufgabe der `proxy.txt` Datei aus dem Ausgangspunkt für das jeweilige Projekt übernimmt.

- Zeilen 5–12 sind identisch zu den Inhalten der Konfiguration am Ausgangspunkt.

- Ein essenzieller Unterschied liegt in Zeilen 15 und 16. Da es im Multi-Projekt-Setup mehrere Launcher gibt, wobei jeder für ein anderes Projekt zuständig ist, reicht eine einfache `ProxyPass` Direktive nicht mehr aus. Abhängig vom angefragten Projekt muss die Anfrage an unterschiedliche Launcher-Instanzen weiter geleitet werden.

  Dafür wird zunächst eine Abbildungsfunktion namens `project-to-launcher` definiert, die die `PROJECT_PROXY_FILE` als Quelle verwendet (mit `txt:${PROJECT_PROXY_FILE}`). Anschließend wird eine `RewriteRule` definiert, die alle Anfragen der Form `/project/`

```
1  Define PARAVIEW_HTML_ROOT /srv/visualizer-www
2  Define PROJECT_PROXY_FILE /srv/pv-configurator/launchers.txt
3  Define SESSION_MAPPER_EXEC /opt/pv-session-mapper/pv-session-mapper
4  Define PROJECTS_LOCATION /srv/pv-configurator/project-proxies
5  Define SERVER_NAME 123.111.222.233
6
7  # Hostname des Servers
8  ServerName ${SERVER_NAME}
9
10 DocumentRoot ${PARAVIEW_HTML_ROOT}
11
12 RewriteEngine ON
13
14 # Reverse Proxy für wslink Launcher, Weiterleitung abhängig von der Projekt-ID
15 RewriteMap project-to-launcher "txt:${PROJECT_PROXY_FILE}"
16 RewriteRule ^/project/([^\/]*)$ http://${project-to-launcher:$1}/paraview/ [P]
17
18 # Reverse Proxy für ParaView Visualizer, unter verwendung eines zusätzlichen
       Programms
19 RewriteMap session-and-project-to-port "prg:${SESSION_MAPPER_EXEC}
       ${PROJECTS_LOCATION}"
20
21 # Verwende Projekt-ID und Sitzungs-ID um die korrekte Visualizer-Instanz zu
       ermitteln
22 RewriteCond %{QUERY_STRING} ^project=(.*)&sessionId=(.*)$ [NC]
23 RewriteRule ^/ws.*$ "ws://${session-and-project-to-port:%1 %2}/ws" [P]
24
25 <Directory "${PARAVIEW_HTML_ROOT}">
26         AllowOverride None
27         Require all granted
28 </Directory>
```

Listing 3.7: Eine Konfiguration für den Apache-Server für die Erweiterung zum Multi-Project-Setup

`<Projekt-ID>` abfängt, und an `http://${project-to-launcher:$1}/paraview/` weiterleitet. Die `${project-to-launcher:$1}` Anweisung funktioniert äquivalent zu der `RewriteRule` aus Abschnitt 3.2.2 und resultiert in dem Wert, der in der `project-to-launcher` Abbildung zum Schlüssel `$1` gehört. `$1` wiederum referenziert dabei die erste Gruppierung des regulären Ausdrucks in der `RewriteRule`, also den Teil hinter `/project/`. Man beachte, dass hier `$1` verwendet wurde, anders als in Abschnitt 3.2.2, wo `%1` verwendet wurde. Das liegt daran, dass hier der reguläre Ausdruck der `RewriteRule` referenziert werden soll anstatt eines Ausdrucks in einer `RewriteCond`. Anschließend wird auch hier das `[P]` (für **P**roxy) Flag verwendet, damit nicht mit einem HTTP-Redirect geantwortet wird, sondern die Anfrage direkt an das Ziel weitergeleitet wird. Als Resultat kann somit eine beliebige Menge an Projekten über ein und denselben Apache-Server bereitgestellt werden. Dazu müssen nur die Projekt-IDs, sowie Host und Port der zuständigen Launcher-Instanzen zur `launchers.txt` Datei hinzugefügt werden.

- Etwas komplizierter wird es dann, wenn es darum geht, WebSocket-Anfragen abhängig von Projekt-ID und Sitzungs-ID an die korrekte Visualizer-Instanz weiterzuleiten. Wie bereits erwähnt liegt hier eine doppelte Indirektion vor: Konzeptuell müsste mittels einer vorangestellten Abbildungsregel, zum Beispiel im Stil von `RewriteRule`, festgelegt werden, welche Datei als `RewriteMap` für die eigentliche Proxy `RewriteRule` verwendet werden soll. Eine solche dynamische Auswahl von `RewriteMap`-Quellen unterstützt der Apache-Server nicht.

  Es ist hingegen erlaubt, in den `RewriteMap` Anweisungen neben Textdateien (`txt:`) noch andere Quellen für die Schlüssel-Wert-Paare, darunter externe Programme anzugeben. Dafür wird bei `RewriteMap` in Zeile 19 das `prg:` Präfix verwendet. Der Ausdruck

  ```
  "prg:${SESSION_MAPPER_EXEC} ${PROJECT}"
  ```

  in dieser Zeile wird somit zu

  ```
  "prg:/opt/pv-session-mapper/pv-session-mapper /srv/pv-configurator/project-
      ↪ proxies"
  ```

  ausgewertet, wobei der Apache-Server den Teil hinter `prg:` als Befehl interpretiert. Hier wird also das `pv-session-mapper` Programm (siehe Abschnitt 3.4.2) mit `/srv/pv-configurator/project-proxies` als Argument ausgeführt. Ein hier angegebenes Programm wird einmalig beim Start des Apache-Servers gestartet und muss anschließend über Standardeingabe und Standardausgabe mit dem Apache-Server kommunizieren. Wann immer der Server eine Abbildungsfunktion auswerten muss, wird der Schlüssel, gefolgt von einem Zeilenumbruch in die Standardeingabe des Programms geschrieben. Der Server wartet anschließend darauf, dass das Programm den zugehörigen Wert, gefolgt von einem Zeilenumbruch auf die Standardausgabe schreibt.

- Die Zeile 22 ist wieder sehr ähnlich zum Ausgangspunkt. Der Unterschied besteht dar-

in, dass jetzt in den URL-Query-Parametern nicht nur einer `sessionId`, sondern auch eine Projekt-ID (Parameter `project=`) erwartet wird. Bei beiden Parametern wird der Teil der auf `sessionId=` bzw. `projekt=` folgt, jeweils mit einer Regex-Gruppierung `(.*)` erfasst.

- In Zeile 23 werden die Parameter aus dem vorherigen Regex mit `%1` und `%2` referenziert, `%1` enthält also die Projekt-ID und `%2` die Sitzungs-ID. Beide werden mit einem Leerzeichen verknüpft und an die `session-and-project-to-port` Abbildungsfunktion übergeben, um Host und Port für die Sitzung des angefragten Projekts zu ermitteln. Die Abbildungsfunktion schreibt dabei den kombinierten Text in die Standardeingabe des pv-session-mapper Programms und erhält Host und Port auf der Standardausgabe. Host und Port werden somit bei der `ws://${session-and-project-to-port:%1 %2}/ws` URL in der Mitte eingefügt und die eingehende Anfrage an diese Adresse übergeben.

3.4.2. Das `pv-session-mapper` Programm

Wie im vorherigen Abschnitt beschrieben, wird, um die doppelte Indirektion aufzulösen, ein externes Programm als Quelle für die Abbildungsfunktion in der Konfiguration verwendet. Dieses Programm wurde im Rahmen dieser Arbeit eigens für diesen Zweck entwickelt und in der Programmiersprache Rust geschrieben. Der Quellcode für das Programm ist im Anhang C angefügt. Verwendet wird es wie folgt:
Beim Programmstart wird ein Ordnerpfad als Programmargument erwartet. In diesem Ordner muss für jedes Projekt, das bereitgestellt werden soll, eine `<projekt-id>.proxy.txt` Datei sein, die die Sitzungsinformationen für das jeweilige Projekt enthält.
Standardeingabe und Standardausgabe sind die Kommunikationsschnittstellen für die Interprozesskommunikation mit dem Apache-Server. Auf der Standardeingabe erwartet das Programm einzelne Zeilen, die jeweils Projekt-ID und Sitzungs-ID, verknüpft durch ein Leerzeichen, enthalten. Sobald eine Zeile eingegangen ist, trennt das Programm diese zunächst anhand des Leerzeichens, sodass Projekt-ID und Sitzungs-ID separat vorliegen. Anschließend wird in dem Ordner, der als Programmargument übergeben wurde, nach einer Datei namens `<projekt-id>.proxy.txt` gesucht, wobei `<projekt-id>` für die gesuchte Projekt-ID steht. Dann wird diese Datei geöffnet und Zeile für Zeile gelesen. Die Zeilen sollten jeweils aus einer Sitzungs-ID, gefolgt von einem Leerzeichen, gefolgt von Host und Port der zuständigen Visualizer-Instanz bestehen. Bei jeder Zeile überprüft das Programm, ob die Sitzungs-ID vor dem Leerzeichen mit der Sitzungs-ID, die auf der Standardeingabe übergeben wurde, übereinstimmt. Ist dies nicht der Fall, wird die gleiche Überprüfung bei der nächsten Zeile der Datei durchgeführt. War der Test jedoch erfolgreich, wird der Teil hinter dem Leerzeichen, also Host und Port, gefolgt von einem Zeilenumbruch auf der Standardausgabe ausgegeben.
Die asymptotische Laufzeit dieses Algorithmus verhält sich linear zur aktuellen Anzahl an Sitzungen des angefragten Projekts, da in der `.proxy.txt` alle Zeilen nacheinander überprüft werden müssen und somit die doppelte Zeilenzahl zu doppelt so vielen Überprüfungen führt. Es liegt auch eine lineare Abhängigkeit zur Zeichenzahl der Sitzungs-IDs vor,

da für einen Vergleich der Gleichheit zweier Zeichenketten genauso viele Operationen nötig sind wie Zeichen in der (kürzeren) Zeichenkette enthalten sind. Abhängig vom genutzten Dateisystem können auch noch die Laufzeitkosten der Suche des Projektordners linear abhängig von der Projektzahl sein. In der Praxis sind, zumindest bei Verwendung einer einzelnen Cloud Instanz all diese Aspekte vernachlässigbar, da auf einer einzelnen Instanz sowieso nur verhältnismäßig wenige ParaView-Visualizer-Instanzen gleichzeitig in akzeptabler Geschwindigkeit ausgeführt werden können und somit keine großen Mengen an Sitzungs-IDs zustande kommen können.

In der Dokumentation des Apache-Servers werden mehrere Aspekte hervorgehoben, die bei Verwendung eines Programms als Quelle besonders beachtet werden sollten (vgl. [25]). Erstens wird empfohlen, das Programm so simpel wie möglich zu halten, um die Wahrscheinlichkeit, dass sich das Programm aufhängt zu reduzieren. Außerdem sollte die Standardausgabe nicht gepuffert sein, da es sonst passieren kann, dass eine Ausgabe im Puffer landet, jedoch den Apache-Server nicht erreicht, bis der Puffer zu einem späteren Zeitpunkt geleert wird, zum Beispiel weil durch weitere Ausgaben der Puffer komplett gefüllt wurde. Als Konsequenz würde der Server mit der weiteren Verarbeitung der Anfrage bis zu diesem Zeitpunkt warten, und somit erst nach einer potenziell sehr langen Zeit antworten. Schließlich sollte noch beachtet werden, dass das Programm einen potenziellen Flaschenhals darstellt, da alle auf übereinstimmenden URLs eingehenden Anfragen sequenziell von diesem einzelnen Programm abgearbeitet werden müssen. Dementsprechend sollte das Programm gerade bei vielen gleichzeitige Anfragen angemessen schnell sein.

In Rust ist die Standardausgabe zwar gepuffert, sodass die zweite Empfehlung so nicht direkt umgesetzt werden kann. Es wird intern aber ein `LineWriter` verwendet, der den Puffer leert, sobald ein Zeilenumbruch ausgegeben werden soll. Zudem wird, um sicherzugehen, in dem Programm nach jeder Ausgabe die `flush()`-Funktion aufgerufen, die erneut versucht den Puffer zu leeren. Insgesamt wird so sichergestellt, dass die Ausgaben den Apache-Server so bald wie möglich erreichen. Dadurch können davon abhängige Anfragen schnellstmöglich bearbeitet werden.

