technische universität
dortmund

Diplomarbeit

Verwaltung verteilter Speicherkapazität für wissenschaftliche Daten

Long Duc Phan

23 November 2012

Gutachter:
Prof. Dr. Bernhard Steffen
Dr. Thomas Röblitz

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 5 für Programmiersysteme (LS-5)
http://ls5-www.cs.tu-dortmund.de

Danksagung

Ich möchte mich bei Herrn Dr. Thomas Röblitz und Herrn Prof. Dr. Bernhard Steffen und die Mitarbeiter vom Lehrstuhl 5 für ihren vielseitigen fachlichen Rat, Diskussion bedanken. Ich habe viel von der Diskussion gelernt, dies hat mich während der Diplomarbeit motiviert. Herrn Dr. Thomas Röblitz möchte ich einen besonderen Dank für seine intensive Betreuung aussprechen. Er hat mir viel Tipps, Hinweise in jeder Phase dieser Arbeit begleitet und geholfen.

Außerdem möchte ich mich auch bei meinen Freunden Tobias, Maria, Jasmin und die Mitarbeiter vom Lehrstuhl Biologie TU Dortmund bedanken, die mir bei dem Korrigieren in deutscher Sprache geholfen haben.

Mein herzlicher Dank sowie ich diese Arbeit an meine Familie meine Mutter Lien Kim Nguyen, meine kleine Schwester Nhi Lan Ngoc Nguyen, meine verstorbene Opa Lan Ki Nguyen und verstorbene Onkel Le Ki Nguyen schicken möchte. Ihr seid meine Motivation jeden Tag für mein Auslandsstudium.

Inhaltsverzeichnis

1 Einleitung

> *„We humans have built a creativity machine. Its the sum of three things: a few hundred million computers, a communication system connecting those computers, and some millions of human beings using those computers and communications.“* Vinge Vernor, [23]

Die Informationsmenge wird daher in Zukunft stetig wachsen. Astronomie ist beispielsweise ein Forschungsbereich, wo eine sehr große Datenmenge verarbeitet werden muss. Ein Beispiel ist das globales Forschungsprojekt Square Kilometre Array(SKA) über den Aufbau eines Höchstleistungs-Teleskop-Systems. Dieses soll den Astrophysiker und Forscher detaillierte Informationen für Studien und weitere Forschungen über erste Sterne und Galaxien nach dem Big Bang liefern, also darüber wie die Galaxie sich danach entwickelt hat.

Das Projekt SKA begann mit der ersten Vorbereitungsphase ab 2008 und wird bis zum Abschluss des Aufbaus Ende 2023 fortlaufen. Das Zentrale Verarbeitungssystem wird eine astronomische Datenmenge bis zu 1 Petabyte pro 20 Sekunden verarbeiten. Die komplexe Datenarchivierung und die verteilten Systeme werden den Wissenschaftlern aus der ganzen Welt einen Zugriff auf diese große Datenmenge liefern. Dafür werden Exa-skalierbare Computing Systeme und Exabyte Daten Storage erforderlich sein[1].

Die Datenverarbeitung in der Astrophysik zeichnet sich durch komplexe Arbeitsabläufe aus und verbraucht häufig viele verteilte Ressourcen (Cluster und Supercomputer). Die einzelnen Ressourcen wie Rechenkapazität, Speicherkapazität und Netzwerkbandbreite sind oft beschränkt, voll ausgelastet und alleine nicht ausreichend für große naturwissenschaftliche Projekte. Die Rechenkapazität und Netzwerkbandbreite werden wegen der Nutzung durch einen Rechenauftrag (eng.Job) oder Datentransfer besetzt, jedoch stehen sie wieder zur Verfügung, wenn der Datentransfer oder Job ausgelaufen sind. Im Gegensatz dazu ist die Speicherkapazität dauerhaft belegt, da die gespeicherte Daten längere Zeit im Storage System abgelegt werden müssen.

Der Ablauf der verteilten Arbeitsschritte im Projekt CLUES wird in Abbildung 1.1) dargestellt. Die detaillierte Ablauf werden in 2.3 eingegangen. Die wissenschaftlichen Daten in diesem Kontext sind oft die Rohdaten, die von Satelliten aufgenommen wurden, sowie Simulationen, die von Wissenschaftlern gestartet werden und Snapshots (große Dateien in CLUES Projekt) als Ergebnisse dieser Simulationen. Alle diese Daten erfordern große Ressourcen, hohe Rechenkapazität aus Supercomputern, eine hohe Netzwerk-Bandbreite und daher auch einen großen Speicherbedarf, insbesondere wenn die Wissenschaftler im Projekt mehrere Simulationen mit unterschiedlichen Startparametern analysieren. Die

[1] The Square Kilometre Array Factsheet for Scientists and Engineers, Sci Eng SPDO April 2010.

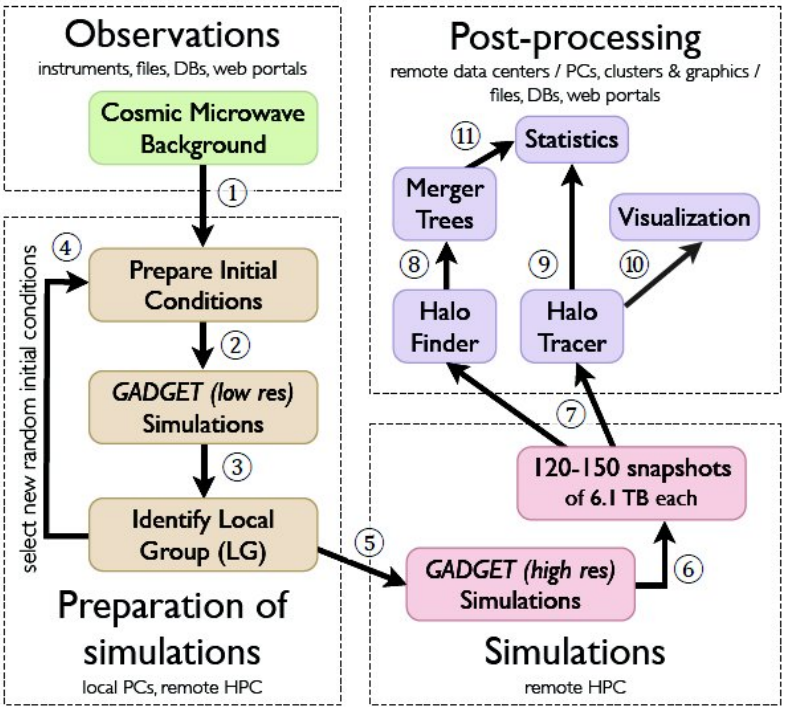

Abbildung 1.1: Verteilte Arbeitsschritte im CLUES Projekt [20].

Durchführung vieler Simulationen kann dazu führen, dass die Speicherkapazität an den Höchstleistungsrechnern erschöpft ist.
Um diesen Kapazitätsmängeln vorzubeugen müssen die Engpässe auf Datenspeichern im Storage System rechtzeitig erkannt und geeignete Gegenmaßnahmen durchgeführt werden, bevor sie zu Schäden führen können. Einige Maßnahmen, die oft von Wissenschaftler **manuell** durchgeführt werden, sind:

1. das Komprimieren einzelner oder mehrerer Snapshots,

2. das Löschen einzelner oder mehrerer Snapshots falls eine Kopie existiert,

3. das Verschieben einzelner oder mehrerer Snapshots zu anderen Datenspeichern,

4. die Wahl einer anderen Rechenressource falls (1), (2) und (3) nicht genügend freie Datenspeicherkapazität bereitstellen, oder

5. das Aufteilen einer Simulation auf mehrere Höchstleistungsrechenzentren.

Die Herausforderung für die Wissenschaftler in der Nutzung dieser Maßnahmen besteht darin, möglichst wenig Arbeitszeit aufzuwenden und gleichzeitig wichtige Merkmale wie hohe Effizienz, die Vermeidung von Datenverlust, die möglichst geringe Belastung von Recheneinheiten, sowie eine hohe Skalierbarkeit zu erreichen. Der manuelle Einsatz dieser Maßnahmen bei immer größeren Datenmengen ist daher nicht mehr ausreichend für

die wissenschaftliche Datenverarbeitung in verteilten Umgebungen. Insbesondere muss erkannt werden, wann ein Engpass eingetreten ist (oder besser: eintreten wird) und die geeignete Maßnahme (1-5) rechtzeitig ausgewählt und effizient ausgeführt werden.

1.1 Ziel und Aufbau der Arbeit

Das Ziel dieser Arbeit ist die Unterstützung der Wissenschaftler in der Nutzung verteilter Umgebungen. Die Verfahren sollen für die Verwaltung, genauer Schaffung, freier Speicherkapazität entwickelt werden und mit häufig genutzten Umgebungen/ Frameworks prototypisch implementiert werden. Diese Implementierungen sollen anhand ausgewählter funktionaler und nicht-funktionaler Merkmale vergleichend evaluiert werden. Um diese Ziele zu erreichen, wird in den folgenden Schritten vorgegangen:

Kapitel 2 Betrachtung ausgewählter Fallbeispiele und deren Generalisierung sowie die Darstellung zentraler Herausforderungen für eine leistungsfähige Infrastruktur

Kapitel 3 Analyse der Probleme in verteilter Umgebung und eine Ableitung von Anforderungen an Verfahren für die Gewinnung freier Speicherkapazität

Kapitel 4 Entwicklung von abstrakten Lösungsansätzen für die in Kapitel 3 aufgestellten Anforderungen,

Kapitel 5 Auswahl der geeigneten Technologie für die Implementierung von Lösungsansätzen.

Kapitel 6 Prototypische Implementierung für die Lösungsansätze mithilfe ausgewählter Technologie.

Kapitel 7 Demonstration von implementierten Verfahren und ihre Bewertung auf Performance.

Im Fazit wird die Arbeit mit einer Zusammenfassung abgeschlossen.

2 Wissenschaftliche Datenverarbeitung in verteilten Umgebungen

Dieses Kapitel stellt verteilte wissenschaftliche Umgebungen anhand von drei Fallbeispielen vor: CERN für die Hochenergiephysik (vgl. Kapitel 2.1), C3Grid für die Klimaforschung (vgl. Kapitel 2.2) und CLUES für die Astrophysik (vgl. Kapitel 2.3). Im Anschluss werden die wichtigsten Merkmale einer verteilten wissenschaftlichen Umgebung generalisiert (vgl. Kapitel 2.4). Das Kapitel endet mit einer kurzen Darstellung der wichtigsten Herausforderungen an eine leistungsfähige, verteilte wissenschaftliche Umgebung (vgl. Kapitel 2.5).

2.1 Fallbeispiel CERN

Das CERN[1], die Europäische Organisation für Kernforschung, ist eine Großforschungseinrichtung bei Meyrin im Kanton Genf in der Schweiz. Im CERN wird auf vielfältige Weise physikalische Grundlagenforschung betrieben, besonders bekannt ist der große Teilchenbeschleuniger. Das Worldwide LHC Computing Grid (**WLHG**) Projekt am CERN ist eine globale Zusammenarbeit von vielen Wissenschaftlern aus unterschiedlichen Instituten in der ganzen Welt, um die unterschiedlichen Analysen und Simulationen auszuführen. WLCG streckt sich über nationale Rechenzentren, Universitäten und Forschungsinstitute aus und versorgt die Computing Ressourcen für die Analyse, das Verarbeiten und das Speichern von großen Datenmenge, 25 Petabytes, die von den großen Large Hadron Collider (LHC) Experimenten (ALICE, ATLAS, CMS, LHCb,...) generiert werden.

Die Architektur der Computing-Infrastruktur für LHC wird in einem hierarchischen Multi-Tier Modell organisiert. Das Multi-Tier-Modell[2] besteht aus vier Schichten. Jede Schicht (Tier) hat spezifische Aufgaben, Ressourcen und Dienste (vgl. Abbildung 2.1) [3].

Tier 0 Hier ist der zentrale CERN Computer. Er besitzt große Infrastruktur-Ressource von Rechenkapazität, Storage Kapazität, Netwerkbandbreite usw. CERN ist verantwortlich für das Speichern der Rohdaten in einer erste Kopie, bewahrt diese Datenmenge und kopiert sie in die Schicht 1, um bessere Datenzugriffe und Fehlertoleranz zu ermöglichen.

Tier 1 Auf dieser Schicht sind viele Hochleistungsrechenzentren auf nationalem Level.

[1] http://public.web.cern.ch/public/
[2] http://lcg-archive.web.cern.ch/lcg-archive/public/tiers.htm
[3] http://wlcg.web.cern.ch

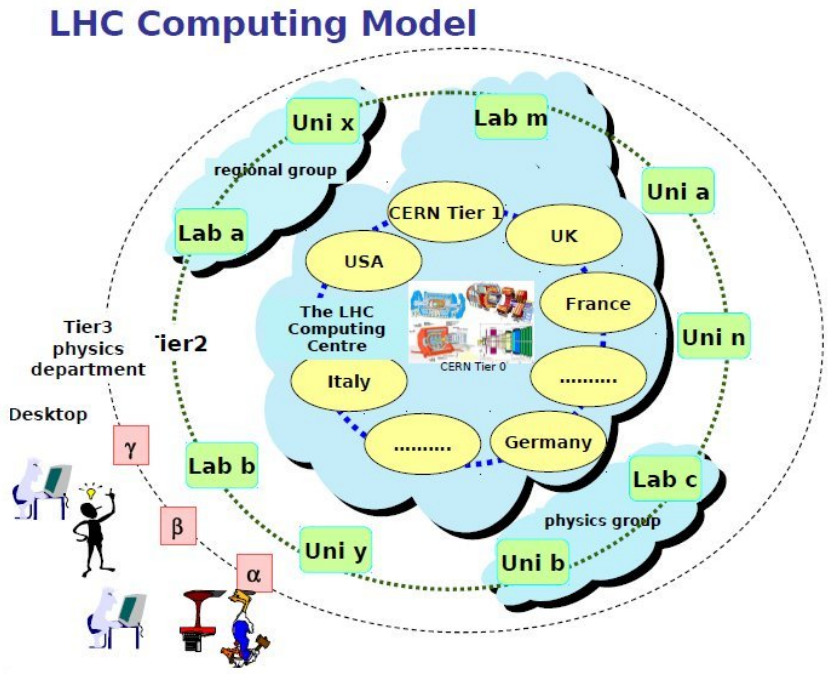

Abbildung 2.1: LHC Computing Model (Quelle: Deploying the LHC Computing Grid The LCG Project - Ian Bird IT Division, CERN CHEP 2003 27 March 2003)

Diese Rechenzentren verfügen über große Datenspeicher und haben hocheffiziente Verarbeitungskapazitäten. Die gesendeten Daten aus dem CERN werden analysiert, bearbeitet und archiviert. Tier 1 arbeitet als Data Center, verteilt die Daten und Unterstützung weiter an eine tiefere Schicht. Einige Standorte sind TRIUMF (in Kanada), KIT (in Deutschland), Fermilab-CMS (in den USA), usw.

Tier 2 sind die Universitäten, Forschungsinstitute und regionale Rechenzentren, die genügend IT Ressourcen und Infrastrukturen besitzen, welche die spezifischen Tasks verarbeiten. Es gibt momentan 140 Standorte auf der ganzen Welt.

Tier 3 sind die individuellen Wissenschaftler mit lokalen Computer Ressourcen (Institute Server), welche entweder zur Universität oder einer Arbeitsgruppe gehören.

Ein Grund für die Nutzung der hierarchischen Struktur von multi-computing Standorten ist das Speichern und Verarbeiten von Daten, inklusive der Kosten und der hohen Verfügbarkeit von Ressourcen. Ein anderer Grund ist die geographische Örtlichkeiten und Fehlertoleranz. Das Speichern von Daten in geographischen Standorten nah an dem Standort des Benutzers reduziert die Zeit der Information und Datenübertragung im Netzwerk. Die Fehlertoleranz aufgrund mehrerer Kopien der Daten erhöht die Sicherheit im Speichern und vermeidet Datenverlust, falls Katastrophen an einem Ort geschehen[4].

Das LHC Projekt arbeitet in eng Zusammenarbeit mit anderen Technologie-Anbietern und Großprojekten (zb. EGEE) zusammen um eine Lösung für die Datenanalyse von LHC Experimenten zu liefern. Die Struktur des LHC Projekte [4] fokussiert folgende Bereiche wie

[4] http://lcg-archive.web.cern.ch/lcg-archive/public/components.htm

Abbildung 2.2: WLCG Computing Grid Komponente
(Quell http://lcg-archive.web.cern.ch/lcg-archive/public/components.htm)

Networking, Hardware, Middleware und Serviceware(vgl. Abbildung 2.2). Diese werden im weiteren Verlauf wie folgt definiert:

Physics Software (Serviceware) ist eine Menge von objekt-orientierten Kern-Bibliotheken und Frameworks für die Datenanalyse. Sie wird in LHC Experimenten und in HEP-Gemeinschaften (ROOT) benutzt, und ist ein Daten Persistente Framework für Petabyte-skalierbare Datenspeichersysteme (POOL).

Middleware bezeichnet die Grid Middleware, die für Nutzer entworfen wurde, um auf die Grid-Ressourcen zu zugreifen. Die wichtigsten Grid Middlewares sind gLite, UNICORE, dCACHE und Globus Toolkit.

Fabric (Hardware) sind Kern-Ressourcen von Rechenzentren wie z.B. Storage System, HPC und unterschiedliche Hardware Geräte. Sie sind bedeutend für Forschungsarbeiten, die von jedem Beteiligten am WLCG Projekt-Zentren werden (Fabric).

Data Transfer (Networking) Der Datenaustausch zwischen WLCG-Zentren wird von einem Grid File Transfer Service verwaltet, der vom EGEE Projekt entwickelt wurde. Die Netzwerkgeräte ermöglichen die hohe Bandbreite für die Datenübertragung zwischen CERN und 11 hauptsächlichen Tier 1 Rechenzentren[14].

Für die Verwaltung dieser großer Datenmenge erstrebt man ein verteiltes Daten Management System, das in der Umgebung mit folgenden Eigenschaften eingerichtet wird:

- das Speichern von Daten wird in unterschiedlichen Standorten hoch verteilt, um eine **Fehlertoleranz** und hohe **Datenverfügbarkeit** zu gewährleisten,
- die Dateien sind sehr groß, oft von hunderte Megabyte bis einige Gigabyte,
- die Daten werden nach der Produzieren kaum geändert, neue Daten werden weiter hinzugefügt,

- es existieren viele unterschiedliche *'service-level agreement(SLA)'* und *'Quality of Service(QoS)'* von Computing Ressourcen, die von unterschiedlichen Rechenzentren und Universität (Tier1, Tier2) eingerichtet werden.

- Es gibt keine zentrale Administration über alle verfügbaren Ressourcen.

- das System erwartet eine hohe **Skalierbarkeit**, ermöglicht die Aufnahme neuer zusätzlicher Ressource.

Die Motivation dahinter ist die Anwendung eines verteilten Daten-Management-Systems, den sogenannte DQ2, (wird im ATLAS Experiment benutzt). Die grobe Architektur von DQ2 (vgl. Abbildung 2.3) unterstützt den Nutzer in unterschiedlichen Funktionalitäten (Datenmenge erzeugen, Datenmenge replizieren, Benachrichtigung, ganze Datenmenge oder einen Teil davon abfragen, usw.). Lokale Services übernehmen die Rolle des Speicherns eigener Informationen der erzeugten Datenmenge und des Verarbeitens der Dateien (verschieben zu anderen Storages, Löschen, Suchen). Globale Service funktionieren als *'Master'* für alle Aktivitäten zwischen Nutzer und den lokalen Services. Wenn der Nutzer eine Datenmenge abfragt, wird die Abfrage an den *'Master'* über *'dataset redirection service'* weitergeleitet. Der *'Master'* wird sie in eine Warteschlange schieben und dann die Arbeit für lokale Agent übergeben.