Unter anderem wegen der letzten Empfehlung der Dokumentation wurde sich dafür entschieden, das Programm in der Programmiersprache Rust zu schreiben. Rust ist eine moderne Systemprogrammiersprache, die eine Performance ähnlich zu C und C++ liefert, gleichzeitig aber noch Sicherheit in Form von Speichersicherheit und sicherer Nebenläufigkeit bietet. Wie C oder C++ ermöglicht auch Rust den Entwicklern präzise Kontrolle über die Speichernutzung, ist besonders Maschinennah, und erlaubt somit eine präzisere Vorhersage der Laufzeitkosten des Programmcodes im Vergleich zu anderen Programmiersprachen [35]. Die Speichersicherheit ist dabei das, was die Sprache in diesem Anwendungsfall gegenüber C/C++ hervorhebt. So wird in einem Bericht des Chromium Projekts, das unter anderem die Basis für Googles Chrome Browser darstellt, erläutert, dass ungefähr 70 % der schwerwiegenden Sicherheitsbugs im Projekt auf fehlerhafte Speicherzugriffe zurückzuführen sind [36]. Deshalb wird experimentiert, in welchem Umfang Rust die Verwendung von C/C++ ergänzen oder ersetzen kann [37].

Gerade da der pv-session-mapper Eingaben verarbeitet, über die ein möglicherweise

böswilliger Endnutzer Kontrolle hat, sollte Speichersicherheit besonders hervorgehoben werden. Damit kann zumindest eine Kategorie von Sicherheitslücken präventiv verhindert werden.
Abschließend wurden noch Benchmarks des pv-session-mappers durchgeführt (siehe Abschnitt A.2). Dabei wurde gezeigt, dass die Zahl der Anfragen, die in 100 ms bearbeitet werden die Maximalanzahl der Visualizer-Instanzen um mehrere Größenordnungen übersteigt. Da der Apache-Server den pv-session-mapper nur für den *Beginn* einer WebSocket Verbindung verwendet, nicht jedoch für die darauffolgende bidirektionale Kommunikation, muss im Endeffekt nur eine Anfrage pro Visualizer-Instanz behandelt werden. Der pv-session-mapper wird somit keinen Flaschenhals darstellen.

3.4.3. Projektkonfigurationsskript

Um das Bereitstellen von Projekten für die Endnutzer komfortabel zu machen, wurde mit Python ein Konfigurationsskript entwickelt, das automatisch die Ordner und Konfigurationsdateien anlegt und systemd-Services registriert, sodass auch nach einem Neustart der Compute-Cloud-Instanz die Launcher für alle Projekte automatisch gestartet werden.
Da das Skript ein Python-Programm ist und auch Abhängigkeiten von externe Bibliotheken aufweist, wird zusätzlich über ein Wrapper-Skript (`create_config.sh`) das Python Environment korrekt konfiguriert. Schließlich kann noch ein weiteres Wrapper-Skript (`pvconfig`) angelegt werden, das `create_config.sh` mit `sudo` vorangestellt aufruft, sodass die Nutzer nicht selbst `sudo` voranstellen müssen.
Es werden fünf Unterbefehle unterstützt:

1. `$ pvconfig publish -d <data-dir> [-f <file>]` stellt alle Dateien im Ordner `<data-dir>` bereit, optional kann auch noch eine Datei (`<file>`) aus diesem Ordner angegeben werden, die standardmäßig geladen werden soll. Für das Projekt wird dabei eine neue ID generiert, die von den anderen Befehlen benötigt wird.

2. `$ pvconfig modify <id> [-d <data-dir>] [-f <file> | --noLoadFile]` erlaubt es, die Ordner/Dateipfade für ein existierendes Projekt mit der ID `<id>` anzupassen.

   `--noLoadFile` kann angegeben werden, wenn anfangs keine Datei geladen werden soll, so wie wenn bei `publish` das `-f` Argument weggelassen worden wäre.

3. `$ pvconfig list` zeigt IDs und Ordnerpfad aller Projekte an, die von dem aufrufenden Nutzer veröffentlicht wurden.

4. Mit `$ pvconfig show <id>` können Details (Ordnerpfad, Datei, URL für den Zugang) eines Projekts des Nutzers angezeigt werden.

5. Mit `$ pvconfig unpublish <id>` kann Webzugriff auf das Projekt `<id>` wieder zurückgenommen werden. Dabei werden auch alle Konfigurationsdaten des Projekts gelöscht.

Das Skript baut dann, um die Projekte zu verwalten, folgende Ordnerstruktur auf:

```
/
|-- srv/pv-configurator/
|   |-- configurator_settings.json
|   |-- launchers.txt
|   |-- ports.json
|   |-- projects.json
|   |-- projects/
|   |   |-- <projektid1>/
|   |   |   |-- launcher_config.json
|   |   |   |-- config.json
|   |   |   `-- pv-<projektid1>-launcher.service
|   |   |-- <projektid2>/
|   |   |   `-- ...
|   |   `-- ...
|   `-- project-proxies/
|       |-- <projectid1>.proxy.txt
|       |-- <projectid2>.proxy.txt
|       `-- ...
`-- var/log/paraview-launcher/
    |-- <projekt-id>/
    |   |-- launcher.log
    |   |-- <sitzungsid1>.log
    |   |-- <sitzungsid2>.log
    |   `-- ...
    `-- ...
```

- `configurator_settings.json` ist eine JSON-Datei, die verschiedene Werte (z.B. Dateipfade) angibt, die vom Konfigurationsskript benötigt werden. Die Werte beschreiben dabei die Systemumgebung. In der Datei wird ein JSON-Objekt erwartet, das folgende Attribute besitzt:
  - `"servername"` um die IP-Adresse oder den Hostnamen festzulegen, unter dem der Server vom Browser aus erreichbar ist. Wenn nicht der HTTP Standardport 80 verwendet wird, muss noch der Port angefügt werden (`<host>:<port>`).
  - `"python_exec"` muss der Pfad zum `pvpython` Programm festgelegt werden, das in der ParaView Installation enthalten ist.
  - `"visualizer_exec"` muss der Pfad zum Backend des ParaView Visualizers festgelegt werden. Der Visualizer ist auch Teil der ParaView Installation.
  - `"launcher_exec"` muss angeben, wo sich das Wrapper Skript für den wslink-Launcher befindet.

  Diese Werte werden dann bei den generierten Konfigurationsdateien eingesetzt.

- `launchers.txt` ist die Datei, die in Zeile 15 der Apache-Konfiguration (Listing 3.7) für die Zuordnung von Projekt-ID zu Launcher-Instanz verwendet wird. Bei jedem Aufruf von `publish` fügt das Konfigurationsskript eine entsprechende Zeile zu dieser

Datei hinzu, bzw. entfernt die korrespondierende Zeile beim Aufruf von `unpublish`. Da der Apache-Server bei jeder eingehenden Anfrage die Datei neu liest, werden Änderungen sofort beachtet.

- In `ports.json` speichert das Konfigurationsskript eine Liste von belegten Portnummerintervallen. Jedes Mal, wenn ein neues Projekt gepublisht wird, werden dieser Liste die neu reservierten Intervalle hinzugefügt, bzw. entfernt im Fall von `unpublish`. Das Skript geht davon aus, dass alle Portnummern über 9000 nur von diesem Skript (bzw. den Launcher- und Visualizer-Instanzen) genutzt werden. Ist das nicht der Fall, kann es passieren, dass einzelne Launcher- oder Visualizer-Instanzen dadurch blockiert werden.

  Wenn das Skript ein Intervall reservieren möchte, sucht es ab Port 9000 in dieser Datei nach Lücken, und füllt diese auf. Für jedes Projekt wird eine Portnummer für den Launcher belegt, und in der aktuellen Version genau 5 Portnummern für die Visualizer-Instanzen reserviert, es wäre jedoch denkbar, den Nutzer (oder Systemadministrator) bestimmen zu lassen, wie viele Ports für ein bestimmtes Projekt reserviert werden sollen. Der Portreservierungsalgorithmus ist in dem Fall dazu in der Lage, auch kleinere Lücken zunächst zu füllen, wenn ein neues Projekt veröffentlicht wird. Sind also Intervalle [9000, 9004] und [9007,9010] reserviert und es sollen für ein neues Projekt 6 weitere Ports reserviert werden, dann werden diesem Projekt die Intervalle [9005,9006] und [9011,9014] zugewiesen. Der Launcher würde dabei Port 9005 erhalten, die Visualizer-Instanzen den Port 9006, sowie die Ports 9011–9014. Damit wird sichergestellt, dass alle für Launcher und Visualizer verwendeten Ports möglichst nah am Port 9000 liegen. Wenn gleichzeitig beachtet wird, möglichst keine Dienste auf Portnummern in der Nähe von Port 9000 laufen zu lassen, kann so ein Kollisionsrisiko minimiert werden.

- In `projects.json` speichert das Skript, welche Projekt-IDs welchem Nutzer gehören. Diese Informationen sind relevant um zu überprüfen, ob ein Nutzer das jeweilige Projekt anpassen darf und um einem Nutzer die Liste seiner Projekte anzuzeigen.

- Der `projects`-Ordner enthält für jedes Projekt einen Unterordner, benannt mit der Projekt-ID.

- In jedem dieser Unterordner liegt eine `config.json` Datei, die vom Konfigurationsskript verwendet wird, um projektspezifische Informationen zu speichern. Aktuell werden vom Skript vier Werte darin gespeichert:

  1. Der Port auf dem der verantwortliche Launcher gestartet werden soll,
  2. die Portintervalle, die für die Visualizer-Instanzen reserviert sind. Im Fall einer Fragmentierung (falls für verschiedene Projekte verschiedene Anzahlen an Ports reserviert werden), können hier mehrere Intervalle gelistet sein, z.B. `[[9006,9006], [9011,9014]]` aus obigem Beispiel.
  3. Der Ordner, der bei `publish` oder `modify` angegeben wurde,

4. die Datei, die bei `publish` oder `modify` angegeben wurde. Dieser Wert kann weggelassen werden.

- `launcher_config.json` ist die Konfigurationsdatei für die jeweilige Launcher-Instanz. Diese Dateien werden vom Konfigurationsskript generiert. Im Fall einer Änderung wird nicht versucht, in der Launcher-Konfiguration die einzelnen Werte (Ordner, Port, etc.) anzupassen, stattdessen werden die Werte aus der `config.json` Datei verwendet, und die `launcher_config.json` von Grund auf neu generiert. Dadurch liegt zwar eine gewisse Redundanz vor, da Ordnerpfade, Dateipfade und Portintervalle sowohl in der Launcher-Konfiguration, als auch in der `config.json` Datei gespeichert sind, gleichzeitig wird so aber das Konfigurationsskript weniger komplex.

- Schließlich gibt es noch die `pv-<project-id>-launcher.service` Dateien, die die jeweiligen systemd-Units beschreiben.