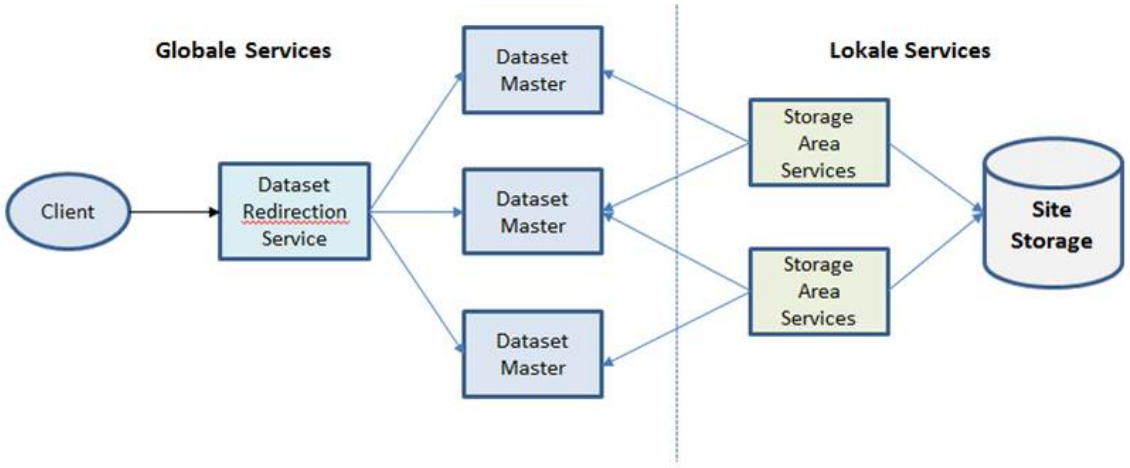

Abbildung 2.3: DQ2 Architektur

Der reale weltweite Auslastungsfaktor mit DQ2 bei ATLAS Das System verwaltet über 7 Petabytes Daten in über 60 *'distinct computing center'*, über 50 Million eindeutige Dateien, und insgesamt 80 Million Replikationen. Der große Skalierbarkeit-Test stellt das Versenden von Petabytes Daten an weltweite Forschungsinstitute und Rechenzentren mit einer Geschwindigkeit bis 1.5GByte/sec (7 Terabyte pro Stunden) in das Wide-Area-Network(WAN) dar[4].

2.2 Fallbeispiel C3Grid

In der Klimaforschung werden die verschiedenen, teilweise extrem umfangreichen und in unterschiedlichen Standards abgespeicherten Daten in wissenschaftlichen Anwendungen (Workflows) verarbeitet. Die Daten aus den Beobachtungen oder Modellexperimenten sind, bei kontinuierlich anwachsendem Datenvolumen, auf viele heterogene Datenarchive verteilt. Für den einzelnen Wissenschaftler wird es dabei immer schwieriger, sich einen Überblick über die verfügbaren Datensätze zu verschaffen.

Aufgrund der explosionsartig an wachsenden Klimadatenbestände ist eine passende Dateninfrastruktur erforderlich, um die Klimaforscher bei komplexen Datenauswertungsaufgaben zu unterstützen. Die Datensätze werden dezentral in den unterschiedlichen Institutionen erzeugt, haben oft eine riesige Größe und werden deswegen auch dort gehalten. Die Klimaforschern treffen oft auf Probleme wenn sie gespeicherte Datensätze in großen Datenmengen von Millionen von Dateien suchen möchten. Es fehlt eine **einheitliche Sicht** auf die komplexen und heterogenen Datenbestände und das Datenmanagement zwischen Datenstandorten.

An dieser Stelle setzt das Projekt C3-Grid (Collaborative Climate Community Data and Processing Grid)[5] an. Als ein Ziel des Projektes wird im C3-Grid eine Umgebung eingerichtet, die es dem Anwender ermöglicht, einen eigenen Workflow zu formulieren. Das C3-Grid stellt Dienste bereit, die den konkreten Datenzugriff, den Datentransport sowie das Ausführen wissenschaftlicher Workflows auf Daten unterstützt. Die C3Grid Infrastruktur stellt deshalb einen verteilten virtuellen Kollaborationsbereich dar, der (erstens) eine einheitliche Suche in angeschlossenen Datenarchiven erlaubt und (zweitens) einen einheitlichen Datenzugriff bereitstellt, sowie (drittens) eine verteilte kollaborative Datenverwaltung und Datenprozessierung unterstützt[22]. Die verteilte Datenhaltung befindet sich entweder in unterschiedlichen Data Centern (z.B. World Data Center, Universitäten usw.) oder ist global verteilt und wird jetzt in einheitlichen **Collaborative C3Grid Workspace** (vgl. Abbildung 2.4) organisiert. Viele standardisierte Schnittstellen (REST,HTTP,GridFTP) werden in C3Grid integriert, um den Datentransfer in die verteilten Umgebungen zu ermöglichen. Durch C3Grid werden Metadaten in einem zentralisierten Metadatenkatalog gesammelt und sind so über ein gemeinsames Suchinterface auffindbar. Daten werden mit zugehörigen Metadaten im C3Grid Workspace verwaltet und können extern zugreifbar gemacht werden[12].

[5] C3Grid

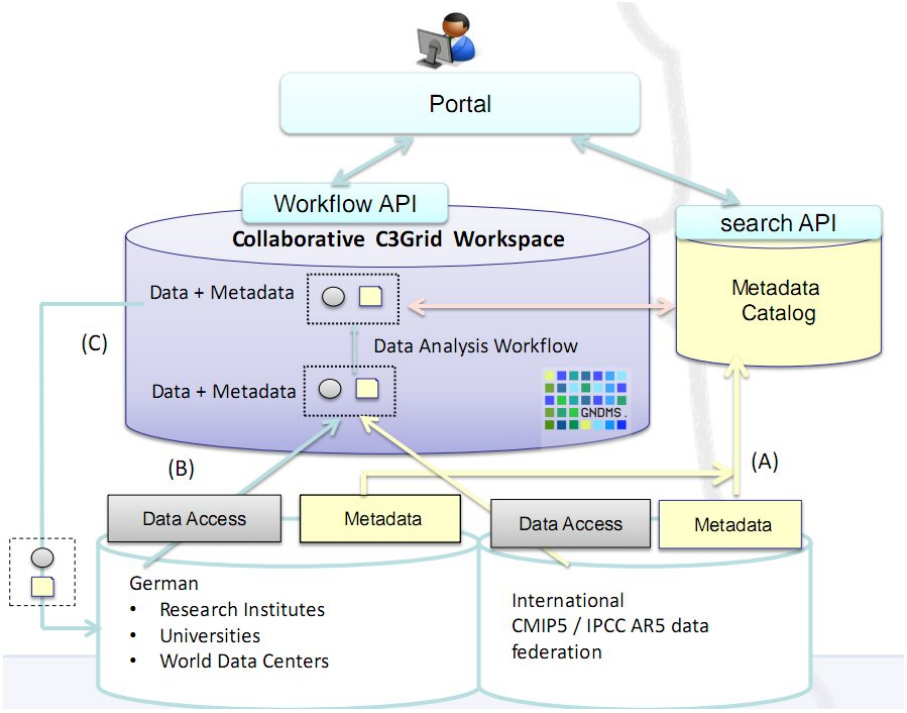

Abbildung 2.4: C3Grid Modell (Quell: [12])

2.3 Fallbeispiel CLUES

Das Projekt CLUES (Constrained Local Universe Simulation) (vgl. Abbildung 1.1) zeigt grob, wie Forschungsdaten über mehrere Schritte durchgearbeitet werden . Es besteht aus 3 wichtigen Phasen:

Phase 1 Vorbereitungsschritt für die Simulation (Schritte (2),(3),(4)). Die beobachteten Rohdaten werden für die initialisierte Bedingung vorbereitet. Die Vorsimulationen mit geringerer räumlicher Auflösung werden durchgeführt. In dieser Vorsimulationsphase ist die Auswahl geeigneter Startparameter ein entscheidender Faktor. Die Simulation mit bestem Ergebnis der Local Group wird für die nächste Simulation mit höherer Auflösung ausgewählt. Für diese Vorsimulationen können auch Cluster, remote HPC oder lokale Rechner verwendet werden.

Phase 2 Simulationsschritt (Schritte (6)). Da die Simulationen mit hoher Auflösung sehr großen Arbeitsspeicherbedarf brauchen, werden sie nur in wenigen Supercomputer in Europa (Leibniz Rechenzentrum Garching, Barcelona Supercomputing Center, Jülich Rechenzentrum) durchgeführt. Die Ergebnisse der Simulationen mit voller Auflösung werden zunächst auf den Datenspeichern der Supercomputer gelagert. Für CLUES werden je Simulation 120-150 Snapshots a 6,1 TB aufgezeichnet und in Rechenzentren gespeichert.

Phase 3 Nachbearbeitungsschritt (Schritt (8),(9),(10),(11)) Für die Nachbearbeitung (Visualisierung, Statistik,...) werden die Snapshots teilweise oder komplett zum Ort der Nachbearbeitung kopiert oder verschoben.

Die Daten werden über bekannte Protokolle wie SCP, FTP und GridFTP übertragen. Die zur Verfügung stehenden Ressourcen reichen für die Simulation und als Datenspeicher aus, jedoch kann es sein, dass die Engpässe während der Datenverarbeitung und Datenspeichern die Arbeit der Wissenschaftler behindern. Eines von vielen Problemen ist, dass man die Daten in unterschiedlichen Instituten speichert und der vorhandene Datenspeicher irgendwann voll sein wird. Die Datenspeicher werden von großen Simulationsdateien, sogenannten Snapshots, besetzt. Ein Snapshot hat 6TB (für eine Simulation) und enthält etwa 500 Dateien. Jede Datei ist etwa 12 GB groß. Insgesamt wurde die Speicherkapazität von etwa 120-150 Snapshots x 6TB = 720-900TB besetzt, mit 500 (Dateien) * 120-150 (Snapshots) = 60.000 bis 75.000 Dateien für alle Snapshots. Für die Verwaltung einer so großen Anzahl von Dateien sind **manuelle** Verfahren ungeeignet.

Im Laufe der Zeit wird die Speicherkapazität weiter von neuen Snapshots besetzt und ist schnell ausgeschöpft. Die Wissenschaftler müssen sich selbst daher den besetzte Speicherplatz freimachen, indem sie die Operationen von 1) bis 5) ausführen (vgl. Kapitel 1) um die neuen Daten weiter speichern zu können. Nicht nur durch die Verwaltung der Speicherkapazität wird die Verwaltung der gespeicherten Daten in CLUES Projekt weiter erschwert, dies geschieht auch, wenn mehrere Wissenschaftler mit unterschiedlichen Rechnerinfrastrukturen (Laptop, externe Festplatte, HPC, ...) auf gemeinsame Daten zugreifen müssen und die Daten auf verteiltes Rechner gespeichert werden. Manuelle Operationen sind hierzu zeitaufwändig, fehleranfällig und ineffizient [10].

2.4 Analyse der Fallbeispiele

Aus den drei oberen Beispielen hat sich die Verschiedenheit in Herangehensweise der komplexen verteilten Architekturen gezeigt: CERN mit Multi-Tier Schichten, C3Grid mit Metadaten-Catalog und verteilte Datenarchiven und CLUES mit einer Flach-Hierarchie und großen Snapshots. Im Weiteren wird die verteilte wissenschaftliche Umgebung generalisiert um die Abstraktion verteilter wissenschaftlicher Umgebungen zu verdeutlichen.

Die Petabyte-Datenmenge aus der wissenschaftlichen Umgebung ist oft sehr groß und wissenschaftliche Applikationen für die Analyse und Verwaltung dieser Datenmenge sind oft komplex. Deswegen unterstützt ein wissenschaftliches Rechenzentrum oft beide Dienste (Storage Systeme, Datenmanagement und wissenschaftliche Anwendungen). Der End-Nutzer benutzt Client-Anwendungen um auf die Ressource unter der Kommunikation einer Abfrage an laufende komplexe wissenschaftliche Applikation im Rechenzentrum zuzugreifen. Das Ergebnis wird an den User zurückgesendet, nachdem die Aufträge abgearbeitet wurden [11].

Im Wesentlichen besitzt eine verteilte wissenschaftliche Umgebung (z.B. die vernetzter Forschungsinstitute und Rechenzentren) jeweils eine lokale IT-Infrastruktur mit den folgenden Ressourcentypen:

(Super-)Computer parallele Rechner mit hoher Rechenleistung, um große Berechnungen, Simulationen und Visualisierung in der Forschung zu ermöglichen. Je nach Bedarf der Forschungsfragen können Wissenschaftler lokale oder entfernte Computer-

Ressourcen verwenden. Dazu senden sie Rechenaufträge (engl. job) an die Ressource.

Speicherressource sind große Datenserver oder verschiedene Speichersysteme (z.B. Bandarchive, schnelle Dateiserver, ...). Sie bieten eine große Speicherkapazität, um die verschiedenen Daten aus unterschiedlichen Quellen zu speichern. Ein Data Center könnte die Replikation der Daten aus anderen Rechenzentren unterstützen, um unvorhersehbare Datenverluste zu vermeiden.

Netzwerk Ressource sind Netzwerk-Infrastruktur und Netzwerk-Geräte. Sie ermöglichen den Datentransfer im lokale Netzwerk (LAN), zwischen Forschungsinstitute, Rechenzentren (WAN) oder den Datenaustausch und die Zusammenarbeit von Wissenschaftlern.

Diese Ressourcen (Computer, Speicher, Netzwerk) unterstützen Wissenschaftler in der Verarbeitung und Verwaltung von der **Forschungsdaten**, die Jahr für Jahr erforscht und im Storage System ständig erhoben, im Storage System ständig erfasst und archiviert werden und in unterschiedlichen Formaten, z.B. von Rohdaten (von Satelliten aufgenommen) bis hinzu verarbeiteten und analysierten Daten existieren. Die Daten und Forschungsergebnisse ermöglichen die Reproduzierbarkeit sowie die Weiterebearbeitung und Auswertungen in der Wissenschaft. Im Laufe der Zeit steigert sich diese Datenmenge rasant und die Verwaltung der großen Datenmenge wird in Zukunft eine der zentralen Herausforderungen in der Informationstechnologie sein.

Die folgende Abbildung (vgl. Abbildung 2.5) generalisiert eine verteilte Forschungsumgebung von kooperierenden Instituten und Multi-Nutzern. (S: Wissenschaftler; C: Computer Ressourcen; D: Daten)
Jedes Institut besitzt oft eigene Infrastruktur mit unterschiedlichen Ressourcen. Manche haben nur Datenspeicher-Ressourcen (Institute C), andere haben nur Computer-Ressourcen (z.B. Institute D) oder aber beide Ressourcen (z.B. Instituten A, B, E). Die Daten eines jeden Instituts befinden sich lokalen Systemen und werden von Computer-Ressourcen für die Berechnung und Simulation als Start-Parameter eingegeben oder von Mitarbeitern im selben Institut und von externen Mitarbeitern aus anderen Instituten mit Befugnis abgerufen. Wenn beide Institute kooperativ zusammenarbeiten, können Wissenschaftler von beiden Instituten Daten miteinander austauschen, bzw. die Mitarbeiter können möglicherweise auch die entfernten Ressourcen (Daten, Speicher, Computer, Software Service...) von anderen Instituten per Remote-Methode benutzen (z.B. Wissenschaftler S1 in Institute A, B und E). Wenn der Wissenschaftler die Daten aus zwei unterschiedlichen Rechenzentren miteinander korrelieren möchte, werden die Storage Systeme von beiden Rechenzentren in diesem Fall zu einem Bund zusammengeschlossen. Die gespeicherten Daten könnten zwischen Instituten als Backup oder Replikation (z.B. D3 und D4) kopiert werden. Seltener aber auch möglich ist der Fall von Institut D nur mit Computer-Ressource (C4). Aus unterschiedlichen Gründen (z.B. knappe Datenspeicher-Ressourcen, fehlende Ressourcen, Bedarf vom Nutzer) werden die Computer-Ressourcen die Daten aus der externen Speicher-Ressource (C4 und D5) aufrufen.

Erschwerend für eine automatisierte Verwaltung kommt hinzu, dass bisher keine IT-gestützte 'Buchhaltung' für die Speicherorte von Dateien (Snapshots und Ergebnissen der Nachbearbeitung) verwendet wird.

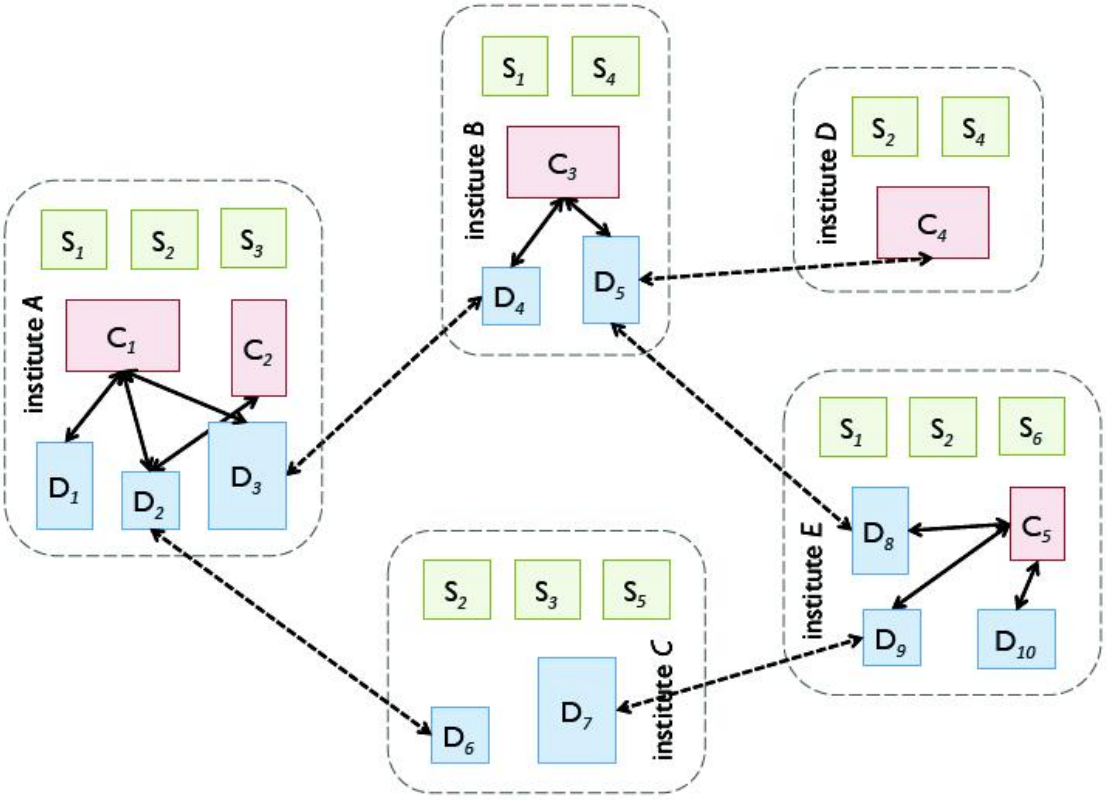

Abbildung 2.5: Entwurf von verteilten Arbeitsumgebung

Der Kapazitätsmangel verursacht oft viele Hindernisse und ist eine die Barriere in der wissenschaftlichen Datenverarbeitung. **Probleme**:

1. Erstellung eine Kopie einer großen Datenmenge. Die Backup-Arbeit/Replikation von Daten wird regelmäßig ausgeführt um die Daten sicher zu bewahren. Jedoch beansprucht dieser Vorgang sehr viel Speicherkapazität.