- Der `project-proxies` Ordner enthält für jedes Projekt eine `<projekt-id>.proxy.txt` Datei, die die Funktion der ursprünglich `proxy.txt` genannten Datei übernimmt. Der zugehörige Launcher trägt darin die aktuellen Sitzungen ein. Das ist auch die Ordnerstruktur, die vom pv-sesssion-mapper-Programm erwartet wird, um die Visualizer-Instanzen zu ermitteln, die zu einer Sitzung gehören. (vgl. Abschnitt 3.4.2)

Für die Generierung der `launcher_config.json` Datei verwendet das Konfigurationsskript die Schablone aus Listing 3.8, die wiederum auf dem Ergebnis aus Abschnitt 3.3 basiert. Die mit `<>` hervorgehobenen Elemente werden für jedes Projekt individuell mit konkreten Werten ersetzt: `<port>` wird mit der für den Launcher reservierten Portnummer ersetzt, `<project-id>`, `<data-dir>` und `<load-file>` analog mit Projekt-ID und den vom Nutzer angegebenen Ordner und Dateipfaden. `<from-port>` und `<to-port>` sind Start- und Endnummer eines einzelnen für den Visualizer reservierten Portintervalls, wobei diese Zeile im `resources` Block mehrfach eingefügt wird, falls mehrere Intervalle reserviert wurden. `<username>` ist schließlich der Linux-Nutzername des Nutzers, der das Skript aufgerufen hat.
Für den Befehl wurde, im Vergleich zu Abschnitt 3.3 noch `sudo -u <username>` vorangestellt. Das liegt daran, dass die Visualizer-Instanzen auf die DSS-Dateien des Nutzers zugreifen sollen, weshalb der Visualizer mit den Rechten dieses Nutzers ausgeführt werden muss. Für genaueres zum Rechtemanagement, siehe im nächsten Abschnitt (3.4.4)

3.4.4. UNIX-Rechtemanagement

Bei der Rechtevergabe an die Prozesse, sowie bei der Zugriffsbeschränkung auf die Konfigurationsdateien sollen zwei Aspekte besonders berücksichtigt werden:

1. Compute-Cloud-Nutzer sollen exklusiv über das Konfigurationsskript in begrenztem Umfang Anpassungen am wslink-Launcher vornehmen können, direkte Änderungen der Konfigurationsdateien sollen nicht möglich sein.

```
{
    "configuration": {
        "host": "localhost",
        "port": <port>,
        "proxy_file": "/srv/pv-configurator/project-proxies/<project-id>.proxy.txt,
        "endpoint": "visualizer",
        "sessionURL": "ws://<servername>/ws?project=<project-id>&sessionId=${id}",
        "fields": [],
        "timeout": 60,
        "log_dir": "/var/log/paraview-launcher/<project-id>",
    },

    "resources": [
        {"host": "localhost", "port_range": [<from-port>, <to-port>]},
        <...>
    ],

    "apps": {
        "visualizer": {
            "cmd": [
                "sudo", "-u", <username>, "/opt/paraview/bin/pvpython",
                "--dr", "/opt/paraview/share/paraview-5.11/web/visualizer/server/pvw-
                    visualizer.py",
                "--port", "${port}", "--data", <data-dir>, "--loadFile", "<load-file>
                    "],
            "ready_line": "Starting factory"
        }
    }
}
```

Listing 3.8: Die Vorlage für die `launcher_config.json` Konfigurationsdateien. Die mit <> hervorgehobenen Elemente werden für jedes Projekt individuell mit konkreten Werten ersetzt.

2. Allgemein soll, soweit möglich das grundlegende Prinzip der geringsten Privilegien (Principle of Least Privilege) angewendet werden. Dieses besagt, dass jedes Programm und jeder Nutzer eines Systems nur die kleinstmögliche Menge an Privilegien erhält, die für die Durchführung einer Aufgabe notwendig sind. Dadurch wird in erster Linie der potenzielle Schaden begrenzt, der im Fehlerfall verursacht werden kann [38].

Rechtevergabe mit `sudo`

Ein Vorteil der Verwendung von Ubuntu als Betriebssystem ist, dass es bereits für die Verwendung des `sudo`-Programms eingerichtet und vorkonfiguriert ist. `sudo` erlaubt, dass ein ausgewählter Nutzer nur bestimmte Befehle mit Root-Rechten oder den Rechten eines bestimmten (System-) Benutzers ausführen darf. Dieses Feature ist notwendig, um die erste Anforderung umsetzen zu können, da dafür Dateien beschrieben werden müssen, auf die der Nutzer außerhalb des Konfigurationsskripts keine Zugriffsrechte haben soll.
Die genauen Berechtigungen werden in der Datei `/etc/sudoers` festgelegt. Standardmäßig ist die Datei so konfiguriert, dass zusätzlich noch alle Dateien aus dem `/etc/sudoers.d/` Verzeichnis zur Konfiguration herangezogen werden. Das hat den Vorteil, dass einzelne Systemkomponenten `sudo`-Berechtigungen in separaten, kleineren und somit leichter zu verwaltenden Dateien festlegen. Für dieses Projekt werden diese Anpassungen in der Datei `/etc/sudoers.d/pv-sudoers` definiert. **Änderungen an den `sudoers` Dateien sollte nur über den Befehl `visudo` vorgenommen werden.** `visudo` warnt, wenn die Datei mit fehlerhafter Syntax gespeichert wird. Eine fehlerhafte Syntax hätte als Konsequenz, dass `sudo` die `sudoers` Datei nicht liest und somit jeden `sudo`-Zugriff auf dem System ablehnt. Gerade auf Ubuntu, wo standardmäßig der Wechsel zum Root-User ohne `sudo` deaktiviert ist, gäbe es somit keine Möglichkeit mehr, Aktionen als Root-User auszuführen und den Fehler zu beheben.
Ein weiteres Feature von `sudo` ist, dass bei dessen Verwendung die `SUDO_USER` Umgebungsvariable definiert wird, die den Nutzernamen enthält, der `sudo` ausführt. Dieser Wert wird von `pvconfig`-Skript verwendet, um zu bestimmen, wessen Projekte verwaltet werden sollen.

Evaluierung der benötigten Zugriffsrechte der verschiedenen Komponenten

Um die Anforderungen umzusetzen wurde zunächst evaluiert, welche Prozesse bzw. Programme Zugriff auf welche Dateien und Ressourcen benötigen. Die Ergebnisse wurden in Abbildung 3.4 visuell zusammengefasst. Als ersten Schritt ist zu beachten, dass einzelne Komponenten des Diagramms in Bezug auf die Rechteverwaltung von sich aus unflexibel sind:

- Der LRZ-Nutzer, der Daten veröffentlichen möchte, ist grundsätzlich kein Root-User und hat auch zunächst keine Rechte, per `sudo` einen Befehl mit Root-Rechten auszuführen.
- Gleichzeitig kann nur dieser Nutzer und kein anderer direkt auf den Nutzerordner des DSS-Containers zugreifen, nicht einmal der Root-User. Der Grund liegt, wie in

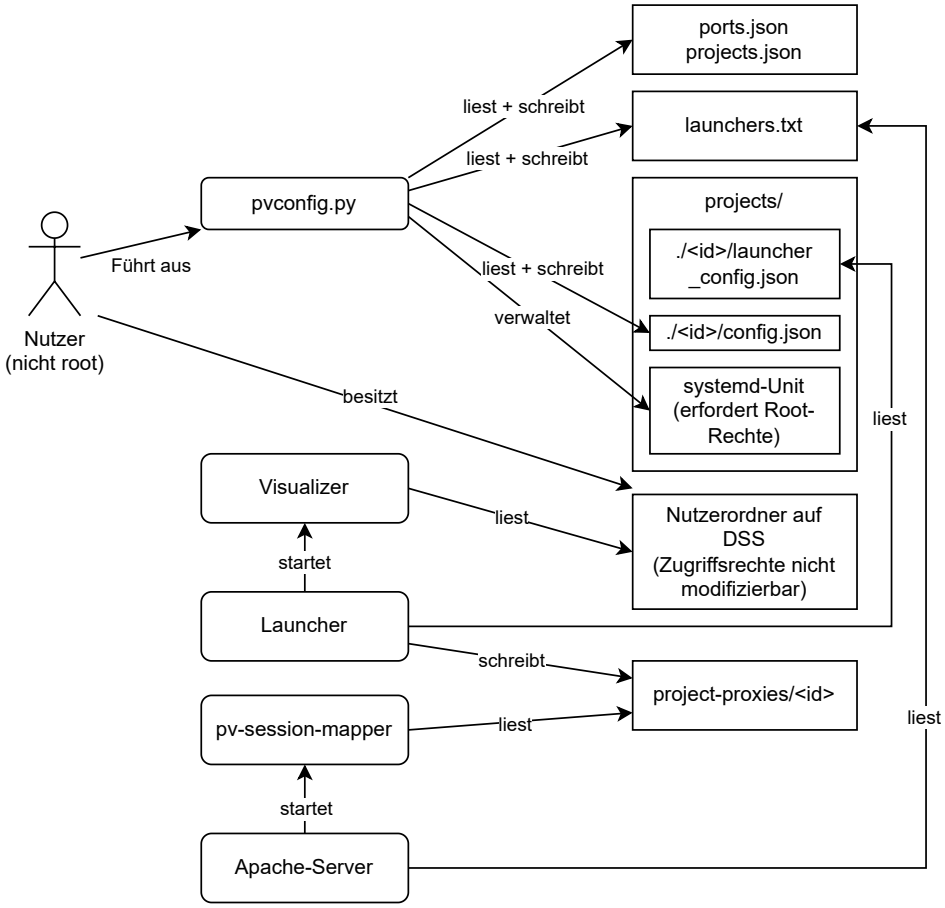


Abbildung 3.4.: Zugriffsanforderungen der Programme und Nutzer (links) auf Ressourcen (rechts)

Abschnitt 3.1 erläutert, in der Konfiguration des NFS-Servers des DSS-Containers seitens LRZ. Dabei werden Netzwerkzugriffe durch einen Root-User behandelt wie die eines gewöhnlichen Nutzers und somit werden auch alle DSS-seitigen Dateiberechtigungen angewendet. Da auf die Nutzerordner standardmäßig nur der Besitzer des Verzeichnisses zugreifen darf (auch Lesezugriff ist nicht erlaubt für andere Nutzer), darf letztendlich auch der Root-User nicht auf den Nutzerordner zugreifen.

- Außerdem muss das `pvconfig.py` Konfigurationsskript auch systemd-Units anlegen, aktivieren und starten, was Root-Rechte benötigt.

Besondere Flexibilität ist hingegen bei den Komponenten „Launcher“, „Apache-Server“ und „pv-session-mapper“ möglich. Der Launcher wird beim Systemstart (und am Ende des `publish` Befehls) von systemd gestartet, unter Verwendung der systemd-Unit, die vom `pvconfig` Konfigurationsskript erstellte wurde. In dieser lässt sich angeben, mit welchen Benutzer-/Gruppenrechten der Dienst ausgeführt werden soll.
Der Apache-Server wird mit Root-Rechten gestartet, lässt diese Rechte aber selbstständig nach dem Start fallen und wechselt zu dem Nutzer und der Gruppe, die in der Apache-Konfiguration angegeben sind.
In `/etc/apache2/apache2.conf` festgelegt via:

```
...
User <Benutzername>
Group <Gruppenname>
...
```

In beiden Fällen, Launcher und Apache-Server, ist es somit möglich, die Programme mit den Rechten beliebiger Nutzer und Gruppen ausführen zu lassen.
Bei der `RewriteMap` Konfiguration für den pv-session-mapper lässt sich auch angeben, mit den Rechten welches Nutzers und welcher Gruppe das Programm ausgeführt werden soll, sodass auch der pv-session-mapper mit beliebigen Rechten ausgeführt werden kann. Das sieht dann so aus:

```
RewriteMap session-and-project-to-port
    ↪  "prg:${SESSION_MAPPER_EXEC} ${PROJECTS_LOCATION}" <Benutzer>:<Gruppe>
```

So ist eine besondere Granularität bei der Rechtevergabe möglich, indem jeweils ein Systembenutzer für Apache-Server, einer für den Launcher und einer für den pv-session-mapper erstellt werden, jeder mit unterschiedlichen Zugriffsrechten. Damit jedoch eine systemd-Unit mit den Rechten eines beliebigen Nutzers ausgeführt werden darf, muss die Unit vom Root-User aktiviert werden, weshalb auch das Konfigurationsskript mit Root-Rechten ausgeführt werden muss. Im Endeffekt werden die Zugriffsanforderungen

aus Abbildung 3.4 wie folgt realisiert: Zunächst werden drei Systembenutzer mit identisch benannten Gruppen erstellt: „`apache-proxy`“, „`pv-launcher`“ und „`pv-session-mapper`“. Das Konfigurationsskript muss mit Root-Rechten ausgeführt werden. Da gleichzeitig der Nutzer keine allgemeinen Root-Rechte haben soll, wird zunächst eine Gruppe `pv-publisher` erstellt, die allen LRZ-Nutzern auf der Compute-Cloud zugewiesen wird. Anschließend wird in der