2. Der Speicherraum ist der gemeinsame Zugriffspunkt von vielen Wissenschaftlern aus unterschiedlichen Standorten. Die Folge ist, dass eine Speicherressource schnell erschöpft sein wird.

3. Die Vielfalt von Daten mit unterschiedlichen Erstellungsdaten, Formaten und Beschreibungen ist im Laufe der Zeit massiv geworden und beansprucht eine große Speicherkapazität im Speichersystem. Insbesondere, wenn viele Daten einmal erzeugt, aber nie gelöscht häufig gelesen werden.

4. Viele neue Datenformate kommen hinzu und erzeugt neue Datenmenge, Deshalb wird weiter zusätzliche Speicherkapazität gebraucht.

5. Die unterschiedliche Speicherbedarf von Nutzern in der Zukunft.

Die Folge des Kapazitätsmangels sind oft Datenverlust oder Unterbrechungen der Berechnungsergebnisse der Simulation . Der Wissenschaftler muss selber viel Zeit auf Ne-

bentätigkeiten (Manuelle Schaffung freier Speicherkapazität) investieren, statt sich auf seine spezielle Arbeit konzentrieren zu können.

2.5 Herausforderungen

Zusammengefasst aus der oberen Beispielen ergeben sich folgende Herausforderungen[21], die in dem Bereich der Verarbeitung und Verwaltung verteilter wissenschaftlicher Daten noch auftreten werden, wenn die Datenmenge in Zukunft weiter stetig zunimmt:

- Wie kann man die wissenschaftliche Daten archivieren? wenn die Verschiedenheit der Medien, also Geräte von Kassetten, Festplatten, bis über mobile USB Geräte und den Online Speicher Cloud, und ihre Formate sich stets weiterentwickeln.

- Wie kann man die Integrität der Daten garantieren? Sollen alle Forscher die Zugriffsrechte auf Daten Repositorie haben oder soll man die Zugriffe auf die Daten mit unterschiedlichen Rechten erlauben?

- Wie kann man den Kontext und die Herkunft der Daten vermitteln?

- Wie kann man die Privatsphäre bzgl. verlinkter Daten schützen?

- Wie kann man die Daten in Sammlung für die Reproduzierbarkeit und mit welcher Disziplin für die Interoperabilität organisieren?

- Wie kann man die Effizienz und Geschwindigkeit in der Datenverarbeitung verbessern wenn die Nachfrage nach Tools und Computer-Ressourcen, um die wissenschaftliche Datenanalyse zu verrichten, steigt, sogar schneller als das Datenvolumen selbst?.

Diese Herausforderungen im Management der Datenspeicherkapazität bestehen sowohl an den Höchstleistungsrechenzentren (betrifft im Wesentlichen die Snapshots der Simulationen) als auch an den Forschungsinstituten (Speicherung der Snapshots sowie der Ergebnisse der Nachbearbeitung).

3 Lösungsansätze und Anforderungen

Die Fallbeispiele CERN, C3Grid und CLUES in Kapitel 2 verdeutlichen, dass die Verwaltung verteilter Ressourcen in wissenschaftlichen Umgebungen viele Aspekte berücksichtigen muss(vgl. Herausforderungen in Kapitel 2.5). Neben dem Schwerpunkt dieser Arbeit – die Verwaltung der verteilten Speicherkapazität in wissenschaftlichen Umgebungen – sind die vorhandene Rechenkapazität (z.B. bei Nutzung von Datenkompression), die verfügbare Netzwerkbandbreite und die Netzwerklatenz (z.B. bei Zugriff auf entfernt gespeicherte Daten) weitere Aspekte, die aber in dieser Arbeit nur am Rande betrachtet werden. Um den Kern dieser Arbeit abzugrenzen, beschränken wir uns im Bereich Speicherverwaltung auf Situationen, in denen die Speicherkapazität an einem Ort (nahezu) ausgeschöpft ist. Momentan haben Wissenschaftler nur die Möglichkeit, selbstständig und ohne Unterstützung automatisierter Verfahren ausreichend freie Speicherkapazität zu schaffen. Zunächst schauen wir uns mögliche Vorgehensweisen der Wissenschaftler an(vgl. Kapitel 3.1). Anschließend skizzieren wir fünf Szenarien, in denen unterschiedliche Lösungsansätze verfolgt werden können (vgl. Kapitel 3.1). Schließlich beschreiben wir wichtige Kriterien, die eine Verwaltung verteilter Speicherkapazität in wissenschaftlichen Umgebungen erfüllen muss (vgl. Kapitel 3.2).

3.1 Lösungsansätze: Ad-hoc vs Systematische Arbeitsweise

Die Ad-hoc Arbeitsweise ist oft die schnelle und einfache Lösung und wird von Benutzer manuell durchgeführt. Die verwendete Basis-Ansätze erfordern wenige Informationen zur Datenverarbeitung bei den Nutzern, daher benötigt der Nutzer wenig Zeit, um eine Entscheidung zu treffen. Der Vorteil über diese Arbeitsweise läge im Kontext Lokal, wenn die Nutzer einen guten Überblick über ihre Datenspeicher hätten. Die Nachteile treten aber bei der Effizienz, besonders in einer hohen Daten-intensiv verteilten Umgebung auf, wenn Nutzer nur grobe Informationen über die Datenverarbeitung und die Datenhaltung im System besitzt. Eine falsche Entscheidung könnte hierbei zu einem permanenten Datenverlust führen.

Die Systematische Arbeitsweise bietet detailliertere Ansätze. Sind neben der Größe der verarbeiteten Daten auch andere Aspekte, wie der Inhalt der Daten (System-Metadaten), und dessen Einfluss auf die Leistungseffizienz, entscheidend, bietet die systematische Arbeitsweise mehr Möglichkeiten. Diese Arbeitsweise behebt die Nachteile der Basis-Ansätze, da die Daten der Nutzer ordentlich und sorgfältig bearbeitet werden; sowohl in einer lokalen als auch verteilten Ressource-Umgebung. Die systematische Arbeitsweise verbessert somit die Leistung der Ad-hoc Arbeitsweise. Die Nachteile sind jedoch dass der Nutzer mehrere Informationen und die Kenntnisse über die

Beschreibungen der Daten, den Speicherraum und das Daten Management System, wo die Daten gelagert sind, braucht.

Die folgenden Szenarien werden Schritt für Schritt die unterschiedlichen möglichen Situationen vorstellen und verdeutlichen, wie die Wissenschaftler mit ihrer Speicherkapazität umgehen könnten.

Szenario S1 Einfache Situation Man benötigt dringend freie Speicherkapazitäten innerhalb eines Standortes, in welchem der seine neuen Daten speichern möchte. Deshalb verwendet der Nutzer die Ad-hoc Arbeitsweise, um seine Entscheidung schnell treffen zu können. Die notwendigen Basis-Informationen sind die Pfade und der Standort der gespeicherten Daten, auf welchen die Operation darauf anwenden kann.

Szenario S2 Datenverarbeitung mit Bedingung Um das Risiko (z.B. Datenverlust) von Szenario 1 zu verringern, möchte der Benutzer eine Bedingung für die Ausführung der angewendeten Operation festlegen. Die Ausführung der Operation wird abbrechen, wenn die voreingestellte Bedingung nicht erfüllt werden kann. Diese Bedingung verzögert die Ausführung der Operation, die zu einer unerwünschter Situation führt. Ein Beispiel ist dass die Daten verarbeitet werden, wenn mindestens eine Replikation in Ressource XYZ existiert. Die nötigen Informationen sind die Pfade, der Standort der gespeicherten Daten und zusätzliche Informationen über die eingestellte Bedingung.

Szenario S3 Datenverarbeitung mit eingeschränkter Speicheranforderung Der Benutzer sieht voraus, wie viel neuer freier Speicherplatz geschafft werden soll. Die frei geschaffte Speicherkapazität darf nicht größer als benötigt sein. Die festgelegte eingegebene Speicheranforderung wird die Menge an frei gewonnenen Speicherplatz ausreichend halten. Die nötigen Informationen sind die Pfade, der Standort der gespeicherte Daten, und bestimmte Speicheranforderung.

Szenario S4 Multi-Speicher Ressourcen In dem Kontext einer verteilten Umgebung, in welcher man oft die freie Speicherkapazität in einem oder mehreren Speicherressourcen benötigt. Jede Speicherressource ist gegeneinander entfernt und hat unterschiedliche freie Speicherkapazitäten. Die Lösung wäre die Ausnutzung und Sortierung sowie Kombination freier Speicherräume aus unterschiedlichen Quellen (z.B. die Daten werden von einer Speicherressource zu einer anderen Ressource verschoben), bis eine Ressource, in welcher man die neue Daten speichern möchte, genügend Speicherplatz hätte. Die Informationen über die verfügbare Speicherkapazität, die Adresse sowie den erlaubten Operationen und Zugriffsrechten in allen Ressourcen, werden in diesem Fall benötigt.

Szenario S5 Kombination vieler Ansätze Bei der automatische Kombination von allen Ansätzen (z.B. Komprimieren, dann Löschen und schließlich Verschieben) sowie mehrerer Speicherressourcen könnten bessere Ergebnisse erwartet werden.

Diskussion über Szenarien

Die Zugriffsrechte auf Speicherressourcen sind in vielen Fallen unterschiedlich bei dem Administrator und den Nutzer. Der Administrator hat uneingeschränkten Zugriff auf alle Speicherressourcen und ändert die Speicherkapazität für jeden Benutzer, um die Überlastung des Systems zu vermeiden. Der Benutzer kann sich aber auch persönlich an den Administrator wenden, um den eigenen Speicherraum zu ändern, was jedoch in der Tat umständlich ist. Die Tabelle (vgl. 3.1) stellt die vorgeschlagenen Lösungsansätze gemäß der obigen Merkmale gegenüber.

In **Lokaler Umgebung**, bzw. wenn man nur eine Speicherressource hat, muss man freie Speicherplätze in diesem Ressource gewinnen. Die Operationen *Komprimieren, Löschen* sind in diesem Fall geeignet (vgl. die Szenarien S1, S2, S3). Szenario S1 tritt als allgemeine Situation auf, wenn man mit weniger Vorkenntnissen in kurzer Zeit eine schnelle Entscheidung auf die gespeicherte Daten mit allen Drei Operationen treffen will. Das Resultat mit dem Vorteil über die geschaffene Speicherkapazität liegt mehr beim *Löschen* als beim *Komprimieren* aber das Risiko für einen permanenten Datenverlust steigt an. In Szenario S2 kann man auch das Löschen, jedoch unter bestimmten Bedingungen anwenden, um das Risiko von Szenario S1 zu vermindern. In Szenario S3 kann man die Operation *Komprimierung* einsetzen. Um eine unnötige Rechenkapazität zu verhindern, weil das Komprimieren gewöhnlich mehrere Rechenaktivitäten von CPU verbraucht, braucht man in diesem Fall eine bestimmte Speicheranforderung.

In **verteilter Umgebung** mit verfügbaren Multi-Ressourcen (vgl. Szenario S4) ist das Verschieben von Daten geeignet. Die Suche nach freier Speicherkapazität in einem Standort kann durch das Verschieben von Daten von einem Standort zu einem anderen Standorten angeordnet werden. Die Laufzeit bzw. Wartezeit liegt hierbei nicht an der Rechenaktivität, wie beispielsweise dem Komprimieren, sondern an der Netzwerkleistung bzw. Übertragungsgeschwindigkeit. Außerdem müssen einige Aspekte wie die verfügbare Speicherressource, die Zugriffsrechte, die Verfügbarkeit von Servern in anderen Standorten und die installierten Übertragungsprotokolle (z.B. SCP, GridFTP, SFTP), berücksichtigt werden.

Das Szenario S5 zeigt die Verschiedenheit in der Datenverarbeitung, indem man die Szenarien S1 bis S4 miteinander kombiniert, um ein optimales Ergebnis zu erreichen. Bei dieser Kombination wird viel Zeit für die Analyse der Vorteile und Nachteile benötigt, entsprechend der unterschiedlichen Kontexte und der Beachtung aller Aspekte der Drei Operationen *Komprimieren, Löschen, Verschieben* (z.B. CPU-Rechenaktivität, Übertragungsbandbreite, aufwändige Implementierung usw.).

3.2 Wichtige Kriterien

In diesem Abschnitt werden aus den Herausforderungen (vgl. Kapitel 2) und oberen Szenarien, die ausgewählten Kriterien für die Verwaltung verteilter Speicherkapazitäten abgeleitet. Unter dem Kontext, dass der Wissenschaftler den freie Speicherplatz braucht, um die

Operationen	**Wissenschaftler**	**Administrator**
Daten komprimieren	✓	
Daten löschen	✓	
Daten verschieben	✓	
Quota erhöhen		✓
Andere Ressourcen auswählen	✓	

Tabelle 3.1: Angewendete Operationen

neue wissenschaftliche Daten aufnehmen zu können. Angenommen, dass die vorhandenen Speicherräume von Nutzern nicht selbst erhöht werden können.

K1 Die wissenschaftlichen Daten sind die Quellen von Ausschöpfung der Speicherressource. Um freien Speicherplatz zu schaffen muss man mittels unterschiedlicher Methoden die Anzahl bereits gespeicherter Daten, möglichst ohne Datenverlust reduzieren. Die Hilfsmittel für die Ermittlung wissenschaftlicher Daten sind das Daten Management System (DMS)(z.B. Dateisystem, Datenbank), die Metadaten (die systematischen Parameter, die die gespeicherten Daten in DMS beschreiben) und die Quota (Speicherzustand- die bereits benutzte Speicherkapazität/ Limitation der Speicherkapazität).

K2 Es existiert noch freier Speicherplatz in vielen Orten, jedoch reicht dieser nicht für das Speichern neuer Daten aus. Die Aufgabe ist, dass man diese freien Speicherplätze aus unterschiedlichen Speicherressourcen ausnutzen kann, indem die bereits gespeicherte Datenmenge ohne Verlust verändert, reorganisiert bzw. umgeräumt werden muss.

K3 Man benötigt einen einfachen Lösungsansatz mit möglichst wenig erforderlicher Information für die schnelle Schaffung freier Speicherkapazität.

K4 Man benötigt einen systematischen Lösungsansatz für den ausreichenden Gewinn von freiem Speicherplatz mit effizienter Leistung und verbessert zugleich die Nachteile der Ad-hoc Arbeitsweise. Bei der effizienten Datenverarbeitung geht es um den Bedarf an Betriebsmitteln (Anzahl der bearbeiteten Daten, Laufzeiteffizienz, Speichereffizienz). Im Allgemeinen soll bei den Verfahren so gut wie möglich kein unnötiger Verbrauch an Betriebsmitteln auftreten.

4 Die Verfahren für die Schaffung freier Speicherkapazität

Die Problemanalyse aus den Fallbeispielen (vgl. Kapitel 3) und die relevanten Kriterien (vgl. Kapitel 3.2) verdeutlichen die vorhandene Probleme und konkretisieren die Ziele für die Verwaltung verteilter Speicherkapazitäten. Dafür werden die entsprechenden Verfahren in diesem Kapitel vom Basis- Ansatz bis zum komplexen Ansatz entwickelt. Zuerst schauen wir uns die Verfahren mit der Operation *Komprimieren* (vgl. Kapitel 4.1), dann fahren wir fort mit der Operation *Löschen* (vgl.Kapitel 4.2) und gelangen schließlich zur letzten Operation *Verschieben* (vgl.Kapitel 4.3).

4.1 Komprimierung von Daten

Das Verfahren der Komprimierung von Daten wird oft in lokalen Datenspeichern angewendet, wenn:

- die Erweiterung um neuen Ressourcen durch die Aufnahme der Verbindung mit dem Administrator, um die Quota zu erhöhen in diesem Fall ausgeschlossen ist. Der Grund dafür besteht darin, dass der Administrator hier nicht immer zuständig ist.

- die Lösung der Speicherverteilung mithilfe der anderen Ressourcen auch in diesem Fall ausgeschlossen wird, weil der Nutzer nur einen Speicherraum hat oder die Verbindung zu anderen entfernten Ressource unterbrochen ist. Deshalb ist die Datenübertragung nicht möglich. Daher wird die Schaffung nach freier Speicherkapazität über das Verschieben von Daten zur anderen Ressource nicht realisierbar sein.

- für die gespeicherten Daten keine Duplikaten existieren. Deswegen ist das Löschen von Daten unerwünscht wenn die Daten zukünftig noch gebrauchen werden könnten.

Die weiteren Absätze beschreiben wie die Strategie der Datenkompression in unterschiedlichen Bedingungen angewendet wird. Nach jedem angewendeten Ansatz wird die Diskussion über den Ansatz eingeführt.

4.1.1 Basis-Ansatz

In dem Basis Szenario verwendet der Nutzer die Komprimierungsmaßnahme auf **allen** Daten beginnend beim eingegebenen Pfad als Start-Verzeichnisstruktur. Die Abbruchbedingung ist erfüllt, wenn alle Dateien bereits abgefragt und bearbeitet wurden. Der Unterschied zwischen der Größe der Datei vor und nach der Komprimierung ist die Gewinnung der Speicherkapazität. Die bereits komprimierten Daten und ihre Verzeichnisstruktur müssen in dem Fall verlustfrei in originale Daten wiederhergestellt/dekomprimiert werden, wenn der Benutzer sie wieder verwenden möchte.

Idee

Man traversiert die Verzeichnisstruktur und bearbeitet alle darin gefundene Dateien drin. Angenommen wird, dass die Verzeichnisstruktur einer Baumstruktur gleicht. Die eingegebene Adresse Start-Pfade entspricht dem Wurzelknoten des Baums. Jeder Knoten kann entweder weitere Kindknoten als ein Verzeichnis mit/ohne weiteren Kind-Knoten enthalten oder ein Blatt in Form einer physikalische Datei sein.

Pseudocode 1

Algorithmus 1 Komprimieren aller Dateien in einem Verzeichnis und allen {Unter}*Verzeichnissen

Eingabe: Die wählbaren Pfade aus der Verzeichnisstruktur des Nutzers.
Resultat: Alle Daten werden in ihrer Verzeichnisstruktur werden komprimiert.

```
1:  procedure KomprimiereAlleDateien(root)
2:      if Verzeichnis (root) leer ist then
3:          return;                          ▷ Abbruch, Ausgang von der Funktion.
4:      else
5:          for all e ∈ root do              ▷ Iteriert über alle Elemente im Verzeichnis
6:              if Element (e) Verzeichnis ist then
7:                  KomprimiereAlleDateien(e)          ▷ Rekursiver Aufruf
8:              else
9:                  Komprimiere(e)                     ▷ Komprimiert die Datei e
10:             end if
11:         end for
12:     end if
13: end procedure
```

Der Algorithmus läuft durch alle Elemente der Verzeichnisstruktur (vgl. Zeile 5) und überprüft die Bedingung, ob das Element ein Verzeichnis ist (vgl. Zeile 6). Ist dies der Fall, wird dieses Verzeichnis per rekursiven Aufruf mit neuer Adresse weiter nach Daten überprüft (vgl. Zeile 7). Ist dies nicht der Fall, ist es ein Element der Daten und wird mit der externen Komprimierungsfunktion gebunden, um die Datei komprimieren

zu können (vgl. Zeile 9). Die rekursive Methode wird genutzt, um den Ablauf auf der Baumstruktur zu verdeutlichen. Der Algorithmus terminiert, nachdem alle Elemente in der Verzeichnisstruktur überprüft wurden und die eingegebene Verzeichnisstruktur leer ist (vgl. Zeile 2).