/etc/sudoers.d/pv-sudoers-Datei allen Mitgliedern der pv-publisher-Gruppe nur gestattet, das Wrapper-Skript für das Konfigurationsskript mit Root-Rechten auszuführen.
Ein ähnliches Prinzip wird auch für den Launcher verwendet. Da die Visualizer-Instanzen auf die zu teilenden DSS-Dateien des LRZ-Nutzers zugreifen müssen, gleichzeitig aber der gemountete DSS-Server nur Dateizugriffe von der Nutzer-ID (UID) erlaubt, die auch Eigentümer der jeweiligen Dateien auf dem DSS ist, muss der Visualizer mit den Rechten des Projekterstellers bzw. LRZ-Nutzers ausgeführt werden. Man könnte nun überlegen, auch den Launcher mit den Rechten des LRZ-Nutzers auszuführen, damit die vom Launcher gestarteten Visualizer-Instanzen diese Rechte erben. Die Konsequenz wäre jedoch, dass die vom Launcher erstellte proxy.txt Datei diesem LRZ-Nutzer gehört, und somit der Nutzer auch manuell die proxy.txt modifizieren könnte. Deshalb muss der Launcher stattdessen als separater Systembenutzer ausgeführt werden und kann dann mittels sudo die Visualizer-Instanzen mit den Rechten des LRZ-Nutzers ausführen. Dazu wird ein weiterer Eintrag in der /etc/sudoers.d/pv-sudoers-Datei vorgenommen, der dem pv-launcher Nutzer erlaubt, den Visualizer-Befehl mit den Rechten eines beliebigen Nutzers in der pv-publisher-Gruppe auszuführen. Die finale pv-sudoers-Datei sieht also wie folgt aus:

```
%pv-publisher ALL = (root:root) /opt/pv-configurator/create_config.sh
pv-launcher ALL = (%pv-publisher) /opt/.../pvpython --dr /opt/.../pvw-visualizer.py *
```

Des Weiteren lässt sich feststellen, dass auf jede Ressource im Diagramm höchstens von zwei verschiedenen Programmen zugegriffen wird. Das bedeutet wiederum, dass bei jeder Datei der Systembenutzer des einen Programms als Eigentümer festgelegt wird und die Gruppe des anderen Programms als Inhabergruppe. Dadurch kann für jede Datei separat festgelegt werden, welches der zwei Programme schreiben und welches lesen darf. Für alle anderen wird grundsätzlich jeglicher Zugriff verboten. Darauf basierend wird die Rechtevergabe für die Dateien dabei wie in Tabelle 3.1 umgesetzt:

Datei/Ordner	Eigentümer:Gruppe	Rechte
ports.json	root:root	rw- --- ---
projects.json		
projects/<id>/config.json		
projects/<id>/<systemd-unit>		
projects/ Verzeichnis	root:pv-launcher	rwx --x ---
projects/<id> Verzeichnis		
projects/<id>/launcher_config.json		rw- r-- ---
launchers.txt	root:apache-proxy	rw- r-- ---
project-proxies/ Verzeichnis	pv-launcher:pv-session-mapper	rwx r-x ---
project-proxies/<id>.proxy.txt		rw- r-- ---

Tabelle 3.1.: Eigentümer und Berechtigungen für Verschiedene Verzeichnisse und Dateipfade nach der Rechtevergabe

Alle Dateien der ersten Zeile werden nur vom Konfigurationsskript bearbeitet, also braucht

nur der Root-User Zugriffsrechte. In den Ordnern und in der Datei der zweiten Zeile wird die Konfiguration für den Launcher vom Konfigurationsskript generiert, das Skript muss in die Ordner und die Datei also schreiben können, der Launcher muss lesen können. Das gleiche Prinzip liegt in der dritten Zeile vor, hier muss statt dem Launcher der Apache-Server die Datei lesen können. Die Ressourcen der letzten Zeile werden vom Launcher erstellt und müssen vom pv-session-mapper Programm gelesen werden. Der pv-session-mapper Nutzer braucht auf die Inhalte im `project-proxies` Ordner nicht nur allgemeinen Zugriff (mit `--x`), sondern muss auch die Inhalte des Ordners auflisten können, was in der `r-x` Rechtevergabe resultiert.

Zur Erwähnung: Verwendung von systemd-User-Units

Systemd erlaubt neben den vom Systemadministrator (mit Root-Rechten) verwalteten systemd-Units auch noch die verwendung von systemd-User-Units, die von einzelnen Nutzern ohne Root-Rechte erstellt werden dürfen. Auf den ersten Blick könnte damit vermieden werden, das Konfigurationsskript als Root auszuführen, da der einzige Grund für die Verwendung von Root-Rechten im Skript die Einrichtung der systemd-Dienste ist. Eine solche Umsetzung wurde erfolglos ausprobiert, wobei das Problem darin liegt, dass die Verwaltung von systemd-User-Units nur für die Verwendung von echten Nutzern (und nicht von Systemnutzern, besonders solchen ohne Login-Shell) gestaltet ist. Die Verwaltung der systemd-Units eines Nutzers ist nur möglich, indem davor eine systemd-Login-Sitzung gestartet wurde. Der Nutzerwechsel mittels `su` oder `sudo` übernimmt hingegen die Login-Sitzung des ursprünglichen Nutzers, selbst wenn eine Login-Shell per `sudo -i` gestartet wird. Login-Sitzungen werden unabhängig von Login-Shells verwaltet und sind konzeptuell in erster Linie an physische Logins (neue ssh Verbindung, etc.) gekoppelt. Somit würde, selbst wenn das `pvconfig`-Skript mit den Rechten eines extra angelegten `pvconfig`-Systembenutzers ausgeführt wird, nur die Login-Sitzung des LRZ-Nutzers laufen. Systemd würde somit Probleme haben, auf die Login-Sitzung des `pvconfig`-Users zuzugreifen, da keine existiert.

3.4.5. Zugangslinks für den Browserzugriff

Der letzte Schritt auf dem Weg zu einem Funktionsfähigen Dienst ist, über die Weblinks Zugriff auf die verschiedenen veröffentlichten Projekte zu ermöglichen. Die Zugangsbeschränkung soll dabei, ähnlich wie bei den Filehosting Diensten Google Drive oder Dropbox, alleine über die Kenntnis eines Zugriffslinks geregelt werden, ohne zusätzliche passwortbasierte Authentisierung o.Ä.
In Abschnitt 3.2 war die Benutzung der Anwendung noch unproblematisch: Es musste nur das Wurzelverzeichnis im Webbrowser aufgerufen werden (`http://example.com/`), woraufhin die Webanwendung eine HTTP-POST-Anfrage gesendet hat, um eine Visualizer Instanz im Backend zu starten (vgl. Schritte 4 ff. in 3.2). Während zu dem Zeitpunkt noch nicht zwischen verschiedenen Launcher-Instanzen unterschieden werden musste, muss jetzt für jedes Projekt eine andere Post-Anfrage gesendet werden.
Eine der anfangs definierten Richtlinien besagt, dass Modifikationen an bereits vorhandener

Software vermieden werden soll, folglich muss auch für jedes Projekt dasselbe Web-Frontend bereitgestellt werden. Nach den Änderungen in Abschnitt 3.4 muss nun irgendwie sichergestellt werden, dass die POST-Anfragen an die gewünschten Ziele erreichen (vgl. URL in Schritt 4 in 3.4), aber ohne den Frontend-Code zu verändern.
Standardmäßig sendet das Frontend die POST-Anfragen immer an `/paraview`, unabhängig davon, auf welchem Pfad das Frontend selbst gehostet wird. Das bedeutet, auch wenn man das Frontend auf `/projects/<Projekt-ID>` bereitstellen würde, würde die Anwendung immer noch die POST-Anfrage an `/paraview` senden und nicht an `/projects/<id>/paraview`. Auch Query-Parameter in der URL werden nicht für die POST-Anfrage beibehalten. Gleichzeitig werden die Query-Parameter aber auch nicht vollständig ignoriert: Alle Schlüssel-Wert-Paare aus den Query-Parametern werden eingelesen und als Einträge in den Rumpf der POST-Anfrage eingefügt. Wenn also ein bestimmtes Projekt unter URL `/project?projectid=<Projekt-ID>` aufgerufen wird, würde die Visualizer-Anwendung eine POST-Anfrage an `/paraview` senden, wobei der Rumpf so aussehen würde:

```
{ "application": "visualizer",
"projectid": <Projekt-ID> }
```

In den `RewriteCond`/`RewriteRule` Regeln des Rewrite-Moduls des Apache Servers kann sich aber nur auf den URL-Pfad, HTTP-Header und bestimmte andere Werte bezogen werden, nicht jedoch auf den Rumpf einer POST-Anfrage. Infolgedessen kann immer noch keine Weiterleitung an den zuständigen Launcher erfolgen, obwohl erreicht wurde, dass individuell für jedes Projekt eine andere POST-Anfrage gesendet wird.
Der Frontend-Code verwendet die Query-Parameter aber nicht exklusiv für den Anfragenrumpf. Bestimmte Query-Parameter werden zusätzlich noch als Argumente für einzelne Funktionen des Frontends verwendet. Der entscheidende Query-Parameter ist der `sessionManagerURL`-Parameter. Mit diesem kann der Standardpfad für die POST-Anfrage überschrieben werden: Ist der Query-Parameter enthalten, wird die Anfrage anstatt an `/paraview` an die Adresse gesendet, die beim `sessionManagerURL`-Parameter angegeben ist. Der Apache-Server erwartet entsprechend der Konfiguration aus Abschnitt 3.4.1 eine Anfrage auf `/project/<Projekt-ID>`. Folglich muss als Query-Parameter

```
sessionManagerURL=http://example.com/project/<Projekt-ID>
```

angegeben werden. Der vollständige Link um auf ein bestimmtes Projekt zuzugreifen, lautet somit

```
http://example.com/?sessionManagerURL=http://example.com/project/<Projekt-ID>
```

4. Zusammenfassung und mögliche Erweiterungen

4.1. Zusammenfassung

In dieser Arbeit wurde zunächst ermittelt, wie ein LRZ-DSS-Container mit der LRZ-Compute Cloud verbunden werden kann. Dabei mussten die Einschränkungen beim Datenzugriff über NFS-Dateisysteme beachtet werden. Ein Compute-Cloud Nutzer kann nur auf eine Datei oder ein Verzeichniss zugreifen, wenn seine UID mit der UID des Dateieigentümers auf dem DSS übereinstimmt. Auch der Root-User hat keinerlei Zugriffsrechte auf die Dateien im Container. Um diese Einschränkung zu umgehen, mussten folglich die Benutzerkonten der DSS-Nutzer auf der Compute Cloud repliziert werden.
Daraufhin wurde die Anleitung und die Beispiele aus der ParaViewWeb Dokumentation verwendet, um einen ersten funktionsfähigen Prototypen für die Visualisierung zum Laufen zu bringen. Bei der genaueren Evaluierung der Funktionsweise wurde festgestellt, dass die Konfigurationsbeispiele so, wie sie in der ParaViewWeb Dokumentation dargestellt sind, eventuell ein Sicherheitsrisiko in sich bergen. Dieses Problem konnte schließlich erfolgreich umgangen werden, indem auf die Verwendung von Platzhaltern in der Konfigurationsdatei weitestgehend verzichtet wurde.
Anschließend wurde das eigentliche Ziel der Arbeit umgesetzt, für das jeder LRZ-Nutzer beliebig viele Ordner auf dem DSS selbstständig teilen können soll. Dafür wurde ein Teil der Komponenten aus der Beispielkonfiguration zu einer logischen Einheit zusammengefasst. Diese Einheit kann dann nach Bedarf automatisiert für jedes Projekt repliziert werden. Dafür waren wiederum Anpassungen an der Konfiguration des Apache-Servers notwendig. Außerdem musste ein zusätzliches Hilfsprogramm entwickelt werden, um eine korrekte Weiterleitung von Anfragen an die einzelnen Projekte sicherzustellen. Schließlich wurde noch ein Konfigurationsskript geschrieben, das LRZ-Nutzern die selbstständige Verwaltung ihrer Projekte über die Kommandozeile ermöglicht. Dazu werden die entsprechenden notwendigen Konfigurationsdateien automatisch generiert und angepasst.
Um geläufige Sicherheitspraktiken, insbesondere das „Principle of Least Privilege“ zu erfüllen, wurde dann evaluiert, wie die Zugriffsrechte der Nutzer, sowie die Privilegien der einzelnen Komponenten möglichst auf das notwendigste beschrankt werden können und eine konkrete Lösung für die Rechtevergabe präsentiert.
Abschließend musste noch eine Möglichkeit gefunden werden, die Zugriffslinks für die externen Forscher aufzubauen. Die Schwierigkeit lag dabei darin, das Visualizer Frontend dazu zu bringen, sich anhand der Informationen im Zugriffslink mit dem korrekten Projekt zu verbinden. Auch dafür konnte eine Lösung gefunden werden.

In Anhang B befindet sich eine detaillierte Anleitung, die basierend auf den Erfahrungen und Ergebnissen dieser Arbeit Schritt-für-Schritt erläutert, wie dieses Setup auf einer Compute-Cloud-Instanz installiert werden kann.