Diskussion

Der Basis-Ansatz fordert an der Eingabe wenige Informationen über die gespeicherten Daten, aus dem Speicherraum des Nutzers. Das extern aufgerufene Programm für die Komprimierung kann frei vom Nutzer ausgewählt werden. Dies spielt in dieser Arbeit jedoch nur die Nebenrolle. Der Basis-Ansatz ist jedoch kein effizienter Ansatz, weil der Algorithmus auf **alle Daten** angewendet wird. Viele Daten mit kleiner Größe in der Verzeichnisstruktur werden unnötig, ohne Auswahl bearbeitet. Wenn eine große Datenmenge unnötig komprimiert ist, verursacht dies weiter die Verschwendung von Rechenkapazität, die mit der externen Komprimierungsprogramm zusammenhängt (z.B. CPU, Festplatte, Speicher usw.). Es reduziert daher die gesamte Leistung des ganzem Systems. Außerdem könnten diese Daten bedeutend für den Nutzer sein und der Nutzer muss sie dekomprimieren, falls sie später wieder gebraucht werden.

Die weiteren Ansätze werden die Effizienz des Basis-Ansatz kontinuierlich mit zusätzlicher Eigenschaften verbessern.

4.1.2 1.Verbesserung Ansatz

Idee

Es wäre besser, wenn die Anzahl der Daten nach dem eigenen Bedarf des Nutzer komprimiert werden. Dieser Ansatz erbt alle Eigenschaften vom vorherigen Ansatz und verbessert ihn weiter, indem die Speicheranforderung vom Nutzer als Vorbedingung eingegeben werden muss. Die eingegebene Speicheranforderung grenzt die Anzahl der bearbeiteten Daten ein. Das bedeutet, dass die Daten komprimiert werden, bis die geschaffene Speicherkapazität die Speicheranforderung erfüllt.

Pseudocode 2

Der Algorithmus funktioniert ähnlich wie der Algorithmus vom Basis-Ansatz. Was ergänzt wird ist die Speicheranforderung. Die Operation Komprimierung wird auf der Datei nur ausgeführt wenn die gewonnene Speicherkapazität (Variable sum) die Speicheranforderung (Variable X) noch nicht erreicht hat (vgl. Zeile 6). Die Operation errechnet die gewonnene Kapazität, wenn die Größe der Datei nach der Komprimierung geändert wurde. Die Größe der Datei vor und nach der Komprimierung muss unterschiedlich sein und der Unterschied über die Größe summiert sich steigend (vgl. Zeile 14). Die Funktion gibt das Ergebnis als geschaffte Speicherkapazität (vgl. Zeile **??**) zurück. Der Algorithmus terminiert (vgl. 4),

Algorithmus 2 Komprimieren aller Dateien unter Berücksichtigung des Speicherbedarfs

Eingabe: root ist der Startpfad (Verzeichnis), X ist der zu schaffen Speicherplatz

Resultat: Der genügende Speicherplatz wurde geschaffen oder alle Dateien in allen {Unter}*Verzeichnissen werden komprimiert.

```
1:  sum = 0                                              ▷ Globale Hilfsvariable
2:  procedure KomprimiereBisSpeicherplatz(root,X)
3:      if Verzeichnis (root) leer ist  then
4:          return;
5:      end if
6:      if sum < X then                                  ▷ es fehlt noch Speicherplatz
7:          for all e ∈ root do
8:              if Element (e) Verzeichnis ist then
9:                  KomprimiereBisSpeicherplatz(e,X)     ▷ Rekursiver Aufruf
10:             else
11:                 e_o = ermittelt die Dateigrösse (e)
12:                 Komprimiere(e)                       ▷ Komprimiert die Datei
13:                 e_z = ermittelt die Dateigrösse (e)
14:                 sum = sum + (e_o - e_z)              ▷ Berechne den Gewinn des freie
    Speicherkapazität
15:             end if
16:         end for
17:     end if
18: end procedure
```

nachdem alle Dateien in allen {Unter}*Verzeichnissen per rekursivem Aufruf gefunden wurden (vgl. Zeile 7) oder die Speicheranforderung schon erfüllt wurde.

Diskussion

Dieser Ansatz verbessert die Effizienz des Basis-Ansatzes. Der Algorithmus komprimiert die Daten, bis die Speicherkapazität erreicht ist. In diesem Fall kann man sehen, dass nicht alle Daten wie beim Basis-Ansatz bearbeitet werden konnten. Daher spart man die unnötige Verarbeitung und reicht genügend freie Speicherkapazität für den Bedarf. Der Nachteil liegt in der Auswahl der Daten für die Komprimierung. Zum Beispiel lohnen sich die Daten mit kleiner Größe nicht, da sie wenig bis gar keine freie Speicherkapazität bieten. Ziel der Kriterien ist das Reduzieren von unnötigem Verbrauch bzw. die Anzahl der bearbeiteten Daten.

4.1.3 2.Verbesserung Ansatz

Idee

Um die Effizienz des Ansatzes weiter zu verbessern bzw. die Daten mit kleiner Größe heraus-zu-filtern wird die Sortierfunktion verwendet. Die Idee ist, dass die Daten mit höchster Größe zuerst bearbeitet werden. Dafür wird neben der Pfade der Datei noch die Information über die Größe der Daten ermittelt. Der Entwurf von Algorithmus wird dem oberen Algorithmus 2 um die Sortierfunktion ergänzen, um alle Dateien in Verzeichnisstrukturen nach ihrer Größe zu sortieren, bevor sie verarbeitet werden.

Pseudocode 3

Algorithmus 3 Komprimieren aller Dateien unter Berücksichtigung ihrer Größe

```
1:
2: function KomprimiereMitSortieren(root)
3:     sortlist = sort(root,size)          ▷ Sortiert alle Elemente nach ihrer Größe
4:     for all e ∈ sortlist do
5:         e_o = ermittelt die Dateigrösse (e)
6:         Komprimiere(e)
7:         e_z = ermittelt die Dateigrösse (e)
8:         sum = sum + (e_o - e_z)
9:     end for
10:    return(sum)
11: end function
```

Die Funktion *sort(root,size)* enthält 2 Parametern *root* als der Startpfad des Verzeichnisses, und *size* als eine erkennbare System-Eigenschaft von der Datei (vgl. Zeile 3). Die Dateien

werden in diesem Fall nach der Größe sortiert werden, weil die Daten außer der Größe noch andere Eigenschaften besitzen. Das Ergebnis von der Vorsortierung ist eine Liste aller gefundener Dateien, die bereits nach der Größe absteigend sortiert wurden. Diese Liste der Daten wird weiter in die Schleife (vgl. Zeile 4) angeordnet und die einzelne Datei wird nacheinander verarbeitet. Wie die Sortierfunktion funktioniert, ist in diesem Fall nicht relevant.

Diskussion

Die Sortierfunktion in diesem Ansatz hat die Daten mit kleiner Größer, die wenig bedeutend für die Komprimierung sind, nach hinten verschoben. Die Sortierung der Daten vor der Verarbeitung könnte die Anzahl der bearbeiteten Daten weiter reduzieren. Damit wird die Effizienz des Verfahrens weiter erhöht. Weiter zu bemerken ist, dass die Sortierfunktion die Rechenkapazität lang besetzen könnte. Je mehr die Daten es gibt, desto mehr Zeit braucht die Sortierung. Außerdem kann, wenn die Daten von Dateisystem verwaltet werden, die Sortierung große temporäre Speicher nutzen, um das Zwischenergebnis während der Sortierung zu speichern und daher Einfluss auf die gesamte Leistung des Arbeitssystems haben.

4.2 Löschen von Daten

Die nächste Beobachtung ist die Operation *Löschen*. Der Unterschied zu der Operation Komprimieren ist, dass die ausgeführte Operation Löschen nicht umkehrbar ist. Die gelöschter Daten werden permanent vom System entfernt. Dafür wird der Ansatz Löschen mit einer zusätzlichen Bedingung verbessert.

4.2.1 Löschen, falls die Replikation von Daten existiert

Idee

Hierbei wird noch eine zusätzliche Eigenschaft der Daten beobachtet, nämlich die Replikation der Daten. Ein möglicher Ansatz wäre, dass die Daten nur gelöscht werden, falls eine Replikation (Kopie) von ihnen woanders existiert. Die Replikation besteht aus mindestens einer Kopie der Daten, und kann sich entweder im Lokal des gleichen Speicherraums oder in anderen entfernten Speicherräumen lagern. Eine Kopie der Daten enthält denselben Inhalt der Daten, und besitzt die gleichen Eigenschaft wie die originalen Daten. Die Überprüfung der Replikation der Daten kann über ein Daten Management System durchgeführt werden, welches die Replikation verwaltet. Der folgende Algorithmus gibt dem Nutzer eine sichere Entscheidungshilfe, nach welcher erst die Operation *Löschen* auf die Daten angewendet wird.

Pseudocode 4

Algorithmus 4 Löschen die Daten unter Berücksichtigung ihrer Replikation

Eingabe: root ist der Startpfad (Verzeichnis)
Resultat: alle Dateien mit Replikation werden gelöscht.

```
1: function REPLICATION(root)
2:     replist = Suchen in (root)▷ gibt alle Dateien mit mindestens einer Kopie zurück
3:     return replist                ▷ die Liste aller Dateien mit mindestens einer Kopie
4: end function
5:
6: procedure LOESCHEN(replist)
7:     if Liste (replist) nicht leer ist then
8:         for all e ∈ replist do
9:             DELETE(e)
10:        end for
11:    end if
12: end procedure
```

Die Funktion Replication(root) wird alle Dateien ausgehend von Start-Pfad root überprüfen, ob sie ein Replikation besitzen (vgl. Zeile 2). Das Ergebnis der Funktion *Replication* ist eine Liste aller Dateien, die mindestens eine Kopie besitzen. Es kann sein, dass keine Datei eine Kopie hat. In diesem Fall ist die Liste leer. Die Funktion *Loeschen* (vgl. Zeile 6) wird die zugeordnete Liste bearbeiten, bzw. alle Dateien löschen, nachdem die Existenz der Replikation von Daten anerkannt wurden (vgl.Zeile 9).

Diskussion

Die Operation Löschen von Daten bringt bessere Leistung für die freie Speicherkapazität als Komprimieren, weil die ganzen Daten vom System entfernt werden. Damit werden alle Speicherplätze, die von Daten besetzt werden, sofort freigegeben. Die Verwaltung der Replikationen der Daten kann dem Replikation Manager erleichtert werden. Replikation Manager legt vorher eine Kopie (Replikation) der Daten an. Die Abfrage nach Existenz der Replikation von Daten kann sich über Replikation Manager entschieden werden. Außerdem kann man die Operation *Löschen* auch mit obiger Ansätzen (Speicheranforderung, Sortierung von Daten) kombinieren.

Andere mögliche Erweiterungsideen sind die Auswertung von mehreren Eigenschaften der Dateien. Zum Beispiel werden die Daten gelöscht wenn sie mindestens eine Replikation besitzt und ihre Größe > XY ist und ihr Format 'SNAPSHOT' ist. Das Datenmanagement mit Datenbank ermöglicht die Datei mit vieler identifizierte Metadaten-Eigenschaften im System. Die Abfragen nach Daten wird durch die Durchführung und Änderung von SQL-Bedingungen erleichtert.

4.3 Verschieben von Daten

Neben der Operation Komprimieren und Löschen, besteht noch andere Möglichkeit um Speicherkapazität in verteilten Arbeitsumgebung zu schaffen: die Anwendung der Operation Verschieben. Dabei werden folgende Ansätze von Basis-Ansatz bis zu mehreren Erweiterungen beobachtet.

4.3.1 Basis Ansatz

Idee

Beim Basis-Ansatz der Operation Verschieben werden die Daten von Benutzer in einer Quell-Ressource ausgewählt und zu einer entfernten Ziel-Ressource nacheinander verschoben. Bekannt ist dem Nutzer die Quell-Ressource und Ziel-Ressource. Der Nutzer hat Zugriffsrechte auf beide Ressourcen und die Datenübertragung können mit gewöhnlicher Netzwerk Protokolle wie FTP, SFTP, GridFTP usw. durchgeführt werden.

Pseudocode 5

Algorithmus 5 Verschieben alle Daten von QuellRessource zum ZielRessource

Eingabe: QuellRessource, Pfade zur Daten bei QuellRessource, ZielRessource wo die Daten hinverschoben werden und ZielPfade am ZielRessource

Resultat: Die freie geschaffte Speicherkapazität in Quell-Ressource und neue besetzte Speicherkapazität in Ziel-Ressource.

```
 1: procedure VERSCHIEBENDATEN(QuellRessource,root,ZielRessource,ZielPfade)
 2:     if Verzeichnis (root) leer ist then
 3:         return;
 4:     else
 5:         for all e ∈ root do
 6:             if Element (e) Verzeichnis ist then
 7:                 VERSCHIEBENDATEN(QuellRessource,e,ZielRessource,ZielPfade)
 8:             else
 9:                 VERSCHIEBEN(QuellRessource,e,ZielRessource,ZielPfade)
10:             end if
11:         end for
12:     end if
13: end procedure
```

Angenommen, die Daten sind vorher von einem Dateisystem oder Datenbank verwaltet worden, liegen in einem einzelnen Verzeichnis oder unter hierarchische Verzeichnisse. Die Daten werden rekursiv abgefragt (vgl. Zeile 7), es kann mit einer SQL-Anfrage an Datenbank oder Such-Algorithmen im Dateisystem aufgerufen werden. Die zurückgegebenen

Dateien von Daten Management System werden nacheinander zur Ziel-Ressource verschickt(vgl. Zeile 9).

Diskussion

Der Basis-Ansatz ist einfach, ähnlich wie Basis-Ansatz 4.1.1 von Komprimierung. Man kann noch mit anderen beschriebener erweiterte Ansätze (Speicheranforderung 4.1.2, Sortiert-Funktion 4.1.3) kombinieren, um die Leistung und Effizienz weiter zu verbessern. Der Unterschied zu anderen Operation ist, dass die Operation Verschieben ganz abhängig von der Netzwerk Leitung für die Datenübertragung und die entfernte Speicherressource ist. Datenverlust könnte während der Datenübertragung passieren, wenn die Netzwerkverbindung instabil oder unterbrochen wird. Jedoch werden in diesem Kontext die Aspekte der Datenübertragung wie Datenintegrität, Datensicherheit, Netzwerk Leistung, Software für die Datenübertragung als konstant angenommen oder nicht berücksichtigt. Der weitere erweiterte Ansatz wird die Vorteile bei der Operation Verschieben ausnutzen, um die Effizienz bei der Auswahl und Verschicken von Daten zu verbessern.

4.3.2 Erweiterung des Ansatzes

Der Algorithmus erweitert den Basis-Algorithmus der Operation Verschieben (vgl. 4.3.1) mit mehreren anderen Ziel-Ressourcen.

Idee

Der Ansatz beginnt mit einer Quell-Ressource und mehreren Ziel-Ressourcen. Die Daten von einer Quell-Ressource können nun an mehrere Ziel-Ressourcen verteilt werden statt nur an eine Ziel-Ressource wie am Basis-Ansatz. Der zusätzliche Aspekt ist in diesem Fall die Einführung von Quota. Quota ist die zugewiesene Speicherkapazität, die Benutzer von Administrator bekommen hat. Der Zustand der Quota informiert der Benutzer über die verfügbare und besetzte Speicherkapazität, in der ein Benutzer ihre Daten speichern kann. Außerdem hat Quota ein Limit, als obere Schranke, wobei die gespeicherte Datenmenge nicht überschritten werden darf.

Das Problem tritt auf wenn die Quota an einem Ziel-Ressource beschränkt ist, und nicht über genügend freie Speicherkapazität für die Daten verfügt. Man kann eine Kombination von Operationen anwenden, zuerst Schaffung freier Speicherplatz in Ziel-Ressource mit der Operationen wie Komprimieren, Löschen und dann verschieben die Daten von einer QuellRessource zur ZielRessource (Basis-Ansatz).

Andere Möglichkeit in diesem Fall ist die Anwendung von weitere anderen Ziel-Ressourcen. Dafür kann man die geschickte Daten von Quell-Ressource weiter an die anderen Ziel-Ressourcen gleich verteilen. Angenommen in diesem Kontext, dass Nutzer mehrere unterschiedliche Ziel-Ressourcen besitzt.

Pseudocode 6

Algorithmus 6 Verschieben alle Daten von QuellRessource zur mehreren ZielRessourcen mit Quota

Eingabe: Quell-Ressource, Pfade zur Daten bei Quell-Ressource, Liste der Ziel-Ressourcen, ZielPfade.

Resultat: Die freie geschaffte Speicherkapazität in Quell-Ressource und neue besetzte Speicherkapazität in Ziel-Ressourcen.

```
 1: procedure VERSCHIEBENDATEN(QuellRessource,root,ListResc,ZielPfade)
 2:     if Verzeichnis (root) leer ist then
 3:         Return;                          ▷ Ausgang wenn das Verzeichnis leer ist
 4:     else
 5:         for all e ∈ root do
 6:             if Element (e) Verzeichnis ist then
 7:                 VERSCHIEBEDATEN(QuellRessource,e,ListResc,ZielPfade)
 8:             else
 9:                 VERSCHIEBERESSOURCE(QuellRessource,e,ListResc,ZielPfade)
10:             end if
11:         end for
12:     end if
13: end procedure
14:
15: function VERSCHIEBERESSOURCE(QuellRessource,e,ListResc,ZielPfade)
16:     for all i ∈ die Länge von Liste (ListResc) do
17:         if Verfügbare Quotas von Ressource (ListResc[i]) > 0 then
18:             VERSCHIEBE(QuellRessource,e,ListResc[i],ZielPfade)
19:             Return true;          ▷ Ausgang nach dem erfolgreichen Verschieben
20:         end if
21:     end for
22:     Return false;                  ▷ Ausgang nach dem erfolglosen Verschieben
23: end function
```

Der Unterschied zum Basis-Ansatz von Verschieben ist die Eingabe einer Liste der Ziel-Ressourcen. Jede eingegebene Ziel-Ressource in der Liste hat eigene freie Quota (Speicherkapazität), die zum Speichern von neuen Daten **mit unterschiedlichen Größen** besetzt werden. Die Zustände von Quota werden vom Nutzer abgefragt bzw. berechnet. Die Voraussetzung für das Verschieben von Daten ist Stand des Quota von Ziel-Ressource (vgl. Zeile 17). Alle Dateien werden zur Ziel-Ressource nacheinander verschoben (vgl.Zeile 9) bis der Status des Limit-Quota dieses Ziel-Ressource erreicht hat und dann weiter zur anderen Ziel-Ressource verschoben (vgl. Zeile 16). Der Algorithmus terminiert nachdem die Funktion *VerschiebeRessource* den Wert *True* oder *False* zurückgibt. Das Endergebnis kann sein, dass entweder alle Dateien zum Ziel erfolgreich verschoben werden wenn alle Ziel-Ressourcen über genügend freie Speicherkapazität verfügen oder erfolglos, wenn keine Ziel-Ressource keine verfügbare freie Speicherkapazität hat.