4.2. Ausblick

Da sich das Projekt noch in der Testphase befindet, wurde das Cloud Kontingent schlank gewählt. Es umfasst maximal eine *lrz.xlarge*-Instanz mit 10 vCPUs, 45 GiB Arbeitsspeicher und ohne Grafikprozessor. Diese VM Flavor ist aber nicht praktisch für Grafikanwendungen. Da der Instanzen ein dedizierter Grafikprozessor fehlt, müssen folglich alle Visualisierungen vom Hauptprozessor übernommen werden (Softwarerendering vs. Hardwarerendering). Dieser ist aber in keinster Weise auf die Arten von Berechnungen optimiert, die bei typischen Grafikanwendungen notwendig sind, besonders im Fall von 3D-Grafik, sodass die Berechnung eines Bildes um ein vielfaches länger braucht, als im Fall eines Grafikprozessors. Grundsätzlich bietet das LRZ jedoch Cloud-Instanzen mit Grafikbeschleunigern an. Der nächste Schritt wäre, auf eine solche Instanz umzusteigen, das Setup darauf zu installieren und schließlich die Performance, insbesondere im Fall riesiger Datensätze genauer zu evaluieren.

4.3. Mögliche Erweiterungen

Für den browserbasierten Zugriff auf die Visualisierung werden momentan vergleichsweise unsichere Zugriffslinks unter Verwendung der Projekt-ID eingesetzt, deren Geheimhaltung alleinig darin besteht, dass durch die Länge der Projekt-IDs diese nicht effizient zu erraten sind. Eine Verbesserungsmöglichkeit, die in jedem Fall mehr Flexibilität bei der Verbreitung von Zugriffslinks erlaubt, läge darin, pro Projekt mehrere passwortgeschützte, zeitbeschränkte oder nach Zugriffsanzahl beschränkte Links nach belieben erstellen zu lassen. Diese Links könnten dann auch individuell verwaltet oder daktiviert werden. Es könnte also verschiedenen Personen unterschiedliche Links gegeben werden, und somit anschließend auch der Zugang für einzelnen Personen zurückgenommen werden.
Gerade für zeitbeschränkte Zugriffslinks müsste bloß das Konfigurationsskript dahingehend erweitert werden, dass die Nutzer Zugriffslinks verwalten können und irgendwo die Assoziationen von Zugriffslinks zu Projekten gespeichert werden. Schließlich müsste nur noch der pv-session-mapper angepasst werden, sodass dieser zunächst die Gültigkeit des Zugriffslinks prüft, und anschließend das dahinterliegende Projekt ermittelt.

Ein abschließender Punkt, an dem eventulell noch Verbesserungsbedarf besteht, liegt darin, dass momentan auf der Compute-Cloud-Instanz separate Benutzerkonten notwendig sind und deren UIDs von den jeweiligen Benutzerkonten auf den Login-Nodes des Linux-Clusters kopiert werden müssen. Die Kopien selbst werden auch immer notwendig sein, da das aufgrund der Beschränkungen durch das NFS Dateisystem die einzige Möglichkeit ist, an die DSS-Dateien der jeweiligen LRZ-Nutzer zu kommen. Gleichzeitig bringen diese

aber auch einen verhältnismäßig großen Verwaltungsaufwand mit sich: Für jeden Nutzer, der den Visualisierungsdienst zur Datenverbreitung nutzen möchte, muss die UID auf dem Linux-Cluster ermittelt werden, und eine Kopie des Accounts angelegt werden. Außerdem muss irgendwie dem Nutzer, aber keinem anderen Zugriff auf den neuen Account gegeben werden. Momentan kann dafür dem Benutzer ein zufallsgeneriertes Anfangspasswort zugewiesen werden, dass der Nutzer anschließend nach der erstmaligen Anmeldung auf der Compute-Cloud-Instanz ändern kann. Das neue Passwort sollte sich gemäß üblicher Passwortrichtlinien zusätzlich noch von dem auf dem Linux-Cluster unterscheiden.
Letztendlich ist dieser gesamte Prozess nicht ideal, besonders vor dem Hintergrund, dass das LRZ bereits einen Single-Sign-On (SSO)-Dienst für Webanwendungen bereitstellt [39]. Idealerweise benötigen die LRZ-Nutzer überhaupt keinen Direktzugang per ssh zur Cloud-Instanz. Stattdessen könnte ein Webinterface entwickelt werden, auf dem sich die LRZ-Nutzer per SSO mit ihren LRZ-Zugangsdaten anmelden und direkt über den Browser ihre Projekte verwalten können. Nach Prüfung der Identität durch den Single-Sign-On-Dienst kann das Webservice-Backend bei der ersten Anmeldung automatisiert auf der Cloud-Instanz ein Benutzerkonto mit passender UID erstellen. Dafür wird die LRZ-Kennung verwendet, die vom SSO-Dienst übermittelt wurde. Da die Nutzer anschließend jegliche Verwaltung über das Webinterface vornehmen, werden die Benutzerkonten exklusiv vom System verwendet, um Zugriff auf die DSS-Dateien des jeweiligen Nutzers zu erhalten. Da sich somit die Nutzer nie direkt auf der Cloud-Instanz anmelden, wäre kein extra Passwort und auch keine Einrichtung durch einen Systemadministrator notwendig.

A. Messergebnisse

A.1. Datenverbrauch Remote-Rendering

Zur Analyse der Datennutzung des Remote-Renderings wurde bespielhaft die `can.ex2`-Datei aus den „ParaViewTestingDataFiles“ [18] verwendet. Die Datei ist ungefähr 17 MB groß. Anschließend wurde die Datei über das ParaView-Visualizer-Setup aus Abschnitt 3.4 im Browser geöffnet und über 5 Sekunden hinweg die Kamera um 180 Grad um das Modell bewegt. Als Qualitätseinstellungen für Interaktionen wurde 50% JPEG-Qualität verwendet mit voller Auflösung (1920x1036 Pixel). Währenddessen wurde mit den Werkzeugen für Entwickler des Firefox Browsers die Netzwerkaktivität aufgezeichnet und die einzelnen Pakete der WebSocket Verbindung analysiert. Während den 5 Sekunden wurden 57 Einzelbilder gesendet, wobei die durchschnittliche Bildgröße ungefähr 18 kB beträgt. In diesen 5 Sekunden durchgängiger Interaktion wurde also knapp über 1 MB an Daten empfangen. Der gleiche Vorgang wurde anschließend noch mit einer 92 MB großen Datei (nicht online verfügbar)durchgeführt. Hierbei wurden nur 27 Einzelbilder gesendet, die durchschnittliche Bildgröße war auch hier ungefähr 18 kB. Bis damit 92 MB an Daten zusammenkommen, müssten 5100 Bilder gesendet werden, bzw. über 15 Minuten lang durchgängig mit der Szene interagiert werden. Bei noch größeren Dateien ist auch der Vorteil des Remote-Renderings folglich noch größer.
Der gleiche Test wurde schließlich noch einmal durchgeführt, wobei der Parameter für die *interaktive* Bildauflösung auf 50 % reduziert wurde (960x518). Da die zu rendernde Pixelzahl somit nur noch 25 % der Ursprünglichen beträgt, konnte auch der Prozessor viel mehr Bilder in den 5 Sekunden berechnen. Gleichzeitig ist auch die Datenmenge pro Bild entsprechend gesunken. Im Endeffekt wurden in 5 Sekunden somit 104 Bilder übermittelt mit einer durchschnittlichen Bildgröße von 6,1 kB, also ungefähr 630 kB an Daten. Nebenbei lässt sich feststellen, dass 104 Bilder in 5 Sekunden in etwa 21 Bildern pro Sekunde entspricht. Das ist fast genügend Bilder, dass die Bewegung als eine durchgängige, flüssige Bewegung wahrgenommen wird (vgl. 24 Bilder pro Sekunde sind in Kinofilmen üblich). Gerade wenn Softwarerendering anstelle von hardwarebeschleunigtem Rendering verwendet wird, empfiehlt es sich folglich die interaktive Auflösung stark zu reduzieren.

A.2. Benchmark des pv-session-mappers

Für die Benchmarks des pv-session-mappers wurde ein Programm geschrieben, das zunächst einen Ordner mit 10 `<Projekt-ID>.proxy.txt`-Dateien füllt, wobei in jede Datei 1, 1000, 2000 bzw. 4000 zufällige Sitzungs-IDs geschrieben werden. Anschließend wird in einem

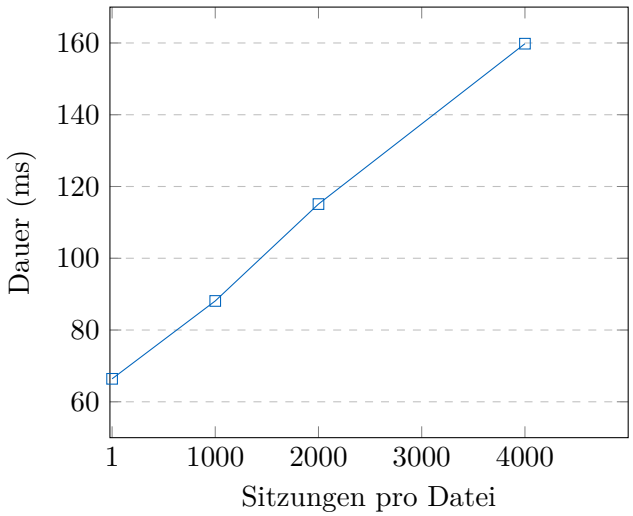


Abbildung A.1.: Durchschnittliche Zeit um 1000 Anfragen zu bearbeiten, wobei unterschiedlich viele Siztungs-ID in den `proxy.txt`-Dateien sind.

Subprozess das pv-session-mapper Programm gestartet und die Standardeingabe und -ausgabe über Pipes verbunden. Dann wird die Zeit gemessen, die benötigt wird um 1000 Anfragen bestehend aus Projekt-ID und Sitzungs-ID zu beantworten. Dieser Prozess wird 10-mal durchgeführt und der Mittelwert berechnet. Die Ergebnisse sind in Abbildung A.1 dargestellt. Wie zu erwarten ist die Antwortzeit linear abhängig von der Anzahl der Sitzungs-IDs in den einzelnen Dateien. Aus der Messung im Fall einer einzelnen Sitzungs-ID lässt sich schließen, dass in allen Messungen etwa 60 Millisekunden Overhead sind (durch die Messung oder Standardeingabe/-ausgabe und Dateisystem-I/O). Als Fazit lässt sich schließen, dass der pv-session-mapper bei Weitem schnell genug ist, um alle realistischen Szenarien zu behandeln.
Der Quellcode für die Benchmarks ist auf GitHub zu finden (Anhang C)

B. Anleitung zur Bereitstellung

Einige Dateien in diesem Tutorial werden mit curl aus der neuesten Version des GitHub Repositories unter

`https://github.com/FabianNowak/pv-visualizer-multi-project-config`

heruntergeladen. Alternativ können diese Dateien auch auf einen lokalen Computer heruntergeladen werden und anschließend per scp o.Ä. auf die Cloud-Instanz hochgeladen werden. Außerdem sind die Skripte, Konfigurationsvorlagen und der Quellcode für den pv-session-mapper auch in Anhang C angehängt.

Erstellen einer LRZ Compute Cloud Instanz

Für diesen Schritt kann dem Tutorial auf `https://doku.lrz.de/create-a-vm-10746085.html` gefolgt werden. In Schritt 7 des Online-Tutorials soll das gewünschte Betriebssystem ausgewählt werden. Hier sollte bei „Select Boot Source“ „Image“ ausgewählt werden und anschließend aus der „Available“ Liste „Ubuntu-22.04-jammy“ (die aktuelle Long-Term-Support Ubuntu-Version). Der Standardwert für „Volume Size (GB)“ ist mit 20 GB zunächst ausreichend und kann nach Bedarf angepasst werden.
Im „Flavor“-Tab in Schritt 8 kann ausgewählt werden wie viele Rechenressourcen der Instanz zur Verfügung gestellt werden. Für beste Performance sollte hier ein Profil mit Grafikprozessor ausgewählt werden. Wenn große Datensätze visualisiert werden sollen, ist auch eine große Menge an RAM vorteilhaft. Da de ParaView Software den Visualisierungsvorgang soweit möglich parallelisiert durchführt, kann sie auch von vielen CPUs profitieren.
Anschließend kann den restlichen Schritten des Online-Tutorials gefolgt werden. Am Ende sollte es möglich sein, sich per SSH und dem Private-Key auf der Instanz einzuloggen. Der Nutzername ist standardmäßig `ubuntu`.