Diskussion

Das Verschieben von Daten auf mehrere Ziel-Ressourcen ermöglicht die Ausnutzung mehrerer freie Speicherkapazitäten in der verteilter Ressourcen-Umgebung. Dadurch kann mehr Speicherkapazität als im Basis-Ansatz freigegeben werden. Außerdem kann man diesen Ansatz mit erwähnten oberen Ansätzen (Speicheranfordeurng, Sortierung von Daten) kombinieren. Eine andere mögliche Verbesserung ist die Sortierung von Ziel-Ressourcen in der eingegebenen Liste nach ihrer freie verfügbare Quota, bevor die Daten dahin verschickt werden. Nach jedem Verschieben von Daten werden die freien verfügbaren Quotas bei allen ZielRessourcen aktualisiert. Dies bewirkt auf die sortierte Reihenfolge von Ziel-Ressourcen, dass die Daten immer zur Ziel-Ressource mit höchster freier verfügbare Quota höchstwahrscheinlich erfolgreich verschickt werden.

Verbesserung der Quota-Abfrage:

Die If-Bedingung (vgl.Zeile 17) könnte das Überschreiten von Limit-Quota des Ziel-Ressource verursachen. Das heißt wenn die Größe der verschickten Daten größer als die freie Speicherkapazität der Ziel-Ressource ist, aber das Quota hat noch freien Speicher, kann die Datenübertragung auch noch stattfinden. Aber entweder werden die Daten erfolgreich übertragen und Status des Quota überschritten oder nur ein Teil der Daten erfolgreich zum Ziel übertragen werden, jedoch die komplette Datei insgesamt als defekte Daten (nicht komplett übertragen) bewertet. Folgende Bedingung verbessert diese Lücke:

If (freie verfügbare Quota-Zustand > die Größe der verschiebende Daten)

Durch die Änderung der Bedingung kann man möglich noch um einige neue Ideen erweitern. Zum Beispiel kann die Priorität von Ziel-Ressourcen für die verschiebende Daten nicht nur nach der freier Quota sondern auch nach der Bewertung anderer Faktoren (z.B. Netzwerk Leistung, die Überlastung von FTP-Servern, Standorten usw.) entschieden werden. Außerdem kann man auch die Speicherung von Daten mit ihrer Eigenschaften (Metadaten) nach bestimmter Speicherressourcen klassifizieren (z.B. welche Speicherressource für Daten mit welcher Dateiformate sind). Dies bedeutet im Kontext wissenschaftlicher Umgebung mit verschiedener Datenformaten (SNAPSHOT, Dokumente, ...) und unterschiedliche IT-Ressourcen-Infrastrukturen. Ein mögliches Beispiel ist:

If (Dateiformat == 'SNAPSHOT') then verschieben nach Ressource A
ElseIf (Dateiformat == 'Dokument') then verschieben nach Ressource B

5 Auswahl und Einführung der Technologie

Im Kapitel 4 haben wir erfahren, welche Verfahren für welche Lösungsansätze eingestellt werden. Um diese Verfahren zu realisieren, werden in diesem Kapitel einige prominente Technologien im Ausblick angeschaut und bewertet (vgl. 5.1). Danach wird eine kurze Einführung auf die ausgewählte Technologie vorgestellt(vgl. 5.2).

5.1 Auswahl der Technologie

Die Anforderungen für die Auswahl der Technologie entsprechen zu wichtiger Kriterien in Verwaltung verteilter Speicherkapazität für die wissenschaftliche Daten bzw. den Schwerpunkt der Arbeit Erlaubnis die Ausführung der Operationen auf dem Datenspeicher für die **Schaffung freier Speicherplatz**. Einige zusammengefasste Aspekte werden als Ziel für den Auswahl der Technologie beschrieben:

T1 Verwalten der gespeicherten Daten (Datenbank, Dateisystem), ermöglicht die Suche, Abfrage nach Daten.

T2 Verwalten der Speicherkapazität (Quota, Administration Tool). Die Speicherräume können reorganisiert werden, um die neue Daten aufzunehmen.

T3 Unterstützt die offene Methode in der Datenverarbeitung für die Implementierung von der vorgeschlagener Verfahren (bzgl. Rekursion, Replikation, Metadaten, Sortieren) (vgl. 4), Schaffung freier Speicherkapazität durch die Verarbeiten von Daten (Komprimieren, Löschen, Verschieben) sowohl Lokal als auch in verteilten Multi-Ressourcen.

T4 Einsetzen in der wissenschaftliche Daten-intensive Umgebung, wobei die Aspekte Zugriffsleistungen, Sicherheit, Skalierbarkeit, Interoperabilität, Erweiterbarkeit, Standard Open Protokolle im Ausblick oft berücksichtigt werden[15].

5.1.1 Dropbox

Dropbox[1] ist ein bekanntes Client-Programm, das die Daten auf unterschiedlichen Plattformen (Windows, Mac, Linux, Smartphone) automatisch repliziert. Der Vorteil von Dropbox ist die automatische Synchronisierung von Daten, wenn der Rechner von Benutzer mit dem Internet verbunden ist. Die Daten werden zwischen Client-Rechner mit Dropbox-Online-Server ausgetauscht um den aktuellen Stand für alle Seiten synchron zu halten. Die Verwaltung von Daten in Dropbox ist jedoch abhängig vom Dateisystem vom Betriebssystem des Nutzers. Die beschränkten Speicherkapazitäten für die angemeldeten Benutzer sind 2 GB Gigabyte (Free Version als Start) bis 18 GB (Erweitert), (Pro Version) bis 500GB.

5.1.2 OGSA-DAI

OGSA-DAI [2] (Open Grid Services Architecture - Data Access and Integration) ist ein Open Source Java-Framework zum Aufbau den verteilte Datenzugriff und Integration Systeme, eine Lösung für die Abfrage und Zugriff von Daten aus vielen verteilten Datenbanken. OGSA-DAI unterstützt andere Grid-Middleware (z.B. Data Access Integration Service (DAIS) wurde in Globus Toolkit integriert), OGSA-DAI unterstützt 3 Arten von Grid Services(vgl. Abbildung 5.1):

- Data Access and Integration Service Group Registry(DAISGR): unterstützt die Registrierung von Daten Ressource im System und ermöglicht dem Client die Suche nach Daten.

- Grid Data Service Factory (GDSF): funktioniert als persistente Datenzugriff Position für Client mit beschriebener Metadaten.

- Grid Data Service (GDS): funktioniert als vorübergehende Datenzugriff Position für Client. GDS wird von GDSF erzeugt. Durch GDS kann Client die Daten aus Daten Ressource zugreifen.

Die Daten in Data Ressource werden über mehrere Schritten mit Services identifiziert, ermöglicht die effiziente Suche und Zugriff nach Daten in großer Datenmenge, die in verteilter heterogene Storage Ressourcen lagern. OGSA-DAI hat die Vorteile in verteilte wissenschaftliche Datenumgebungen durch deren Integration in zahlreiche prominente wissenschaftliche Projekte (z.B. AstroGrid, BioGrid, GEON, BRIDGES usw.) nachgewiesen[3]

Zwei weitere Erweiterungen verstärken die Fähigkeiten von OGSA-DAI Middleware:

OGSA-DQP (Distributed Query Processing) ist ein Lösungsbeispiel von Daten Integration, unterstützt die neue Ressource bei der Integration ins verteilte System, um die Antwort für die Anfrage von Benutzer zu verstärken. OGSA-DAI DQP ermöglicht den Aufbau einer virtuellen Datenbank, indem viele verteilte Daten Ressourcen gebündelt werden.

[1] https://www.dropbox.com/
[2] http://www.ogsadai.org.uk/

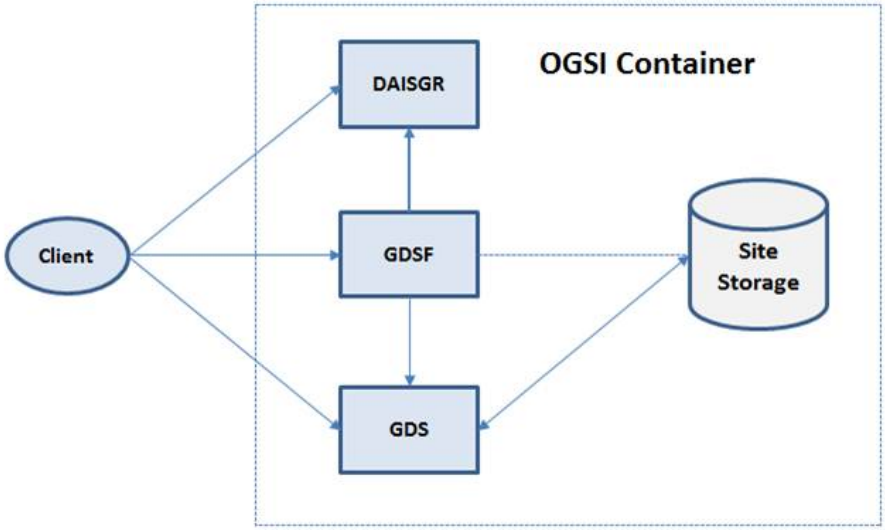

Abbildung 5.1: OGSA-DAI

OGSA-DAI VIEWS unterstützt eine flexible Methode um die Ansicht auf relationale Daten zu definieren. Eine *Datenbank View* ist eine virtuelle Tabelle, bestehe aus eine Menge der Resultaten von gespeicherte Anfragen. Eine *Datenbank View* wird auf dem Top von Ressource aufgebaut um SQL Query auszuführen [6].

5.1.3 dCACHE

dCACHE[3]: Ziel dieses Projektes ist ein System für die Verwaltung der Archivierung und Austausch von hunderte Terabyte-Datenmenge auf verteilten heterogenen Storage Servern. Es wurde in mehreren Rechenzentren installiert (LHC Tier I: SARA (Amsterdam), IN2P3 (Lyon), gridKa (Karlsruhe), Brookhaven (US) und FermiLab (US)). dCache bietet einzige virtuell Dateisystem für den Namensraum, um den Zugriff auf heterogene Storage System zu ermöglichen. Außerdem unterstützt dCACHE auch Methoden für die Verwaltung der Speicherkapazität, Replikation der Datenmenge, Wiederherstellung von abgestützten Storage Servern. Einige technische Kennungen von dCACHE sind Fehlertoleranz; Datei Replikation Manager; Datentransfer Protokolle mit GssFTP, GsiFTP; Load Balancing; Daten Grid Funktionalität mit GIP, SRM.

Die Komponente GIP: Um den zentrale Service in der Auswahl geeigneter Storage Elemente für den Datentransfer zu unterstützen, muss jedes *Storage Element* genügende Informationen über den aktuellen Stand von sich selbst beschreiben. Dies besteht aus die gesamte Speicherkapazität und die verfügbare noch Speicher. Ein Interface *Generic Information Provider (GIP)* ist verantwortlich um diese Information verfügbar an verbundene Grid Middleware zu machen.

Die Komponente SRM: LCG Storage ELement ist ein Protokoll um den Storage Bereich kontrollierbar zu machen. Das Interface *Storage Resource Manager (SRM)* ermöglicht

[3] http://www.dcache.org

Namensraum Operationen und Vorbereiten die Datenübertragung nach Client oder Drittetransfer zwischen *Storage Elemente*[9].

5.1.4 Cloud Computing (Amazon S3)

Es gibt viele Definitionen von Cloud Computing. Cloud Computing beschreibt einen skalierbaren Pool von Ressourcen bzw. IT-Infrastruktur, die aus Rechner, CPU, Speicherkapazität, Netzwerk, Daten, Software usw. besteht. Eine bekannte Eigenschaft von Cloud Computing Modellen ist die Methode *pay as-you-go*, dass man nach Bedarf unterschiedliche IT Ressourcen kaufen kann, wobei die IT-Services als Multi-Schichten (IaaS, PaaS, SaaS) angeboten werden. Das Speichern von Daten ist abhängig von Cloud-Anbieter (bsp. Amazon, Google, Microsoft) , und daher können die Daten in einem nicht vertrauenswürdigen Host lagern wo Zugriffsrechte und Zugriffskontrolle beschränkt werden könnten. Auf die Daten kann daher unerlaubt zugegriffen werden. Außerdem werden die Daten repliziert, oft über große geographische Distanzen, damit die Daten und Applikation bei einem Absturz an einem Standorten bestehen bleiben können[1].

Amazon Web Service (AWS) bietet *Simple Storage Service* (S3)[4], eine Speichern Applikation nach der Cloud-Methode *pay as you go*. Daten Speichern in S3 ist über einen zwei-schichten Namensraum organisiert. Auf dem Top-Level sind *Buckets* (ähnlich wie Container oder Verzeichnis), die einzige globale Name haben und viele Zwecke bedienen: ermöglicht dem Nutzer die Gliederung ihrer Daten, Berechnung von Datentransfer und Daten Speichern. Jeder Amazon Web Services (AWS) Account könnte bis 100 S3 *Buckets* erweitern. Jedes *Bucket* kann unbeschränkt Anzahl der Daten Objekte speichern. Jedes Objekt hat einen Name und das größte Objekt kann bis zu 5 GB groß sein.

Suchen mit der Anfrage ist auf Objekt Name und einzelne *Bucket* limitiert. Die mögliche Suchen mithilfe Metadaten oder inhaltbasiert sind nicht von S3 unterstützt. Zur Zeit unterstützt S3 drei Datenübertragung Protokolle: SOAP, REST und BitTorrent.

Einige Limitationen, die aus Experimente über Performance in S3 evaluiert wurden, sind z.B. limitierte Unterstützung für große gemeinschaftliche Applikation, Zugriffskontrolle zum Daten Ressource aus vielen Teilnehmern, wenig gut skalierbar für großes System mit Tausende Nutzer, unterstützt limitierte Anzahl der Zugriffsrechte usw. [16].

5.1.5 Grid Computing (GlobusToolkit)

Grid Middleware-Globus Toolkit[5] I.Foster definiert ein System als Grid mit drei Eigenschaften wie folgt:

1. Ein Grid integriert und koordiniert die Ressourcen und Nutzer in vielen Gebieten mit unterschiedlicher Kontrolle, Richtlinien und Vorgehensweise.

[4] http:/s3.amazonaws.com
[5] http://www.globus.org/toolkit/

2. Ein Grid wird aus vielen Open-, universale-,und Standard-Protokolle und Interfaces aufgebaut, die fundamentale Probleme wie Authentifizierung, Autorisierung, die Suche nach Ressourcen, Ressourcen Zugriff ansprechen.

3. Ein Grid ermöglicht eine Nutzung der koordinierte Ressourcen um die verschiedene Qualitäten des Dienstes zu erbringen zum Beispiel Reaktionszeiten, Durchsatz, Verfügbarkeit, Sicherheit, gemeinsame Belegung von Ressourcen-Arten so dass die komplexe Anforderungen von Nutzern erfüllt werden[7].

Globus Toolkit (GT) ist ein Grid Middleware OpenSource, die für den Aufbau von verteilter Systeme bzw. Grid Infrastrukturen benötigt. Globus Toolkit bietet eine Reihe von Infrastruktur Diensten, die Interfaces für die Verwaltung der Rechenkapazität, Speicherkapazität und andere Ressourcen implementieren. Die Architektur von Globus Toolkit ist durch die Modularität aufgebaut (vgl. Abbildung 5.2), besteht aus vielen separater Komponente wie Grid Security Infrastructur (GSI), Ressource Management, Information Management, Daten Management und Applikation.

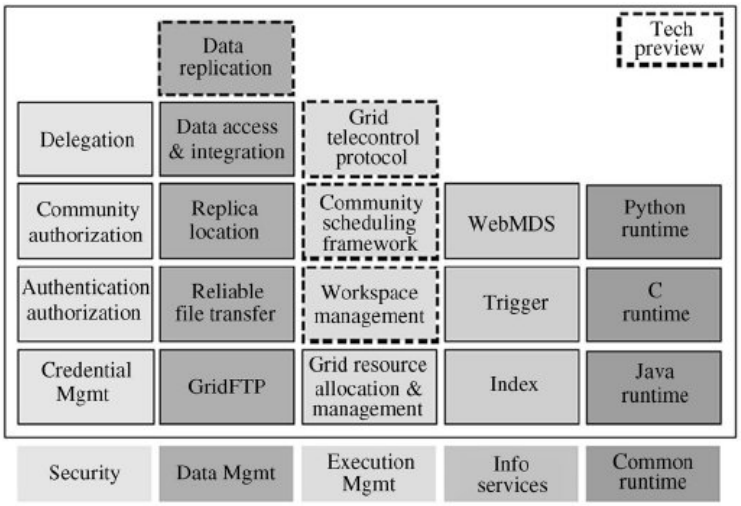

Abbildung 5.2: Globus Toolkit 4 Komponente [8]

Die interessante Komponenten von GT4 sind dabei *Transport*, *Replikation Manager*, und *Resource Discovery, Monitoring, and Management Architecture.*

Transport GridFTP ist die Erweiterung von Standard FTP Protokoll, es übernimmt die Aufgabe der Datenübertragung in Grid Umgebung. GridFTP bietet einen sicheren und hohen Datenübertragungsdurchsatz, sogar auf Hochgeschwindigkeitsverbindung in Wide Area Networks (WAN) und Übertragungen an Dritte.

Replikation die zweite wichtige Komponente in Globus für das Erstellen Data Grid ist *replica management service*. Diese Dienste besteht aus einer *replica catalog*, wo die Informationen über Replikation von Daten gespeichert werden und Abfrage-Operationen wie Datei registrieren, erstellen und die Kopie einer registrierte Datei löschen, Kopie der Daten bestimmen.

Ressource Management „***Virtual Organization (VO)*** *ist eine Reihe von Freigaberegeln, die auf mehrere Einzelpersonen oder Instituten angewendet werden. Diese Regeln legen fest, wie Sharing von Ressourcen (Daten, Computer, Software oder andere Ressourcen) zwischen Ressource-Anbieter und Endverbraucher unter welcher Bedingungen stattfinden, was freigegeben wird, wer freigeben darf, um die kooperative Probleme in der Wissenschaft zu lösen [7].* “

Eine wichtige Komponente von Globus für verteilte *Virtual Organization (VO)* ist die Möglichkeiten wie die Suche nach Ressourcen, gemeinsame Ressourcennutzung, Ressourceverbrauch überwachen, die VO bilden. Globus Monitoring Discovering System (MDS)-2 implementiert eine Dienst, besteht aus 2 weitere Komponenten, *Grid Resource Information Servers (GRISs)*, die mit Ressourcen assoziiert, und Grid Information Index Server (GIIS), die Informationen aus verschiedene GRISs zusammenstellen. Diese Dienste sind verbunden durch *Grid Information Protocol* (die von Client benutzt wird, um die Information Dienste abzufragen) und *Grid Registration Protocol* (die von Ressource-Anbieter benutzt wird, um die Information über die Ressourcen zu registrieren und diese Information nach Periode zu aktualisieren).

Performance-Überwachung ist andere wesentliche Dienste für Data Grids. Die Applikation und Job Vermittlung müssen den Zustand des Netzes (Netzwerksbandbreite, verfügbare Speicherkapazität im Storage Servers, Belastungen von Rechenkapazität, den Verlauf des Datentransfers) kontrollieren. Gestützt auf diesen Informationszustand, kann die Applikation oder Job Vermittlung die bessere Scheduling machen und die richtige Entscheidung für die Auswahl der Ressource treffen[2].

Globus Toolkit wurden bereits in vieler wissenschaftlichen Grid-Projekte eingesetzt zum Beispiel TeraGrid[6], Open Science Grid [7], EGEE [8], LHC Computing Grid [9] usw[8].

5.1.6 Data Grid (iRODS)

Data Grid sind Software Middleware, die die Bildung gemeinsamer Sammlungen aus verteilten Daten ermöglichen. Die Daten können in unterschiedlichen Arten von Storage System in unterschiedlichen Standorten liegen, die durch das Internet verbunden sind. *Data Grid* verwaltet die nützlichen Informationen über die Daten, z.B. wo die Daten gespeichert werden, Metadaten von Daten, Replikation von Dateien, und Kommunikation im Netzwerk. Data Grid implementiert die Virtualisierung durch die Authentifizierung von Benutzer, Zugriffskontrolle auf Dateien, Metadaten und Storage Systeme. *Data Grid* unterstützt eine Reihe von standardisierten Operationen für den Benutzer um auf die Dateien, die sich auf entfernten Rechner befinden zugreifen zu können. Diese Operationen beinhalten Byte-Level (öffnen, schließen, lesen, schreiben, suchen, usw.), Datei-Level (replizieren, Backup, Version erzeugen, Metadaten analysieren, usw.) und Collection-Level (rekursive Operationen auf Verzeichnissen, Zugang zu großer Datenmenge, große Datenmenge verschieben)[15].

[6]The TeraGrid Projekt 2006, www.teragrid.org
[7]Open Science Grid (OSG) 2006, www.opensciencegrid.org
[8]Enabling Grids for eScience(EGEE). 2006, http://public.eu-egee.org
[9]LHC Computing Grid. 2006,http://lcg.web.cern.ch/LCG

Large-scale Data Grid Systems (LDGS) ist ein System, das die Zugriffe für große Sammlungen von Daten, deren Größe bis zu Petabyte erreicht und bis zu hundert Millionen Dateien unterstützt. LDGS bietet ein *Collaboration Framework* wobei Multi-Nutzer ihre Daten aufnehmen, speichern und die Zugriffsrechte darauf an anderer Nutzer bzw. Gruppen weitergeben können. Data Grid Middleware **integrated Rule-Oriented Data System (iRODS)**[10] ist ein Beispiel von LDGS[17].

Das Ziel des iRODS Teams *Data Intensive Cyber Environments* (DICE) ist die Entwicklung einer generischen Softwarelösung, die alle verteilte Daten Management Applikation implementieren kann, durch die Veränderung der **Regeln** und **Verfahren**. Das Ergebnis der Arbeit ist eine hohe erweiterbare Software Infrastruktur, die weiter von Benutzer entwickelt bzw modifiziert werden kann ohne dass der Software Code neu entwickelt werden muss. Das iRODS kann die verschiedenen Daten Management Applikation implementieren (z.B. von Data Grid für die Datenfreigabe in der Zusammenarbeit; bis digitale Bibliothek für die Veröffentlichung von Daten; Datenverarbeitung-Pipeline; System für Real-Time Sensor Daten Streams, die zu einem Bund zusammengeschlossen sind).

iRODS Architektur (vgl. Abbildung 5.3)

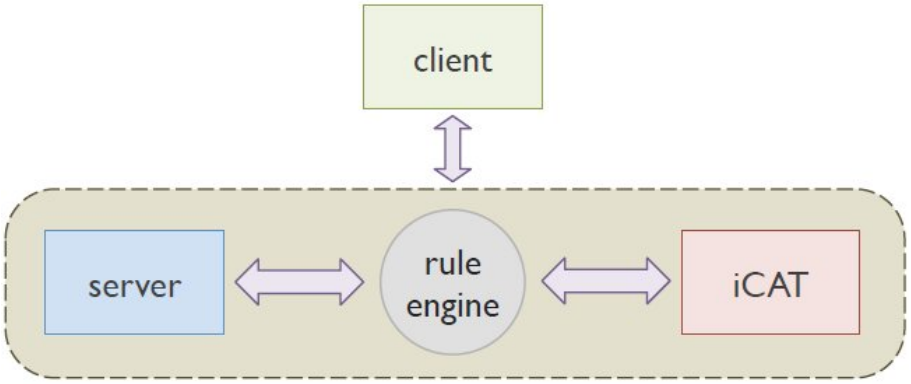

Abbildung 5.3: iRODS Architektur

Die Architektur von iRODS besteht aus 3 wichtigen Komponenten:

Data Grid Architektur basiert auf Client-Server-Modell, die die Interaktion mit verteilte (Storage und Computing) Ressource kontrollieren. iRODS Server ist bei dem Standort, wo die physikalische Daten gespeichert werden, installiert und interagiert mit Storage System durch Interpretieren der Operationen in geeignete Protokolle. Client Interface sind verschieden, von Web-basierten (Java, PHP) bis Unix-Style Shell Befehle.

Metadata Catalog wird von einem relationale Datenbank Management System verwaltet, sie erhalten die Attribute (Metadaten) von Daten und persistente Zustandsinformation von Daten System und Ressourcen in iRODS, sogenannte *iCAT*.

[10] https://www.irods.org

Rule System enthalten ein *Rule-Engine*, das die Regeln aus *Rule-Base* interpretieren um die Kontrolle auf die Operationen von iRODS-Server und Client auszuführen. Ein *Rule-Base* besitzen Basis-Regeln für die Kontrolle und ist repliziert in jeden iRODS Server. Wenn ein Regel aus *Rule-Base* von bestimmter Aufgabe (Task) aufgerufen wird, wird dann die Operationen ausgeführt. Diese Operationen sind *Micro-Services* (in C-Programmiertsprache geschrieben). Man kann die verschiedene Verfahren (Task) einsetzen, indem man die existierte Regeln im *Rule-Base* benutzt, oder *Rule-Base* mit neuer Regeln und neuer erzeugte *Micro-Services* erweitern.

Eine typische Situation verdeutlicht die Arbeitsweise von iRODS. Die Client schickt die Anforderung für die Datenverarbeitung über Netzwerk an iRODS Server. Der Server interagiert mit iCAT Metadata Catalog um die Authentifizierung vom Benutzer und der angeforderter Operationen zu validieren. Der Standort, wo die Operationen ausgeführt werden, wird identifiziert und die Anforderung von Client wird an Standort von *Remote Storage* weitergeleitet. Ein *Rule Engine* an diesem Standort wählt die geeignete Regeln aus *Rule-Base* und ruft die *Micro-Service* für die Ausführung der Operationen auf.

iRODS implementiert *Logical Name Space*(vgl. Abbildung 5.4) für Benutzer, Dateien, und Storage Ressourcen um die Interaktion in verteilter Umgebung mit den Ressourcen in unterschiedlichen Standorten zu erleichtern.

Logical Name für Benutzer jede Person wird beim Zugriff ins Data Grid mit einzigartigen Name authentifiziert.

Logical Name für Dateien und Collection Data Grid organisiert die verteilte Dateien in eine hierarchische Struktur, die durchgesucht werden können. Ein *logical Collection* kann die *logical Dateien* enthalten, die aus unterschiedlichen Standorten kommen.

Logical Name für Storage Ressourcen Data Grid kann die Ressourcen gruppieren und die Operationen darauf anwenden[18].

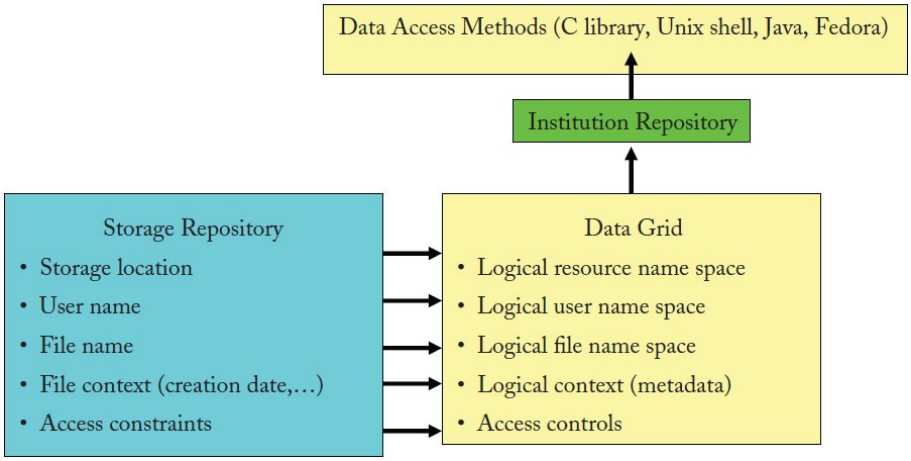

Abbildung 5.4: iRODS Logical Name Spaces [18]

Wie Globus Toolkit, wurde iRODS auch in vielen hohe-skalierbaren globalen wissenschaftliche Projekten eingesetzt, z.B. SHAMAN (Sustaining Heritage Access through Multiva-

lent Archiving), Ocean Observatory Initiative und Large Synoptic Survey Telescope, wo iRODS die eigene Stärke der hohen Skalierbarkeit ausnutzt, um 100 Millionen Dateien und Sammlungen auf der Ebene des Petabytes[17] zu verarbeiten.

5.1.7 Vergleichstabelle von Technologien

Die folgende Tabelle (vgl. Tabelle 5.1) vergleicht alle obige Technologien und bewertet sie mit der Anforderung für die Auswahl (vgl. 5.1) (✓: schlecht, ✓✓: gut, ✓✓✓: sehr gut)

Technologie/ Anforderung	Dropbox	OGSA DAI	dCACHE	Cloud AmazonS3	Globus Toolkit	iRODS
T1	✓✓	✓✓✓	✓✓✓	✓✓✓	✓✓✓	✓✓✓
T2	✓✓	✓✓✓	✓✓	✓✓✓	✓✓✓	✓✓✓
T3	✓	✓	✓	✓	✓✓✓	✓✓✓
T4	✓	✓✓	✓✓	✓✓	✓✓✓	✓✓✓

Tabelle 5.1: Vergleich der Technologien

Diskussion

Dropbox Mit beschränkter Speicherkapazität ist Dropbox geeignet für einzelne Benutzer mit weniger Datenmenge.

OGSA-DAI bietet Kontrolle über verteilte Datenbanken und ermöglicht die schnelle Suche nach Daten. Aber OGSA-DAI muss ins anderer Middleware (z.B. Globus Toolkit) integriert werden.

dCACHE unterstützt die Verwaltung von heterogenen Storage Systemen in wissenschaftlicher Umgebungen. Aber für die Implementierung der vorgeschlagenen Verfahren (vgl. Kapitel4) ist dCACHE wenig geeignet.

Cloud Amazon S3 unterstützt Cloud-Computing-Modell per Methode *pay as you go*, aber für die Implementierung von Verfahren und Operationen auf Speicherraum ist es wenig geeignet. Die Anwendung von Amazon S3 in wissenschaftlich verteilter Umgebung ist noch im Experiment (keine bekannten Projekte).

Globus Toolkit vs iRODS zwei Kandidaten, die alle Anforderungen erfüllt haben, sind Globus Toolkit und iRODS. Die Stärke beider Technologien wurden schon in vielen globalen wissenschaftlichen Projekten anerkannt. Die Auswahl für die Technologie liegt daher an T3 bzw. die Aufwände für die Implementierung von vorgeschlagenen Verfahren (vgl. Kapitel4). Daran würde iRODS mit regelbasierte Programmierung ausgewählt. Mit zahlreichen vorimplementierten Regeln und *Micro-Services* Bibliothek im iRODS könnten die Verfahren und Lösungsansätze mit **weniger Aufwand** als Globus Toolkit implementiert werden.

5.2 Einführung in die ausgewählte Technologie

Für die Implementierungen mit ausgewählter Technologie **iRODS** von der Verfahren werden einige nötige Aspekte im Ausblick aus dem Buch [18] zusammengefasst.

5.2.1 iRODS

Um die Verfahren (vgl. Kapitel 4) zu implementieren werden die Regeln im iRODS entworfen. Im Folgenden werden benötigte Basis-Information von *iRODS Rule System* aufgeklärt wie die implementierte Regeln mit *Micro-Function* aufgebaut und ausgeführt werden.

Rule-Oriented Programming (ROP)

Eine Sitzung startet wenn Client mit dem iRODS Server verbindet, und beendet wenn Client trennt. Während der Sitzung, gibt es zwei *state information* die von Actions und Micro-Services kontrolliert sind.

Persistent State Information (gekennzeichnet mit #) sind verfügbare Informationen während der Sitzung und bleiben in *iCAT Metadata Catalog* nach der Ausführung von Action, zum Beispiel die Information über Benutzer, Daten, Ressourcen. Dies können von Rule-Entwickler durch *Persistent State Information Variable* aufgerufen werden. Eine Liste aller Variable ist verfügbar unter [11] oder durch Aufruf von i-Command (iquest attrs).

Session State Information (gekennzeichnet mit $) bleiben nicht in iCAT wie Persistent State Information, sondern sie existieren während der Sitzung, z.B. um die Werte aller Attribute abzufragen. Eine Liste aller *Session Variablen* liegt in der Datei auf dem iRODS Server /server/config/reConfigs/core.dvm oder online unter [12] und [13].

User Environment Information definiert welche Umgebung der Benutzer eingeloggt hat. Die *Environment Variable* bestimmt welche Data Grid (Host, Adresse) zugegriffen werden. Die Basis-Information von *User Environment Variable* sind irodsUserName (Benutzername zum Einloggen ins iRODS DataGrid), irodsHost (Host Adresse von iRODS Data Grid Server), irodsPort (Portnummer für Data Grid Metadata Catalog, default ist 1247) und irodsZone (einzige Identifikation für den Arbeit Zone im iRODS Data Grid). Diese Information liegt in einer Datei /.irods/.irodsEnv speichert die Konfiguration für den Zugriff.

[11] https://www.irods.org/index.php/Persistent_ State_ Information_ Variables
[12] https://www.irods.org/index.php/Session_ State_ Variables
[13] https://www.irods.org/index.php/Session_ Variables_ Available_ for_ Each_ Rule

Ein Beispiel *User Environment Variable* ist wie folgt:

irodsHost 'zuri.sdsc.edu'
irodsPort 1378
irodsUserName 'rods'
irodsZone 'tempZone'

Rule System

Rule Engine kontrolliert alle Regeln im Rule Base und entscheidet welche Regeln angewendet wird. Die erste Regel in der Liste wird mit ihrer Bedingung überprüft. Wenn ihre Bedingung nicht erfüllt, wird die nächste Regeln aufgerufen. Wenn keine Regel die Bedingungen erfüllt ist, wird die Aktion ausgefallen und ein Fehlerstatus (negative Nummer) wird zurückgegeben. *Micro-Services* können die Struktur Rule Execution Infrastructure (REI) benutzen, um die Fehlermeldung auszugeben. Die REI Struktur wird von *Micro-Services* und die Aktionen für die Übergeben der Informationen benutzt wenn eine Regel aufgerufen ist. Wenn die Bedingungen erfüllt sind, dann wird die *Micro-Services* und Aktionsketten in der Regeln nacheinander ausgeführt. Wenn alle *Micro-Services* erfolgreich ausgeführt werden, wird die Aktion komplett als erfolgreich betrachtet und ein Erfolgsstatus (0) ausgegeben[14].

Es gibt zwei Arten von Regeln in iRODS.

System Level Rules sind die Regeln, die auf dem iRODS Server intern aufgerufen werden, um die *Management Policies* für das System auszuführen. Zum Beispiel *Daten Management Policies* wie Authentifizierung, Integrität, Zugriffskontrolle.

User Level Rules können extern von Client durch Befehl i-Command *irule* oder *rcExec-MyRule API* aufgerufen werden. Zum Beispiel die Regeln von Client, die an iRODS-Server angefordert werden, um eine Reihe von Operationen auszuführen.

Die Basis-Komponente von einem Regeln besteht aus 4 Teile

Name | Bedingung | Workflow-chain | Recovery-chain

Name ist der Name der Regeln

Bedingung die Regeln wird ausgeführt nachdem die Bedingung erfüllt wurde.

Workflow-chain sind die Kette von *Micro-Services und Rules*, die von dieser Regeln ausgeführt werden.

Recovery-chain sind die Menge von *Micro-Services und Rules*, die aufgerufen werden wenn die Ausführung von irgendeiner *Micro-Services/ Rules* in *Workflow-chain* ausfällt.

[14] https://www.irods.org/index.php/iRODS_ Rule_ Engine

Rule Base beinhaltet die *Default iRODS Rules* in der Datei core.irb auf dem iRODS-Server unter /server/config/reConfigs. Eine ausführliche Liste befindet sich online unter [15].

Es gibt 2 Schreibweisen von Regeln, deren Dateien mit unterschiedlicher Formaten (.r und .ir)gespeichert wurden. Das Format .ir ist eine Kette mit dem Aussehen, beschrieben wie oben (Name | Bedingung | Workflow-chain | Recovery-chain). Das Format .r ist neues Format, hat die ähnliche Struktur wie *high-level Programmiertsprache*. Die neue Version von iRODS Rule-Engine können beide Format interpretieren, und der Befehl irule kann beide Formaten aufrufen (z.B. irule -F DateiName.r). Das implementierte Verfahren im nächsten Kapitel wird die Schreibweise .r benutzen. Weitere Informationen befindeen sich online [16].

Micro-Service sind die fundamental programmierten Funktionen von iRODS. Jedes *Micro-Services* führt einen spezifischen Task aus. *Micro-Services* wird von iRODS Team entwickelt, in C-Programmiersprache geschrieben und mit iRODS-Server am Storage Standort installiert. Bei jeder neuer Versionen von iRODS wird die Bibliothek von *Micro-Services* weiter neu erweitert. Benutzer und Administrator können mehrere *Micro-Services* zu einem Bund zusammenschließen um eine große Funktionalität zu implementieren. Ein *Micro-Services* Funktion beginnt mit dem Präfix msi. Eine detaillierte Liste von verfügbarer *Micro-Services* befindet sich online unter [17].

Wichtige iRODS Shell Commands Die kompletten Befehle mit ausführlicher Beschreibung befinden sich online unter [18]. Die nützlichen i-Commands von iRODS kann man in folgende Gruppen einteilen.

Unix-like Commands ils (Daten in iRODS-Umgebung auflisten), icd (das aktuelle Verzeichnis ändern), iexecmd (einen auf dem Server installierten Befehl ausführen), irepl (Daten/ Objekte replizieren) usw.

Metadaten isysmeta (Metadaten zeigen), iquest (die Information aus iCAT Datenbank abfragen) usw.

Informational iquota (Information über iRODS Quotas) usw.

Administration iadmin (Administration Befehlen) usw.

Rules, Delayed Rule Execution irule (ladet die geschriebene Regel von Benutzer an iRODS-Server hoch) usw.

[15] https://www.irods.org/index.php/Default_ iRODS_ Rules
[16] https://www.irods.org/index.php/Changes_ and_ Improvements_ to_ the_ Rule_ Language_ and_ the_ Rule_ Engine
[17] https://www.irods.org/doxygen/
[18] https://www.irods.org/index.php/icommands

5.2.2 Bash Shell

Um die vorgeschlagener Verfahren zu motivieren, werden neben der angewendeter Technologie iRODS, ein Teil der Verfahren im lokalen Kontext implementiert. Das Ziel dieser Arbeit motiviert die Verwaltung von Daten unter der Datenbank iCAT mit iRODS und Dateisystem Linux mit Bash Shell.

Eine Unix Shell is sowohl ein Interpreter von Befehlen als auch eine Programmiersprache. Als Interpreter von Befehl, bietet Shell dem Benutzer eine große Reihe von GNU Dienstprogrammen[19]. Die Eigenschaften einer Programmiersprache unterstützt die Möglichkeit, diese Dienstprogramme miteinander zu kombinieren. Die neue erzeugte Dateien, die unterschiedliche Dienstprogramme enthalten, sind selbst ein Befehl zur Nutzung geworden und haben den gleichen Zustand wie originale Dienstprogramme/ System Befehl in Verzeichnis /bin. Damit kann der Benutzer oder Gruppe ihre Arbeitsumgebung individuell einrichten um ihre Task/ Arbeit/ Aufgabe zu automatisieren.