Notwendige Pakete Installieren

```
$ sudo apt update
$ sudo apt install apache2
$ sudo apt install python3 python3-venv python3-pip
$ sudo apt install libglapi-mesa
```

`libglapi-mesa` ist eine freie Implementierung der GL API (eine Grafikschnittstelle), deren Installation notwendig ist, wenn kein Grafikprozessor mit zugehörigen proprietären Grafiktreibern vorhanden ist. Falls die Compute-Cloud Instanz aber beispielsweise über

eine NVIDIA-Grafikkarte verfügt, sollte stattdessen auf die Grafiktreiber von NVIDIA zugegriffen werden, die eine eigene Implementierung der GL API.

Installation von ParaView

Die Linux-Version von ParaView Version 5.11 ist hier verfügbar: `https://www.paraview.org/download/?version=v5.11&filter=Linux`. Unter der Überschrift „ParaView Server for Headless Machines“ sind die ParaView Serverversionen gelistet.

Für jede ParaView Version gibt es zwei Varianten. Wenn der Server einen Grafikprozessor hat, sollte die „egl“-Variante in Kombination mit den proprietären Grafiktreibern verwendet werden, ansonsten die „osmesa“-Variante, die anstelle von Hardwarerendering auf Softwarerendering zurückgreift. Schritte zur Installation:

1. Die gewünschte Version rechtsklicken und die Link-Adresse kopieren.
2. Mit

   ```
   $ curl -fSLOJ 'https://www.paraview.org/paraview-downloads/download.php?submit
     =Download&version=v5.11&type=binary&os=Linux&downloadFile=ParaView-5.11.2-
     osmesa-MPI-Linux-Python3.9-x86_64.tar.gz'
   ```

   die `tar.gz`-Datei herunterladen, hierbei den kopierten Link verwenden und mit einfachen Anführungszeichen umschließen (Zeilenumbrüche evtl. entfernen).
3. Mit

   ```
   $ tar -xf 'ParaView-5.11.2-osmesa-MPI-Linux-Python3.9-x86_64.tar.gz'
   ```

   die heruntergeladene `tar.gz`-Datei extrahieren. Der exakte Dateiname ist abhängig von der gewählten ParaView Version.
4. Anschließend mit

   ```
   $ sudo cp -r 'ParaView-5.11.2-osmesa-MPI-Linux-Python3.9-x86_64' /opt/paraview
   ```

   den gerade extrahierten Ordner mit Root-Rechten nach `/opt/paraview` kopieren.
5. Optional können nun die heruntergeladene `tar.gz`-Datei und der extrahierte Ordner wieder gelöscht werden.

6. Alternativ kann ParaView auch selbst kompiliert werden, eventuell müssen aber in dem Fall die Dateipfade in den Konfigurationsdateien aus dieser Anleitung angepasst werden.

Systembenutzer und Gruppen erstellen

1. Mit

   ```
   $ sudo adduser --group --system apache-proxy
   $ sudo adduser --group --system pv-launcher
   $ sudo adduser --group --system pv-session-mapper
   ```

   drei Systembenutzer und zugehörige Gruppen für die einzelnen Komponenten erstellen

2. Mit

   ```
   $ sudo addgroup pv-publisher
   ```

   eine Gruppe erstellen, die später an alle Nutzer verwendet wird, die ParaView Visualizer Projekte erstellen dürfen.

pv-session-mapper installieren

1. Mit

   ```
   $ curl -fSLOJ https://github.com/FabianNowak/pv-visualizer-multi-project-
       config/releases/latest/download/pv-session-mapper
   ```

   die pv-session-mapper Executable herunterladen.

2. Anschließend mit

   ```
   $ sudo mkdir /opt/pv-session-mapper/
   $ sudo cp pv-session-mapper /opt/pv-session-mapper
   $ sudo chmod +x /opt/pv-session-mapper/pv-session-mapper
   ```

   ein neues Verzeichnis in `/opt` erstellen und die Datei hineinkopieren.

Konfigurationsskript installieren

1. Mit

   ```
   $ curl -fSLOJ https://raw.githubusercontent.com/FabianNowak/pv-visualizer-
       multi-project-config/releases/configurator/create_config.py
   $ curl -fSLOJ https://raw.githubusercontent.com/FabianNowak/pv-visualizer-
       multi-project-config/releases/configurator/create_config.sh
   $ curl -fSLOJ https://raw.githubusercontent.com/FabianNowak/pv-visualizer-
       multi-project-config/releases/configurator/create_config_sudo.sh
   ```

   das Konfigurationsskript sowie die zwei Wrapper herunterladen.

2. Anschließend mit

```
$ sudo mkdir /opt/pv-configurator
$ sudo cp create_config.py /opt/pv-configurator
$ sudo cp create_config.sh /opt/pv-configurator
$ sudo cp create_config_sudo.sh /opt/pv-configurator
$ sudo chmod +x /opt/pv-configurator/create_config.sh
$ sudo chmod +x /opt/pv-configurator/create_config_sudo.sh
```

ein neues Verzeichnis in `/opt` erstellen, die Dateien hineinkopieren und die zwei bash Skripte ausführbar machen.

3. Mit

```
$ sudo -i
$ python3 -m venv /opt/pv-configurator/venv
$ source /opt/pv-configurator/venv/bin/activate
$ pip install filelock
$ exit
```

ein Python Environment erstellen und aktivieren, in das dann die eine Abhängigkeit des Konfigurationsskripts installiert wird.

wslink-Launcher installieren

1. Mit

```
$ curl -fSLOJ https://raw.githubusercontent.com/FabianNowak/pv-visualizer-
    multi-project-config/releases/launcher/launcher.py
```

das Launcher-Skript herunterladen.

2. Anschließend mit

```
$ sudo mkdir /opt/pv-launcher
$ sudo cp launcher.py /opt/pv-launcher
$ sudo chmod +x /opt/pv-launcher/launcher.py
```

ein neues Verzeichnis in `/opt` erstellen, die Datei hineinkopieren und ausführbar machen.

3. Mit

```
$ sudo -i
$ python3 -m venv /opt/pv-launcher/venv
$ source /opt/pv-launcher/venv/bin/activate
$ pip install wslink
$ exit
```

ein Python Environment erstellen und aktivieren, in das dann die Kitware wslink Bibliothek installiert wird.

Das Arbeitsverzeichnis für das Konfigurationsskript anlegen

1. ```
   $ sudo mkdir /srv/pv-configurator
   $ sudo touch /srv/pv-configurator/launchers.txt
   ```
   ```

```
$ sudo chown root:apache-proxy /srv/pv-configurator/launchers.txt
$ sudo chmod 640 /srv/pv-configurator/launchers.txt
```

um das Verzeichniss zu erstellen, wo das Konfigurationsskript die Werte speichert und eine leere `launchers.txt` mit passenden Berechtigungen anzulegen.

2. Anschließend mittels `$ ls /opt/paraview/share` die Inhalte des Verzeichnisses anzeigen lassen. Darin sollte sich ein Verzeichnis namens `paraview-<version>` befinden, wobei `<version>` der installierten ParaView Version (oder einem Teil davon) entspricht. Diesen Verzeichnisnamen merken.

3. Mit

   ```
   $ sudoedit /srv/pv-configurator/configurator_settings.json
   ```

   eine `configurator_settings.json` Datei anlegen und mit Root-Rechten bearbeiten.

4. In die `configurator_settings.json` Datei muss folgendes eingetragen werden:

   ```
   {
       "servername": "<ip-oder-hostname>",
       "python_exec": "/opt/paraview/bin/pvpython",
       "visualizer_exec": "/opt/paraview/share/paraview-<version>/web/visualizer/
           ↪ server/pvw-visualizer.py",
       "launcher_exec": "/opt/pv-launcher/launcher.py"
   }
   ```

   `<ip-oder-hostname>` muss die Floating-IP Adresse (oder ein Hostname) sein, unter der der Server von außen erreichbar ist. Wenn nicht der HTTP Standardport 80 verwendet wird, muss auch der Port enthalten sein.

   Beim `visualizer_exec` Feld muss darauf geachtet werden, dass der Verzeichnisname aus dem vorherigen Schritt verwendet wird.

Sudoers Datei anpassen

Mit

```
$ sudo visudo /etc/sudoers.d/pv-sudoers
```

eine neue Ergänzung zur sudoers Konfiguration erstellen und die Datei zur Bearbeitung öffnen. **Änderungen immer nur mit visudo vornehmen, da ansonsten im Fall eines Syntaxfehlers sudo nicht mehr verwendet werden kann.** Visudo warnt, falls versucht wird eine fehlerhafte Datei zu speichern. Dann folgende drei Zeilen einfügen:

```
%pv-publisher ALL = (root:root) /opt/pv-configurator/create_config.sh
pv-launcher  ALL = (%pv-publisher) NOPASSWD: /opt/paraview/bin/pvpython --dr \
    /opt/paraview/share/paraview-<version>/web/visualizer/server/pvw-visualizer.py *
```

Auch hier muss bei `paraview-<version>` der Verzeichnisname verwendet werden, der im vorherigen Abschnitt ermittelt wurde. Die erste Zeile erlaubt allen Nutzern der `pv-publisher` Gruppe die `create_config.sh` Datei auszuführen. Die zweite Zeile erlaubt dem `pv-launcher` den Befehl zum Starten des Visualizer-Backends mit den Rechten eines beliebigen Nutzers

aus der `pv-publisher` Gruppe auszuführen, sodass Dateizugriff auf die DSS-Dateien des entsprechenden Nutzers möglich ist.

Symbolische Verknüpfungen erstellen

```
$ sudo ln -s /opt/pv-configurator/create_config_sudo.sh /usr/local/bin/pvconfig
$ sudo ln -s /opt/paraview/share/paraview-<version>/web/visualizer/www /srv/
    ↪ visualizer-www
```

Um erstens den Nutzern zu ermöglichen das Konfigurationsskript per `pvconfig` aufzurufen und zweitens die Dateien für das Browser-Frontend per `/srv/visualizer-www` Verzeichnis bereitzustellen.

Apache-Server einrichten

1. (Optional) `/etc/apache2/ports.conf` mit `sudoedit` bearbeiten und die standardmäßig enthaltene Zeile

   ```
   Listen 80
   ```

   zu einem anderen Port ändern, wenn nicht der HTTP Standardport (80) verwendet werden soll

2. `/etc/apache2/envvars` mit `sudoedit` öffnen, nach den Zeilen

   ```
   export APACHE_RUN_USER=www-data
   export APACHE_RUN_GROUP=www-data
   ```

   suchen und `www-data` mit `apache-proxy` ersetzen um festzulegen, dass die Prozesse des Apache-Servers mit den Rechten des `apache-proxy` Nutzers ausgeführt werden sollen.

3. Mit

   ```
   $ curl -fSLOJ https://raw.githubusercontent.com/FabianNowak/pv-visualizer-
       multi-project-config/releases/apache-config/paraview-multi-project.conf
   $ sudo cp paraview-multi-project.conf /etc/apache2/conf-available/
   ```

   die Vorlage für die Apache-Konfigurationsdatei herunterladen und in den Konfigurationsordner kopieren

4. Dann mit

   ```
   $ sudoedit /etc/apache2/conf-available/paraview-multi-project.conf
   ```

   die Datei bearbeiten und in der Zeile

   ```
   Define SERVER_NAME <ip-oder-name>
   ```

   `<ip-oder-name>` mit der Floating-IP ersetzen, die der Compute-Cloud-Instanz zugewiesen wurde und die Datei speichern.

5. Dann mit

   ```
   $ sudo a2dissite 000-default
   $ sudo a2enconf paraview-multi-project
   ```

```
$ sudo a2enmod rewrite proxy proxy_http
```

einen Teil der Standardkonfiguration deaktivieren und die neue Konfiguration und die benötigten Zusatzmodule laden.

6. Mit `$ sudo systemctl restart apache2` den Apache-Server neu starten um alle Konfigurationsänderungen zu laden.

Zu diesem Zeitpunkt können bereits Daten visualisiert werden.

DSS Container mounten

Hierfür kann dem Punkt „DSS Container NFS Export Creation“ im Tutorial unter `https://doku.lrz.de/dss-how-to-export-a-container-via-nfs-to-your-virtual-machine-in-lrz-11484494.html`
gefolgt werden. Dabei muss der Datenkurator des DSS Containers gebeten werden, die MWN Floating-IP der Cloud-Instanz für den Export freizuschalten. Als Zugriffstyp genügt read-only.
Unter dem Punkt „Mount NFS Volume on Virtual Machine“ wird anschließend beschrieben, wie der Container dann per NFS gemountet wird. Davor muss jedoch per

```
$ sudo apt install nfs-common
```

ein weiteres Paket installiert werden, dass letztendlich das mounten von NFS Dateisystemen unterstützt.