Shell Skripting ist ein Art von Software, die die Kommunikation-Task mit dem Kernel im Unix, Linux System erleichtern. Es existieren viele Versionen von *Shell-Scripting*, die von unterschiedlicher Instituten, Personen aus Universitäten entwickelt wurden wie commonly-used shell (von Stephen R.Bourne in AT&T Bell Labs 1974), C Shell (/bin/csh) (von Bill Joy in Berkeley 1978) und Korn shell (/bin/ksh) und TC shell (/bin/tcsh) erweitern zwei originale Shell (Bourne Shell und C Shell). Bash (/bin/bash) (Bourne-Again Shell) (von Brian Fox entwickelt) ist ein *command language interpreter, for the gnu operating system* und die verbesserte Shell-Version über alle anderen Shell-Versionen. Wie alle High-Level Programmiersprache, unterstützt Shell auch Variablen, Fluss-Kontrolle-Struktur, Funktionen. Eine interessante Eigenschaft von Bash Shell ist die Anwendung von Pipeline. Ein Pipeline ist eine Reihe einer einfachen Befehlen, die durch den Kontrolle-Operator | getrennt wird, die miteinander in eine Kette kombiniert werden indem das Ergebnis von erstem Befehl eine Eingabe für zweiten Befehl ist usw.

Beispiel *grep Diplomarbeit /home/muster/Desktop/Diplomarbeit/Diplomarbeit.txt | wc -l*
Der Befehl grep sucht in der Datei Diplomarbeit.txt das Wort 'Diplomarbeit' durch. Das Ergebnis von Befehl grep wird in weiteren Befehl wc eingegeben, der Befehl wc zählt wie oft das Wort in der Datei existiert.

Jeder Programmdatei von Bash Shell muss mit einer Zeile #!/bin/bash beginnen, um mit anderen Shell-Version zu unterscheiden. Die Aufruf von Bash-Programm erfolgt über die Syntax (bash Dateiname oder ./Dateiname). Die Bash-Datei muss mit geeigneter Rechte zuweisen, um ausführbar machen zu können (Beispiel: chmod +x Dateiname) [5][19]. Eine Referenz über komplette Syntax, Funktionen von Bash Shell befindet sich online unter [20]

[19] http://ss64.com/bash/
[20] http://www.gnu.org/software/bash/manual/

iRODS und SHELL

Mit iRODS kann man eine einheitliche Datenzugriffe und Authentifizierung auf Multi-Speicherressourcen in verteilter Umgebung. Die Daten werden im logische iRODS-Raum organisiert, ermöglicht einen Überblick auf die Daten aus unterschiedlicher Standorten, Storage Ressourcen (z.B. Grid Ressource, FTP Server, Cloud Ressource).

Mit Shell kann man manuell die separate Verbindungen an Multi-Speicherresourcen erstellen, dafür braucht man möglicherweise unterschiedliche Authentifizierung und Zugriffsinformationen (siehe Abbildung 5.5).

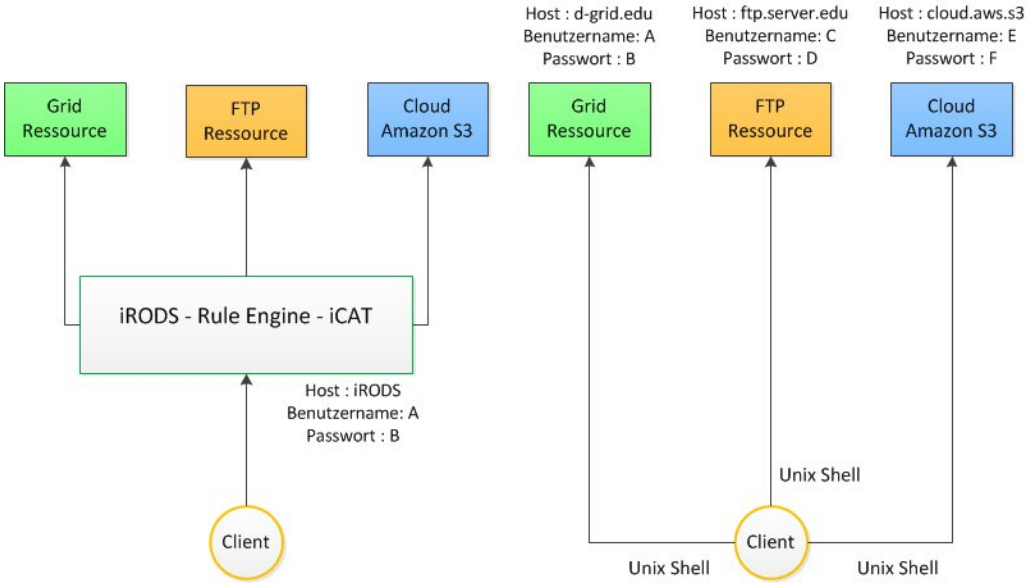

Abbildung 5.5: iRODS und Shell

Kombination von iRODS und SHELL
Die Kombination von i-Command-iRODS mit Shell-Programmierung könnte interessant sein. iRODS unterstützt eine große Menge von Client i-Commands [21] im Verzeichnis (iRODS/client/icommands/bin/..). Man kann diese iRODS-Befehle mit Bash-Shell zusammenstellen um die Stärke von Beiden auszunutzen. Ein Beispiel wäre die Abfrage ins iRODS Umgebung mit iRODS-iCommands, die zurückgegebene Daten am Lokal werden weiter von Bash-Shell verarbeitet. Die Datenverarbeitung am Lokal wird daher vereinfacht, weil Bash-Shell eine große Bibliothek aus Unix-Umgebung unterstützt.

[21] https://www.irods.org/index.php/icommands

6 Implementierung

6.1 Implementierung mit iRODS

Um die Ansätze aus Kapitel 4 zu untersuchen, werden die vorgeschlagenen Verfahren bei jedem Ansatz mit der ausgewählter Technologie aus Kapitel 5 Datagrid Framework iRODS und Bash Shell Skripting implementiert. Die Implementierungen erfolgen mit Hilfe von Musterbeispielen (iRODS/clients/icommand/test/rules3.0/), die zur iRODS Framework (Version 3.1) gehört sind.

6.1.1 Basis Ansatz

Im Folgenden ist die Implementierung für die Verfahren aus Kapitel *4.1* (vgl. Quelltext 6.1)
Die logische Adresse der Datei (beginnt mit INPUT ...) im iRODS wird von Benutzer eingegeben. Die Daten wird dann vom externen Programm gzip komprimiert. Das Programm gzip muss bereits auf dem iRODS-Server unter dem Pfad /iRODS/server/bin/cmd/ installiert werden. Die Regel ruleExecOut wird jeden angewendeten Mikro-Service ausführen (beginnt mit OUTPUT ...).

Die Eingabe ist iRODS-logische Adresse *File (vgl. Zeile 41) und der Aufruf der *Micro-Function msiExecCmd* wird die Datei aus den eingegebener Pfaden komprimieren, durch die Anwendung des externem Programmes gzip (vgl. Zeile 17). Der benötigte Parameter für *Micro-Service msiExecCmd* ist die physikalische Adresse von Daten (vgl. Zeile 9), die man mit einer SQL-Anfrage (vgl. Zeile 5) von iCAT Metadata Catalog (PostgreSQL Datenbank) zu der Datei (vgl. Zeile 5) ermitteln kann. Am Ende des Programms wird die gewonnene Speicherkapazität berechnet (vgl. Zeile 39). Eine Struktur *ruleExecOut* (ist ein Teil von inMsParamArray) emuliert *stdout und stderr Buffer*. Wenn Befehl irule alle Regeln fertig ausgeführt hat, wird *Buffer* auf dem Bildschirm dargestellt. Diese Darstellung wird durchgeführt wenn ruleExecOut als OUTPUT Argument (vgl. Zeile 43) bei jeder Datei (.r) oder (.ir) eingefügt wird. Auch wenn die Pfade *INPUT* falsch ist oder die Datei nicht existiert, wird die Fehlermeldung (stderr Buffer) ausgegeben[1].

Erweitert wird der Basis-Ansatz auf einzelne Datei, wird die Aktion auf mehrere Dateien im hierarchischer Verzeichnisse angewendet (vgl. Quelltext 6.2). Die Implementierung basiert auf dem Verfahren aus Kapitel 4.1.1. Insbesondere wird der rekursive Aufruf mithilfe vorgefertigter *Micro-Service* durch die Abfrage von Daten zu *iCAT Metadata Catalog*

[1] https://www.irods.org/index.php/Test_ Writing_ to_ stdout_ and_ stderr_ Buffers

Quelltext 6.1: Komprimieren einer einzelnen Datei

```
1  basisansatz{
2   # Micro−Service: trennt die Eingabe in Verzeichnis und Datei
3   msiSplitPath(∗Coll,∗Collection,∗Name);
4   # Micro−Service: Abfragen Daten aus iCAT
5   msiExecStrCondQuery("SELECT DATA_SIZE,DATA_PATH,DATA_ID where
6                        COLL_NAME= '∗Collection' AND DATA_NAME like '∗Name'",∗F);
7   foreach (∗F) {
8    # Micro−Service: erhaltet die absolute Pfade von Datei zu komprimieren
9    msiGetValByKey(∗F,"DATA_PATH",∗Path);
10   # Micro−Service: erhaltet die Groesse von Datei bevor Komprimierung
11   msiGetValByKey(∗F,"DATA_SIZE",∗Size);
12   # Micro−Service: erhaltet iRODS−pfade von Datei
13   msiGetObjectPath(∗Collection++"/"++∗Name,∗Ladress,∗status);
14   # Micro−Service: nimmt DATA ID
15   msiGetValByKey(∗F,"DATA_ID",∗DataID);
16   # Micro−Service: Komprimiert die Datei
17   msiExecCmd("/iRODS/server/bin/cmd/gzip",∗Path,"null","null","null",∗Result);
18   # Micro−Service: stellt das Ergebnis dar.
19   writeLine("stdout","Compression successfull"); }
20  # Aktualisiert neue logical iRODS−Pfade
21  ∗NewLadress= "∗Ladress"++".gz";
22  # Aktualisiert neue absolute Pfade
23  ∗NewPath= "∗Path"++".gz";
24  # Erstellt neuen Datentyp
25  ∗NewZipName= "∗Name"++".gz";
26  # Micro−Service: Registriert die neue komprimierte Datei ins iCAT.
27  msiPhyPathReg(∗NewLadress,∗StorageResc,∗NewPath,"null",∗Stat);
28  # Micro−Service: fuehrt die Abfrage Data_ID mit neuer Information (Zip−Name) aus.
29  msiExecStrCondQuery("SELECT DATA_ID where COLL_NAME= '∗Collection'
30  AND DATA_NAME= '∗NewZipName'",∗ID);
31  foreach(∗ID) {# erhaltet die Werte data_id um neue Datentyp (Ziv) zu erstellen
32          msiGetValByKey(∗ID,"DATA_ID",∗NewDataID);
33          msiSetDataType(∗NewDataID,∗NewLadress,"ascii compressed Lempel−Ziv",∗Status); }
34  # Abfrage Daten mit neuen Zip−Format.
35  msiExecStrCondQuery("SELECT DATA_SIZE where DATA_NAME= '∗NewZipName'
36  AND DATA_TYPE_NAME= 'ascii compressed Lempel−Ziv'",∗Z);
37  foreach(∗Z) { # erhaltet die neue Zip−Werte
38          msiGetValByKey(∗Z,"DATA_SIZE",∗ZipSize); }
39  ∗Win = double(∗Size) − double(∗ZipSize); # Win ist die gewonnene Speicherkapazitaet
40 }
41 # Beispiel: ∗Coll = "/tempZone2/home/irods/StorageResc2/bigfile.big", ∗StorageResc= "StorageResc2"
42 INPUT ∗Coll
43 OUTPUT ruleExecOut
```

erleichtert.

Alle Informationen (Metadaten) über die gespeicherten Dateien von hierarchischen Verzeichnisstrukturen werden im *iCAT-Metadata Catalog* bewahrt. Die Abfrage zur iCAT wird durch SQL (vgl. Zeile 7) für Metadaten der Datei gebildet. Um diese Information für **alle Dateien** in der SQL-Abfrage zu bestimmen, muss man die Pfade zum Start-Verzeichnis mit % eingeben (vgl. Zeile46), damit wird die Syntax der SQL-Bedingung mit 'like' (vgl. Zeile 5) geändert. Die Hilfsvariablen (vgl. Zeile 2) bestimmen wie viele Speicherkapazitäten nach dem Ablauf von Programm geschafft werden und (vgl. Zeile3) bestimmt wie viele Dateien bearbeitet werden. Standardmäßig kann der Buffer nur eine bestimmte Anzahl der ausgegebenen Zeile der Datenbank ausgeben. Um die weitere Zeile auszugeben muss man die Abfrage mit extra *Micro-Services* (vgl. Zeilen 10 und 41) ausführen.

Außerdem kann man die Abfrage SQL (vgl. Zeile 7) manipulieren, um die Datei nach bestimmten Eigenschaften/ Metadaten (z.B. Größe, Format, Name usw.) zu bearbeiten, indem man die Bedingung SQL nach *Persistent State Variable* [2] ändert[3].

6.1.2 Verbesserungsansätze

Im Folgenden ist die Implementierung für die Verfahren aus Kapitel 4.1.2 (vgl. Quelltext 6.3). Die Speicheranforderung wird durch INPUT eingegeben. Die Implementierung bearbeitet bzw. komprimiert die Dateien bis die mögliche Speicherbedarf erreicht wird.

Die Implementierung wird ergänzt um die eingegebene Speicheranforderung (vgl. Zeile 48) bei der Eingabe. Die Eingabe bestimmt auch den Name des Quell-Ressource, wo die Daten bearbeitet werden. Im logischem iRODS-Raum existieren mehrere Ressourcen aus unterschiedlichen Standorten. Die geschaffte Speicherkapazität wird während der Bearbeitung mit der Speicherbedarf verglichen (vgl. Zeile 13). Am Ende der Implementierung werden die gesamte geschaffte Speicherkapazitäten mit der Speicherbedarf von Benutzer verglichen (vgl. 41), ob die benötigte Speicherkapazität erreicht wurde und wie viele Dateien bereits komprimiert wurden.

Sortierfunktion

Die weitere Implementierung ist die Ergänzung um eine *Sortierfunktion* (vgl. Kapitel 4.1.3), wobei die SQL-Abfrage geändert wird und die Daten absteigend sortiert bearbeitet werden. Die Zeile 7 wird durch folgende Zeile ersetzt.

*msiMakeGenQuery("COLL_NAME,DATA_PATH,**order_desc(DATA_SIZE)**,*
*DATA_NAME,DATA_ID",*Condition,*GenQ);*

[2] https://www.irods.org/index.php/Persistent_State_Information_Variables
[3] https://www.irods.org/index.php/iquest

Quelltext 6.2: Komprimieren mehrere Dateien

```
1  basisansatz2 {
2   *Sum= 0; # Hilfsvariable: Berechnet die Gewinn von freier Speicherkapazitaet.
3   *I= 0; # Hilfsvariable: Berechnet die Anzahl der komprimierte Dateien, initialisiert= 0
4   *ContOld= 1;
5   *Condition= "COLL_NAME like '*Coll' AND RESC_NAME = '*StorageResc'";
6   msiMakeGenQuery("COLL_NAME,DATA_PATH,DATA_SIZE,DATA_NAME,DATA_ID",
7   *Condition,*GenQ); #Erstellt eine Abfrage mit Bedingung *Condition, gib das Ergebnis *GenQ zurueck.
8   #Fuehrt eine erstellte Abfrage aus msiMakeGenQuery aus, gib das Ergebnis *QOut zurueck
9   msiExecGenQuery(*GenQ, *QOut);
10  #Nehmen die Index *QOut weiter, gib Ergebnis *ContNew zurueck
11  msiGetContInxFromGenQueryOut(*QOut,*ContNew);
12  while(*ContOld > 0) { # Ueberprueft die Bedingungsvariable
13   foreach(*QOut) {
14          msiGetValByKey(*QOut, "DATA_NAME", *Name);
15          msiGetValByKey(*QOut, "COLL_NAME", *Collection);
16          msiGetObjectPath(*Collection++"/"++*Name,*Ladress,*status);
17          msiGetValByKey(*QOut, "DATA_PATH", *Path);
18          msiGetValByKey(*QOut, "DATA_SIZE", *Size);
19          msiGetValByKey(*QOut,"DATA_ID",*DataID);
20          msiExecCmd("/iRODS/server/bin/cmd/gzip",*Path,"null","null","null",*Result);
21          *I= *I + 1; # Berechnet die Anzahl der komprimierte Dateien
22          *NewLadress= "*Ladress"++".gz";
23          *NewPath= "*Path"++".gz";
24          *NewZipName= "*Name"++".gz";
25          msiPhyPathReg(*NewLadress,*StorageResc,*NewPath,"null",*Stat);
26          msiExecStrCondQuery("SELECT DATA_ID where COLL_NAME= '*Collection'
27                  AND DATA_NAME= '*NewZipName'",*ID);
28          foreach(*ID) {
29            msiGetValByKey(*ID,"DATA_ID",*NewDataID);
30            msiSetDataType(*NewDataID,*NewLadress,"ascii compressed Lempel-Ziv",*Status); }
31          msiExecStrCondQuery("SELECT DATA_SIZE where DATA_NAME= '*NewZipName'
32          AND DATA_TYPE_NAME= 'ascii compressed Lempel-Ziv'",*Z);
33          foreach(*Z) # erhaltet neue Zip-Groesse
34                  { msiGetValByKey(*Z,"DATA_SIZE",*ZipSize); }
35          *Win= double(*Size) - double(*ZipSize);
36          *Sum= *Sum + *Win; # Aktualisierte die geschaffte Speicherkapazitaet.
37          writeLine("stdout","WIN Free Capacity is *Sum");
38          }
39          *ContOld= *ContNew; # bekommt neuen Wert, ob es neue Zeile gibt
40          # Abfrage weitere Dateien aus iCAT Datenbank, falls die Anzahl der Zeile von Datenbank > 256 Zeile.
41          if(*ContOld > 0) {msiGetMoreRows(*GenQ,*QOut,*ContNew);}
42   }
43   writeLine("stdout","Anzahl der komprimierte Dateien: *I");
44  }
45  # Beispiel *Coll= "/tempZone2/home/irods/StorageResc2/Storage2%", *StorageResc= "StorageResc2"
46  INPUT *Coll, *StorageResc
47  OUTPUT ruleExecOut
```

Quelltext 6.3: Komprimieren mehrere Dateien unter Berücksichtigung des Speicherbedarfs

```
1  verbesserungsansatz1{
2   *Sum= 0;
3   *I= 0;
4   *ContOld= 1;
5   *Condition= "COLL_NAME like '*Coll' AND RESC_NAME = '*StorageResc'";
6   msiMakeGenQuery("COLL_NAME,DATA_PATH,DATA_SIZE,DATA_NAME,DATA_ID",
7                   *Condition,*GenQ);
8   msiExecGenQuery(*GenQ, *F);
9   msiGetContInxFromGenQueryOut(*F, *ContNew);
10  while(*ContOld > 0) {
11   foreach(*F) {
12    # Wenn die geschaffte Speicher noch kleiner als die Speicheranforderungen ist, weiter komprimieren
13    if (*Sum < *X ) {
14            msiGetValByKey(*F,"DATA_NAME",*Name);
15            msiGetValByKey(*F,"DATA_PATH",*Path);
16            msiGetValByKey(*F,"DATA_SIZE",*Size);
17            msiGetValByKey(*F,"COLL_NAME",*CollName);
18            msiGetObjectPath(*CollName++"/"++*Name,*Ladress,*status);
19            msiGetValByKey(*F,"DATA_ID",*DataID);
20            msiExecCmd("/iRODS/server/bin/cmd/gzip",*Path,"null","null","null",*Result);
21            *I= *I + 1;
22            *NewLadress= *Ladress++".gz";
23            *NewPath= *Path++".gz";
24            msiPhyPathReg(*NewLadress,*StorageResc,*NewPath,"null",*Stat);
25            *NewZipName= *Name++".gz";
26            msiExecStrCondQuery("SELECT DATA_ID where COLL_NAME= '*CollName'
27            AND DATA_NAME= '*NewZipName'",*ID);
28            foreach(*ID) {
29             msiGetValByKey(*ID,"DATA_ID",*NewDataID);
30             msiSetDataType(*NewDataID,*NewLadress,"ascii compressed Lempel−Ziv",*Status);
31            }
32            msiExecStrCondQuery("SELECT DATA_SIZE where DATA_NAME= '*NewZipName'
33            AND DATA_TYPE_NAME= 'ascii compressed Lempel−Ziv'",*Z);
34            foreach(*Z) { msiGetValByKey(*Z,"DATA_SIZE",*ZipSize); }
35                    *Win= double(*Size) − double(*ZipSize);
36                    *Sum= *Sum+ *Win; }
37   }
38   *ContOld= *ContNew;
39   if(*ContOld > 0) {msiGetMoreRows(*GenQInp,*F,*ContNew);}
40  }
41  if (*Sum < *X) {
42   writeLine("stdout","Leider nur *I Dateien und *Sum freie Speicherkapazitaet");
43  } else {
44          writeLine("stdout","Erfolgreiche, *I Dateien und *Sum freie Speicherkapazitaet");
45          }
46  }
47  # z.B.: *Coll= "/tempZone2/home/irods/FTPResc2%", *StorageResc = "FTPResc2", *X = 1500000
48  INPUT *Coll, *StorageResc, *X
49  OUTPUT ruleExecOut
```

6.1.3 Löschen mit Replikation

Im Folgenden ist die Implementierung für die Verfahren aus Kapitel 4.2 (vgl. Quelltext 6.4). Benutzer möchte freie Speicherkapazität in einer Ressource (z.B. StorageResc2) durch das Löschen von Daten gewinnen. In diesem Fall werden nur die Daten, die schon eine Replikation (Kopie) in einer anderen Ressource (z.B. FTPResc2) besitzen, bearbeitet.
Annahme: Die Operation Löschen auf die Daten kann nicht rückgängig machen. Das heißt, die gelöschten Daten werden danach komplett vom System entfernt.