Nutzer für DSS Zugriff erstellen

1. Hierfür muss zunächst die UID des Nutzers auf dem DSS ermittelt werden. Dafür gibt es zwei Optionen:
   a) Wie in Abschnitt „Prerequisites“ im Online-Tutorial unter `https://doku.lrz.de/dss-how-to-export-a-container-via-nfs-to-your-virtual-machine-in-lrz-11484494.html` beschrieben den Datenkurator bitten den

   ```
   (dsscli) $ dss passwd list --containername <container-name>
   ```

   Befehl auszuführen um eine Liste der aktuellen DSS-Nutzer inklusive deren UIDs und GIDs zu erhalten.

   b) Oder, falls möglich, sich auf der Login-Node des LRZ Linux Clusters einloggen und mit `cd` das Verzeichnis des DSS-Containers auf der Login-Node betreten. Das Verzeichnis hat normalerweise die Form

   ```
   /dss/<filesystem name>/<data project>/<data container>/
   ```

   In diesem Verzeichniss dann

   ```
   $ ls -n
   ```

   ausführen. Im Idealfall wird eine Liste an Verzeichnissen zurückgeliefert die alle Nutzer des Containers beinhaltet. Eine Zeile davon schaut in etwa so aus:

```
drwx------ 4 1234567 7654321 4096 Oct 26 10:09 ab12cde3
```

Wobei in diesem Beispiel `1234567` die UID des Nutzers `ab12cde3` ist und `7654321` die GID der Containergruppe (typischerweise identisch für alle Mitglieder des Containers).

2. Anschließend wieder auf der LRZ Compute Cloud Instanz einloggen.
3. (Optional) Mit

   ```
   $ sudo addgroup --gid <GID> <container-name>
   ```

   die Containergruppe replizieren.
4. Dann mit

   ```
   $ sudo adduser --uid <UID> <username>
   ```

   den DSS-Nutzer replizieren. `<UID>` ist dabei die in Schritt 1 ermittelte UID, und `<username>` der LRZ-Benutzername des gewünschten Users. Optional kann im Befehl noch `-gid <GID>` hinter `<UID>` eingefügt werden, falls auch die Containergruppe in Schritt 3 repliziert wurde.

   Der Befehl fragt nach einem Passwort. Falls es sich nicht um den eigenen LRZ-Nutzer handelt, kan hier ein Anfangspasswort bestimmt werden, das der entsprechende LRZ-Nutzer nach dem ersten Login selbstständig ändert.
5. Damit der neu angelegte Nutzer das `pvconfig` Konfigurationsskript nutzen kann, muss er noch per

   ```
   $ sudo adduser <username> pv-publisher
   ```

   der Gruppe hinzugefügt werden.

Passwort-Login per SSH

Wenn schließlich mehrere LRZ-Nutzer sich selbstständig einloggen sollen, muss ihnen ein Login per `ssh` ermöglicht werden. Dazu muss die sshd Konfigurationsdatei angepasst werden:

```
$ sudoedit /etc/ssh/sshd_config
```

und nach der Zeile mit

```
PasswordAuthentication no
```

gesucht werden. Hier gibt es verschiedene Ansätze, womit die Zeile ersetzt werden kann.

1. ```
   PasswordAuthentication no
   Match Group pv-publisher
       PasswordAuthentication yes
   Match all
   ```

   in die Datei einfügen, damit für alle Nutzer in der `pv-publisher` Gruppe ein passwortbasierter ssh-Login erlaubt ist.

2. Zuerst in der Kommandozeile mit
   ```

```
$ sudo addgroup lrz-user
$ ...
```

eine separate Gruppe erstellen.

Anschließend

```
PasswordAuthentication no
Match Group lrz-user
    PasswordAuthentication yes
Match all
```

in die Datei einfügen, damit für alle Nutzer in der `lrz-user` Gruppe ein passwortbasierter ssh-Login erlaubt ist.

Dann noch alle replizierten LRZ-Nutzer dieser Gruppe hinzufügen (`sudo adduser <user> <group>`). Dadurch wird die Berechtigung sich per Passwort einzuloggen von der Berechtigung Visualisierungen bereitzustellen logisch getrennt.

3. ```
   PasswordAuthentication yes
   Match User ubuntu,root,testuser #,... usw.
       PasswordAuthentication no
   Match all
   ```

   einfügen, um Grundsätzlich passwortbasiertes Login zu erlauben, außer für eine Auswahl an Benutzern (insb. mit Administrativen Rechten).

Danach noch das sshd Programm neu starten per

```
$ sudo systemctl restart sshd
```
```

C. Quellcodes

Der Quellcode des gesamten Projekts kann auf GitHub unter

`https://github.com/FabianNowak/pv-visualizer-multi-project-config`

eingesehen werden.

D. Anleitung: DSS-Verzeichnis mittels bindfs mit anderen Nutzern auf der Compute-Cloud-Instanz teilen

D.1. bindfs

`bindfs` ist ein FUSE (Filesystem in Userspace) Dateisystem, mit dem ein Verzeichnis an einen anderen Mountpunkt gespiegelt werden kann. Dabei können die Dateiberechtigungen im neuen Mountpunkt verändert werden. Dieses Tool kann verwendet werden, um einem anderen Nutzer einer Compute-Cloud-Instanz Zugriff auf die eigenen DSS-Dateien zu ermöglichen. Wenn nun ein Nutzer *A* allen Nutzern in der `dss-access-A` Gruppe erlauben möchte, auf das `beispiel`-Verzeichnis in seinem DSS-Ordner zuzugreifen muss *genau dieser Nutzer A* folgenden Befehl ausführen:

```
$ bindfs -g dss-access-A -p u+rx,g+rx --realistic-permissions
      ↪  /dss/a1234/A/beispiel /sharedss/access-A/
```

Wenn von `/sharedss/access-A` aus zugegriffen wird, sieht es durch das Befehlsargument `-g dss-access-A` so aus, als wäre *A* der Inhaber und `dss-access-A` die Inhabergruppe, obwohl eigentlich der Nutzer *A* und die `a1234`-Container-Gruppe die Inhaber sind.

Beispiel:

```
$ ls -l /dss/a1234/A/beispiel
-rw------- ... A a1234 ... beispieldatei
```

aber

```
$ ls -l /sharedss/access-A
-r-xr-x--- ... A dss-access-A ... beispieldatei
```

Über `access-A` kann *B* dann so auf die Dateien zugreifen, als wäre `dss-access-A` die Inhabergruppe. Wenn *B* versucht eine Datei zu öffnen, sorgt das `bindfs` Dateisystem dafür, dass der eigentliche Dateizugriff von *A* auf dem originalen Pfad durchgeführt wird. *B* kann folglich nur auf Dateien zugreifen, auf die *A* zugreifen darf.
Mit dem `-p u+rx,g+rx` Befehlsargument wird festgelegt, dass im gespiegelten Dateisystem Eigentümer und Gruppe auf alle Dateien nur Lese- und Ausführberechtigungen haben. Hier könnten auch noch Schreibberechtigungen vergeben werden. Insgesamt gilt aber, dass *B* nur das darf, was *A* auch darf.

Wenn beispielsweise stattdessen `-p +rwx` verwendet wird (alle Berechtigungen gesetzt), aber die echten Berechtigungen gleichzeitig so aussehen (nur Leseberechtigung für Eigentümer):

```
$ ls -l /dss/a1234/A/beispiel
-r-------- ... A a1234 ... beispieldatei
```

kann B trotzdem nicht in die `beispieldatei` schreiben, obwohl für B die Berechtigungen zunächst so aussehen:

```
$ ls -l /sharedss/access-A
-rwxrwxrwx ... A dss-access-A ... beispieldatei
```

Da die eigentlichen Dateizugriffe im Hintergrund von A durchgeführt werden, wird somit in diesem Fall der Zugriff abgelehnt.

Der `--realistic-permissions` Parameter sorgt dafür, dass für jede Datei nur höchstens die Berechtigungen angezeigt werden, die Nutzer A wirklich hat.
Wenn eine Datei also nur `rw` Berechtigungen für den Eigentümer hat, aber `-p g+rx` angegeben wird, wird letztendlich nur noch das `r`-Berechtigungsbit beibehalten. Somit kann in Kombination mit dem `-p`-Parameter erzielt werden, dass nur genau die Dateien die ursprünglich von Nutzer A ausführbar waren, im gespiegelten System ausführbar werden. Wenn im `beispiel` Verzeichnis eine Datei liegt, die A nicht gehört und auf die A überhaupt nicht zugreifen darf (z.B. `-rwx------ root root`), wird dem Nutzer B als Berechtigungen `--------- A dss-access-A` angezeigt, obwohl `-p u+rx,g+rx` gesetzt ist. Insgesamt wird durch den Parameter sichergestellt, dass B nur Berechtigungen sieht, die er so auch wirklich verwenden kann.
Mit `fusermount -u /sharedss/access-A` kann das Verzeichnis anschließend umounted werden.

D.2. sharedss Konfigurationsskript

Um das Teilen eines Verzeichnisses per `bindfs` zu vereinfachen, wurde mehrere Bash-Skripts entwickelt, die im Hintergrund die nötigen Konfigurationen und Befehle ausführen. Es unterstützt folgende unterbefehle:

- `$ sharedss share <unterordner>` teilt das Verzeichnis `<unterordner>`. `<unterordner>` muss dabei ein Pfad relativ zum eigenen Benutzerverzeichnis auf dem DSS sein.
- `$ sharedss unshare` stoppt das Teilen des aktuell geteilten Verzeichnisses.
- `$ sharedss allow <username>` erlaubt `<username>` den Zugriff auf das geteilte Verzeichnis
- `$ sharedss revoke <username>` entzieht `<username>` die Zugriffsberechtigungen.
- `$ sharedss list` gibt alle Nutzer aus, die momentan Zugriff haben.

Alle Unterbefehle wurden in separaten Skripten implementiert, die alle mit `sudo` aufgerufen werden müssen. Dementsprechend muss allen Nutzern, die das `sharedss` Skript verwenden sollen, sudo-Berechtigungen für die einzelnen Skripte gegeben werden.

share

Für den `share` Unterbefehl wird eine `dss-access-<eigentümer>` Gruppe erstellt. Anschließend wird ein leeres `/sharedss/<eigentümer>` Verzeichnis angelegt, das als Mountpoint dient. Danach werden noch die Inhaber angepasst, sodass `<eigentümer>` der Eigentümer ist, und `dss-access-<eigentümer>` die Inhabergruppe. Dann wird mit `sudo -u <eigentümer>` der `bindfs` Befehl mit Rechten von `<eigentümer>` und den Parametern aus Abschnitt D.1 ausgeführt. Der angegebene Unterordner wird somit an `/sharedss/<eigentümer>` eingehängt.
Abschließend wird noch in die Datei `/var/lib/sharedss/mounts/<eigentümer>` der absolute Pfad zum geteilten Unterordner geschrieben. Ein weiteres Skript, das per `systemd` so konfiguriert werden muss, das es beim Systemstart ausgeführt wird, liest aus all diesen Dateien, und führt die `bindfs` Befehle nach einem Systemneustart aus. Somit wird sichergestellt, dass die geteilten Verzeichnisse auch nach einem Neustart weiterhin geteilt werden.

unshare

Für den `unshare` Unterbefehl werden alle Mitglieder aus der `dss-access-<eigentümer>` Gruppe entfernt. Anschließend wird mit `fusermount -u /sharedss/<eigentümer>` das bindfs-Dateisystem ausgehängt. Schließlich wird das `/sharedss/<eigentümer>` gelöscht.

allow/revoke

Für die Umsetzung der `allow/revoke` Unterbefehle wird der angegebene Nutzer der `dss-access-<eigentümer>` hinzugefügt bzw. aus der Gruppe entfernt.

list

Für den `list` Unterbefehl werden alle Mitglieder der `dss-access-<eigentümer>` Gruppe aufgelistet.

D.3. Einrichten des sharedss-Skripts

Notwendige Pakete Installieren

```
$ sudo apt update
$ sudo apt install bindfs
```

Die einzelnen Skripte installieren

1. Mit

   ```
   $ curl -fSLOJ https://github.com/FabianNowak/pv-visualizer-multi-project-
       config/releases/latest/download/dss-share-scripts.tar.gz
   ```

   die aktuelle Version der sharedss-Skripte herunterladen.

2. Mit

   ```
   $ tar -xf 'dss-share-scripts.tar.gz'
   ```

   die heruntergeladene `tar.gz`-Datei extrahieren.

3. Anschließend mit

   ```
   $ sudo mkdir /opt/sharedss/
   $ sudo cp dss-share-scripts /opt/sharedss/
   ```

   ein neues Verzeichnis in `/opt` erstellen und die extrahierten Skripte hineinkopieren.

4. Dann die `/opt/sharedss/sharedss.sh`-Datei mit `sudoedit` bearbeiten und in der Zeile

   ```
   dss_mountpath=
   ```

   den Pfad, wo der DSS-Container gemountet ist angeben, z.B:

   ```
   dss_mountpath="/dss/t1234"
   ```

5. Mit

   ```
   $ sudo chmod +x /opt/sharedss/*
   ```

   alle Skripte ausführbar machen.

6. Mit

   ```
   $ sudo ln -s /opt/sharedss/sharedss.sh /usr/local/bin/sharedss
   ```

   eine symbolische Verknüpfung in `/usr/local/bin` anlegen, sodass das Skript direkt per `$ sharedss` aufgerufen werden kann.

Notwendige Verzeichnisse erstellen

```
$ sudo mkdir /sharedss
$ sudo mkdir /var/lib/sharedss/
$ sudo mkdir /var/lib/sharedss/mounts
```

In `/sharedss` sind dann die geteilten Verzeichnisse verfügbar. In `/var/lib/sharedss/mounts` wird gespeichert, welche Verzeichnisse momentan geteilt werden.

Sudoers Datei anpassen

Mit

```
$ sudo visudo /etc/sudoers.d/sharedss
```

eine neue Ergänzung zur sudoers Konfiguration erstellen und die Datei zur Bearbeitung öffnen. **Änderungen immer nur mit visudo vornehmen, da ansonsten im Fall eines Syntaxfehlers sudo nicht mehr verwendet werden kann.** Visudo warnt, falls versucht wird eine fehlerhafte Datei zu speichern. Dann folgende Zeilen einfügen:

```
%lrz-user  ALL = (root:root) /opt/sharedss/mount-for-all.sh
%lrz-user  ALL = (root:root) /opt/sharedss/allow-access.sh
%lrz-user  ALL = (root:root) /opt/sharedss/revoke-access.sh
%lrz-user  ALL = (root:root) /opt/sharedss/list-group.sh
```

```
%lrz-user  ALL = (root:root) /opt/sharedss/unmount.sh
```

damit wird allen Nutzern der Gruppe `lrz-user` erlaubt, die für `sharedss` notwendigen Skripte mit Root-Rechten auszuführen. Alternativ kann hier auch ein anderer Gruppenname verwendet werden. Damit also die Nutzer das sharedss-Tool benutzen können, müssen sie per

```
$ sudo adduser <username> lrz-user
```

zu der Gruppe hinzugefügt werden.
Eventuell muss davor die Gruppe erstellt werden:

```
$ sudo addgroup lrz-user
```

/etc/fuse.conf anpassen

Die Datei `/etc/fuse.conf` mit `sudoedit` öffnen, und bei der Zeile

```
#user_allow_other
```

das # entfernen oder eine neue Zeile mit

```
user_allow_other
```

einfügen.
Damit wird erlaubt, dass Nutzer Verzeichnisse so mounten dürfen, dass danach auch andere Nutzer zugreifen können.

Automatische Wiederherstellung der bindfs-Mounts nach Systemneustart

1. Mit

   ```
   $ curl -fSLOJ https://raw.githubusercontent.com/FabianNowak/pv-visualizer-
       multi-project-config/releases/dss-share/mount-after-reboot.service
   ```

   die systemd-Service-Unit-Datei herunterladen. Die Systemd-Unit ist so konfiguriert, dass sie beim Systemstart das in einem vorherigen Schritt installierte `/opt/sharedss/mount-after-reboot.sh` Skript ausführt. Das Skript liest die Dateien aus `/var/lib/sharedss/mounts`. Basierend auf den enthaltenen Informationen werden die bisher geteilten Verzeichnisse wieder gemountet.

2. Mit

   ```
   $ sudo cp mount-after-reboot.service /opt/sharedss
   ```

   die Datei in das `/opt/sharedss` Verzeichnis kopieren.

3. Abschließend mit

   ```
   $ sudo systemctl enable /opt/sharedss/mount-after-reboot.service
   ```

   die systemd-Unit aktivieren.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff und M. Frame, „Data Sharing by Scientists: Practices and Perceptions“, *PLOS ONE*, Jg. 6, Nr. 6, e21101, Juni 2011. DOI: 10.1371/journal.pone.0021101.

[2] H. A. Piwowar und T. J. Vision, „Data reuse and the open data citation advantage“, *PeerJ*, Jg. 1, e175, 2013. DOI: 10.7717/peerj.175.

[3] J. Anderson, *Hypersonic and High Temperature Gas Dynamics*, 2. Aufl. American Institute of Aeronautics and Astronautics, 2000, ISBN: 978-1-56347-780-5.

[4] T. Sterling, M. Anderson und M. Brodowicz, *High performance computing, Modern systems and practices*. Cambridge, MA: Morgan Kaufmann, 2018, ISBN: 9780124202153.

[5] Oak Ridge National Laboratory. „Frontier“. (2022), Adresse: https://www.olcf.ornl.gov/frontier/ (besucht am 10. 01. 2024).

[6] Oak Ridge National Laboratory, *New 200-Petaflops System Debuts as America's Top Supercomputer for Science*, 2018. Adresse: https://www.ornl.gov/news/ornl-launches-summit-supercomputer (besucht am 10. 01. 2024).

[7] N. K. Sehgal und P. C. P. Bhatt, *Cloud Computing*. Springer International Publishing, 2018, ISBN: 978-3-319-77838-9. DOI: 10.1007/978-3-319-77839-6.

[8] Gartner. „Public cloud services end-user spending worldwide from 2017 to 2024 (in billion U.S. dollars)“, [Graph]. zitiert nach www.statista.com. (2023), Adresse: https://www.statista.com/statistics/273818/global-revenue-generated-with-cloud-computing-since-2009/ (besucht am 10. 01. 2024).

[9] P. M. Mell und T. Grance, *The NIST definition of cloud computing*, 2011. DOI: 10.6028/NIST.SP.800-145.

[10] D. Dempsey und F. Kelliher, *Industry Trends in Cloud Computing*. Springer International Publishing, 2018, ISBN: 978-3-319-63993-2. DOI: 10.1007/978-3-319-63994-9.

[11] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang und A. Ghalsasi, „Cloud computing — The business perspective“, *Decision Support Systems*, Jg. 51, Nr. 1, S. 176–189, 2011, ISSN: 0167-9236. DOI: 10.1016/j.dss.2010.12.006.

[12] TOP500, Hrsg. „TOP500 List - November 2023“. (2023), Adresse: https://www.top500.org/lists/top500/list/2023/11/ (besucht am 10. 01. 2024).

[13] Leibniz-Rechenzentrum. „Hardware of SuperMUC-NG Phase 1“, Adresse: https://doku.lrz.de/hardware-of-supermuc-ng-phase-1-11482553.html (besucht am 10. 01. 2024).

[14] Leibniz-Rechenzentrum. „CoolMUC-2“, Adresse: https://doku.lrz.de/coolmuc-2-11484376.html (besucht am 10. 01. 2024).

[15] Leibniz-Rechenzentrum. „CoolMUC-3“, Adresse: https://doku.lrz.de/coolmuc-3-11484375.html (besucht am 10. 01. 2024).

[16] Leibniz-Rechenzentrum. „Data Science Storage“, Adresse: https://doku.lrz.de/data-science-storage-10745685.html (besucht am 10. 01. 2024).

[17] Leibniz-Rechenzentrum. „Compute Cloud“, FAQ, Adresse: https://doku.lrz.de/faq-10745941.html (besucht am 10. 01. 2024).

[18] ParaView. „Download ParaView“, Adresse: https://www.paraview.org/download/.

[19] Khronos Group, *Khronos Native Platform Graphics Interface, EGL Version 1.5 - August 27, 2014*. Adresse: https://registry.khronos.org/EGL/specs/eglspec.1.5.pdf (besucht am 10. 01. 2024).

[20] NVIDIA. „EGL Eye: OpenGL Visualization without an X Server“. (2016), Adresse: https://developer.nvidia.com/blog/egl-eye-opengl-visualization-without-x-server/ (besucht am 10. 01. 2024).

[21] Mesa 3D. „The Mesa 3D Graphics Library latest Documentation, Off-screen Rendering“, Adresse: https://docs.mesa3d.org/osmesa.html (besucht am 10. 01. 2024).

[22] ParaView Reference Manual, *7. Remote and parallel visualization, 7.8. ParaView architecture*. Adresse: https://docs.paraview.org/en/latest/ReferenceManual/parallelDataVisualization.html#paraview-architecture (besucht am 10. 01. 2024).

[23] Kitware, *Documentation | Visualizer*. Adresse: https://kitware.github.io/visualizer/docs/ (besucht am 10. 01. 2024).

[24] Netcraft. „November 2023 Web Server Survey“. (2023), Adresse: https://www.netcraft.com/blog/november-2023-web-server-survey/ (besucht am 10. 01. 2024).

[25] Apache HTTP Server, *mod_rewrite*, Version 2.4. Adresse: https://httpd.apache.org/docs/2.4/mod/mod_rewrite.html (besucht am 10. 01. 2024).

[26] Kitware, *wslink*. Adresse: https://github.com/Kitware/wslink/tree/master/python (besucht am 10. 01. 2024).

[27] Kitware. „ParaViewWeb, Launcher API Documentation“, Adresse: https://kitware.github.io/paraviewweb/docs/launcher_api.html (besucht am 10. 01. 2024).

[28] Leibniz-Rechenzentrum. „DSS documentation for users“, Adresse: https://doku.lrz.de/dss-documentation-for-users-11476038.html (besucht am 10. 01. 2024).

[29] Leibniz-Rechenzentrum. „DSS documentation for data curators“, Adresse: https://doku.lrz.de/dss-documentation-for-data-curators-11476063.html (besucht am 10. 01. 2024).

[30] Kitware. „ParaViewWeb, Multi-user Setup Guide“, Adresse: https://kitware.github.io/paraviewweb/docs/multi_user_setup.html (besucht am 10. 01. 2024).

[31] Kitware. „ParaViewWeb, Using Apache as a front end“, Adresse: https://kitware.github.io/paraviewweb/docs/apache_front_end.html (besucht am 10. 01. 2024).

[32] Kitware. „ParaViewWeb, How to use the Python Launcher“, Adresse: https://kitware.github.io/paraviewweb/docs/python_launcher.html (besucht am 10. 01. 2024).

[33] Apache HTTP Server, *mod_proxy*, Version 2.4. Adresse: https://httpd.apache.org/docs/2.4/mod/mod_proxy.html (besucht am 10. 01. 2024).

[34] Kitware, *launcher.py aus wslink Version 1.12.4*. Adresse: https://github.com/Kitware/wslink/blob/v1.12.4/python/src/wslink/launcher.py (besucht am 10. 01. 2024).

[35] J. Blandy und J. Orendorff, *Programming Rust, Fast, safe systems development*, First edition. O'reilly, 2018, ISBN: 9781491927274.

[36] The Chromium Projects. „Memory safety“, Adresse: https://www.chromium.org/Home/chromium-security/memory-safety/ (besucht am 10. 01. 2024).

[37] The Chromium Projects. „Rust and C++ interoperability“, Adresse: https://www.chromium.org/Home/chromium-security/memory-safety/rust-and-c-interoperability/ (besucht am 10. 01. 2024).

[38] J. H. Saltzer und M. D. Schroeder, „The protection of information in computer systems“, *Proceedings of the IEEE*, Jg. 63, Nr. 9, S. 1278–1308, 1975, ISSN: 0018-9219. DOI: 10.1109/PROC.1975.9939.

[39] Leibniz-Rechenzentrum. „Shibboleth für eigene Webanwendungen“, Adresse: https://doku.lrz.de/shibboleth-fuer-eigene-webanwendungen-10333379.html (besucht am 10. 01. 2024).