Die Implementierung beginnt mit der Ermittlung aller Dateien mit mindesten einer Replikation durch SQL-Anfrage an iCAT Datenbank (vgl. Zeile 6). Die *Persistent State Variable* DATA_REPL_NUM bestimmt ob die Datei eine Replikation hat. Wenn die Datei eine Replikation hätte, wird die erste Replikation mit 1 nummeriert. Alle Dateien mit bereits einer Replikation werden ermittelt, in welcher Ressourcen ihre Replikation liegen (vgl. Zeile 18). Dann wird die Datei gelöscht in der Ressource wo man die freie Speicherkapazität braucht (vgl. Zeile 24).

iRODS unterstützt die Verwaltung der Replikation der Datei mit Replikation-Manager in iCAT. Man kann die Existenz der Replikation von Daten im Datenbank durch *Persistent State Variable DATA_REPL_NUM* überprüfen[4] und die Entscheidung schnell treffen.

[4]https://www.irods.org/index.php/Replicate_Each_File_in_a_Collection

Quelltext 6.4: Löschen die Daten unter Berücksichtigung der Replikation

```
1  deleteReplikation {
2   *Sum= 0;
3   *I= 0;
4   *ContOld= 1;
5   # Erstellt die Bedingung von der Abfrage Daten mit mindestens eine Replikation
6   *Condition= "COLL_NAME like '*Coll' AND DATA_REPL_NUM > '0'";
7   msiMakeGenQuery("COLL_NAME,DATA_NAME,DATA_SIZE",*Condition,*GenQ);
8   msiExecGenQuery(*GenQ, *F);
9   msiGetContInxFromGenQueryOut(*F,*ContNew);
10  while(*ContOld > 0) {
11   # Schleifen aller Dateien mit bereits mindestens einer Replikation.
12   foreach(*F) {
13          msiGetValByKey(*F,"DATA_NAME",*Name1);
14          msiGetValByKey(*F,"COLL_NAME",*CollName1);
15          *SourceFile= *CollName1++"/"++*Name1;
16          # Bestimmt welche Ressource die Replikation der Datei bewahrt
17          msiExecStrCondQuery("SELECT RESC_NAME where COLL_NAME= '*CollName1'
18          AND DATA_NAME='*Name1' AND DATA_REPL_NUM= '1'",*R);
19            foreach(*R) {
20                  msiGetValByKey(*R,"RESC_NAME",*Resource);
21                  writeLine("stdout","Die Datei *Name1 hat eine Kopie in *Resource");
22                  }
23          # Micro-Service: Reduziert die Datei *SourceFile in Ressource *StorageResc um eins
24          msiDataObjTrim(*SourceFile,*StorageResc,"null","1","null",*Status);
25          msiGetValByKey(*F,"DATA_SIZE",*Size1);
26          *I= *I + 1;
27          *Sum= *Sum + double(*Size);
28          }
29  *ContOld= *ContNew;
30  if(*ContOld > 0) {msiGetMoreRows(*GenQInp,*F,*ContNew);}
31  }
32  if(*I > 0) {
33          writeLine("stdout","Anzahl der geloeschte Dateien: *I");
34          writeLine("stdout","Die gesamte geschaffte Speicherkapazitaet: *Sum");
35  } else {
36          writeLine("stdout","Keine Dateien wurde geloescht");
37  }
38 }
39 # *Coll= "/tempZone2/home/irods%", *StorageResc = "StorageResc2"
40 INPUT *Coll, *StorageResc
41 OUTPUT ruleExecOut
```

Quelltext 6.5: Verschieben mehrere Dateien von einer Ressource zu einer anderen Ressource

```
1  verschiebenansatz {
2    *Sum= 0;
3    *I= 0;
4    *ContOld= 1;
5    *Condition= "COLL_NAME like '*Coll' AND RESC_NAME= '*QuellResource'";
6    msiMakeGenQuery("COLL_NAME, DATA_SIZE, DATA_NAME",*Condition,*GenQ);
7    msiExecGenQuery(*GenQ, *QR);
8    msiGetContInxFromGenQueryOut(*QR,*ContNew);
9    while(*ContOld > 0) {
10           foreach(*QR) {
11             msiGetValByKey(*QR,"DATA_SIZE",*Size);
12             msiGetValByKey(*QR,"DATA_NAME",*Name);
13             msiGetValByKey(*QR,"COLL_NAME",*Collname);
14             *SourceFile= *Collname++"/"++*Name;
15             # Verschiebden die Datei *SourceFile von *QuellResource nach *DescResource
16             msiDataObjPhymv(*SourceFile,*DescResource,*QuellResource,"0","null",*Status);
17             writeLine("stdout","Datei *SourceFile mit der Groesse *Size
18             wurde von *QuellResource nach *DescResource verschoben");
19             *Sum= *Sum + double(*Size);
20             *I= *I + 1;
21       }
22   *ContOld= *ContNew;
23   if(*ContOld > 0) {msiGetMoreRows(*GenQ,*QR,*ContNew);}
24   }
25   writeLine("stdout","Erfolgreich, die freie Speicherkapazitaet *Sum wurde geschafft ");
26   writeLine("stdout","Anzahl der verschobene Dateien *I");
27 }
28 # *Coll= "/tempZone2/home/irods%", *QuellResource= "FTPResc2", *DescResource= "StorageResc2"
29 INPUT *Coll, *QuellResource, *DescResource
30 OUTPUT ruleExecOut
```

6.1.4 Verschieben - Basis-Ansatz

Im Folgenden ist die Implementierung für die Verfahren aus Kapitel 4.3.1
Die Idee ist das Verschieben von Daten von einer Quell-Ressource zu einer Ziel-Ressource, um freie Speicherkapazität in Quell-Ressource zu gewinnen (vgl.Quelltext 6.5). Die Implementierung erfolgt in der iRODS Umgebung, wo man mit logischen Daten interagiert um die physikalischen Daten zu verschieben.

Die Dateien zum Verschieben werden durch SQL-Abfrage mit dem Standort *QuellResource ermittelt, wo man die freie Speicherkapazität schaffen möchte(vgl. Zeile 5). Das Verschieben Daten von Ressource an Ressource wird mithilfe von *Micro-Service msiDataObjPhymv* (vgl. Zeile 16) ausgeführt. Die benötige Parametern dafür sind die Eingabe

den Quell-Ressource, Ziel-Ressource und die Start-Pfade von Dateien-Verzeichnis (vgl. Zeile 29). Die erfolgreiche verschobene Datei wird als die geschaffte Speicherkapazität berechnet (vgl. Zeile 19). Zuletzt werden die gesamte gewonnene Speicherkapazitäten und die Anzahl der verschobener Dateien ausgegeben per die Struktur writeLine.

6.1.5 Verschieben - Verbesserung Ansatz

Im Folgenden ist die Implementierung für die Verfahren aus Kapitel 4.3.2)
Die Idee ist das Verschieben von Daten von einem Quell-Ressource zu zwei unterschiedlichen Ziel-Ressourcen unter Berücksichtigung des Quotas. Die Implementierung erfolgt in zwei Teilen. Teil 1 ermittelt die freie Quota bei allen Ressourcen (vgl. Quelltext 6.6). Teil 2 verschiebt alle Dateien zu freien Ressourcen (vgl. Quelltext 6.7).

Teil 1: Die Implementierung beginnt mit der Berechnung von freier Quota aus eingegebener Ressourcen (QuellRessource Zeile 3, erste Ziel-Ressource Zeile 16 und zweite Ziel-Ressource Zeile 29). Die benutzte Version von iRODS (3.1) unterstützt noch keine *Micro-Service* für die Ermittlung von Quota, deswegen muss man selbst die freie Quota per i-Command **iquota** ermitteln und eingeben. Das Programm wird weiter berechnen wie viele freie Quotas noch verfügbar sind. Diese Ergebnisse der freier Quotas bei allen Ressourcen (vgl. Zeile 14, Zeile 27, Zeile 40) werden weiter in Teil 2 benutzt.

Teil 2: Die Implementierung erfolgt weiter mit Hilfsvariablen, die die Parameter für letztendliche Ergebnisse definieren (vgl. Zeile 1). Dann werden die benötigten Parameter von Dateien aus iCAT per SQL für das Verschieben abgefragt (vgl. Zeile 8). Die Bedingung überprüft und entscheidet, welche Ziel-Ressource zuerst die verschiebende Daten aus Quell-Ressource bekommen wird (vgl. Zeile 13), dabei werden die freie Quotas von zwei Ziel-Ressourcen verglichen. Die Ressource mit mehreren freien Quota bekommt erste (höchste) Priorität. Die Dateien von Quell-Ressource wird zuerst an erste Ziel-Ressource verschoben bis Quota von erster Ziel-Ressource nicht mehr frei ist (vgl. Zeile 19) und dann weiter an die zweite Ziel-Ressource verschoben (vgl. Zeile 27). Die Implementierung (vgl. Zeile 39) ist ähnlich, beginnt aber die zweite Ziel-Ressource zuerst.

Verbessern die Überprüfungsbedingung, um Überschritten von Quota zu vermeiden

Um die Bedingung zu verbessern, wird die Implementierungen von Überprüfung-Bedingung (vgl. Zeile 19 und Zeile 27) wie folgt geändert.

if (*Tfree1 >= double(*Size)
und
if (*Tfree2 >= double(*Size)

Außerdem kann die SQL-Abfrage (vgl. Zeile 8 von Quelltext 6.7) mit der Sortierung von der Datengröße (z.B.*order_desc(DATA_SIZE)*) und der benutzerdefinierte Speicheranforderung weiter optimiert werden.

Quelltext 6.6: Teil1-Ermittlung die freie Quotas bei allen Ressourcen

```
1  verschiebenansatz2 {
2    # Phase 1: Ueberprueft die Zustaende von Quota bei allen Storage Ressourcen
3    msiExecStrCondQuery("SELECT SUM(DATA_SIZE) where
4    COLL_NAME like '*Resc' AND RESC_NAME= '*QuellResource' ",*QR);
5    foreach(*QR) {
6          msiGetValByKey(*QR,"DATA_SIZE",*QSize);
7          if (*QSize =="") { # falls Quota ganz frei ist, von keiner Dateien besetzt.
8            writeLine("stdout","1. *QuellResource benutzte Speicherkapazitaet= 0 ");
9          } else {
10           writeLine("stdout","1. *QuellResource benutzte Speicherkapazitaet= "++*QSize);
11         }
12   }
13   # Berechnet die freie Quota bei *QuellResource
14   *Qfree= *QuotaQuellResc - double(*QSize);
15   writeLine("stdout","---> *QuellResource freie Speicherkapazitaet= *Qfree");
16   msiExecStrCondQuery("SELECT SUM(DATA_SIZE) where
17   COLL_NAME like '*Resc' AND RESC_NAME= '*TargetResource1' ",*TR1);
18   foreach(*TR1) {
19         msiGetValByKey(*TR1,"DATA_SIZE",*TSize1);
20         if (*TSize1 =="") {
21           writeLine("stdout","2. *TargetResource1 benutzte Speicherkapazitaet= 0");
22         } else {
23           writeLine("stdout","2. *TargetResource1 benutzte Speicherkapazitaet= "++*TSize1);
24         }
25   }
26   # Berechnet die freie Quota bei *TargetResource1
27   *Tfree1= *QuotaTargetResc1 - double(*TSize1);
28   writeLine("stdout","---> *TargetResource1 freie Speicherkapazitaet= *Tfree1");
29   msiExecStrCondQuery("SELECT SUM(DATA_SIZE) where
30   COLL_NAME like '*Resc' AND RESC_NAME= '*TargetResource2' ",*TR2);
31   foreach(*TR2) {
32     msiGetValByKey(*TR2,"DATA_SIZE",*TSize2);
33         if (*TSize2 =="") {
34           writeLine("stdout","3. *TargetResource2 benutzte Speicherkapazitaet= 0 ");
35         } else {
36           writeLine("stdout","3. *TargetResource2 benutzte Speicherkapazitaet= "++*TSize2);
37         }
38   }
39   # Berechnet die freie Quota bei *TargetResource2
40   *Tfree2= *QuotaTargetResc2 - double(*TSize2);
41   writeLine("stdout","---> *TargetResource2 freie Speicherkapazitaet= *Tfree2");
```

Quelltext 6.7: Teil2-Verschieben alle Dateien nach anderen Ressourcen mit Quota

```
1  *sum= 0; # Hilfsvariable: geschaffte Speicherkapazitaet,
2  *sumfile1= 0; # Hilfsvariable: Anzahl neu besetzter Speicherkapazitaet bei erster Ressource,
3  *sumfile2= 0; # Hilfsvariable: Anzahl neu besetzter Speicherkapazitaet bei erster Ressource,
4  *i1= 0; # Hilfsvariable: Anzahl neu besetzter Speicherkapazitaet bei erster Ressource
5  *i2= 0; # Hilfsvariable: Anzahl neu besetzter Speicherkapazitaet bei zweiter Ressource
6  *ContOld= 1;
7  *Condition= "COLL_NAME like '*Coll' AND RESC_NAME= '*QuellResource'";
8  msiMakeGenQuery("COLL_NAME, DATA_SIZE, DATA_NAME",
9  *Condition,*GenQ);
10 msiExecGenQuery(*GenQ, *T1);
11 msiGetContInxFromGenQueryOut(*T1,*ContNew);
12 # Falls die erste Ziel−Ressource mehr freie Quota als die zweite Ziel−Ressource hat
13 if ( *Tfree1 > *Tfree2 ) {
14   while(*ContOld > 0) {
15    foreach(*T1) {
16           msiGetValByKey(*T1,"DATA_NAME",*Name);
17           msiGetValByKey(*T1,"COLL_NAME",*Collname);
18           msiGetValByKey(*T1,"DATA_SIZE",*Size);
19           if ( *Tfree1 > 0 ) { # Ueberpruefung Quota von erster Ziel−Ressource
20           # Verschieben Daten an TargetResource1
21            *SourceFile= *Collname++"/"++*Name;
22            msiDataObjPhymv(*SourceFile,*TargetResource1,*QuellResource,"0","null",*Status);
23            *sum= *sum + double(*Size);
24            *Tfree1= *Tfree1 − double(*Size);
25            *sumfile1= *sumfile1 + double(*Size);
26            *i1= *i1 + 1;
27           } else if ( *Tfree2 > 0 ) { # Ueberpruefung Quota von zweiter Ziel−Ressource
28            # Verschieben Daten an TargetResource2
29            *SourceFile= *Collname++"/"++*Name;
30            msiDataObjPhymv(*SourceFile,*TargetResource2,*QuellResource,"0","null",*Status);
31            *sum= *sum + double(*Size);
32            *Tfree2= *Tfree2 − double(*Size);
33            *sumfile2= *sumfile2 + double(*Size);
34            *i2= *i2 + 1; }
35    }
36   *ContOld= *ContNew;
37   if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*T1,*ContNew);}
38   }
39 } else { # aehnlich wie obere Teil von Zeile 14 bis Zeile 38, Beginn mit zweiter Ziel−Ressource }
40 writeLine("stdout"," Insgesamt *sum− *sumfile1 mit *i1 Dateien− *sumfile2 mit *i2 Dateien");
41 }
42 # *Resc: Start−Pfade wo man die freie Quota von allen Ressourcen berechnen kann /tempZone2/home/irods%,
43 # *Coll: Start−Pfade wo man die Daten verschieben moechte /tempZone2/home/irods/StorageResc2/Storage1%,
44 # *QuellResource= "StorageResc2", *TargetResource1= "FTPResc2",*TargetResource2= "GridResc2",
45 # *QuotaQuellResc= 30000000, *QuotaTargetResc1= 10000000,*QuotaTargetResc2= 10000000
46 INPUT *Resc,*Coll,*QuellResource,*TargetResource1,*TargetResource2,
47 *QuotaQuellResc,*QuotaTargetResc1,*QuotaTargetResc2
48 OUTPUT ruleExecOut
```

6.2 Implementierung mit Bash Shell Skripting

Die Implementierung mit Bash Shell motiviert noch eine andere Variante für die Schaffung freier lokale Speicherkapazität, wo die Daten von einem Dateisystem verwaltet werden. Die Konstruktion der Shell-Programm wird auf vielen unterschiedlichen Kern-Programmen von Linux-Systemen wie gzip, rm, find, read usw. aufgebaut, und werden von zahlreichen Linux-Gemeinschaft unterstützt.

6.2.1 Basis Ansatz

Die Implementierung basiert auf dem Verfahren (vgl. Kapitel 4.1.1) mit der Operation Komprimierung auf alle Dateien aus eingegebener Pfade (vgl. Quelltext 6.8)

Beschreibung
Zeile 1 ist sogenannte *Shebang* und definiert mit welchem Kommandointerpreter (bzw. welcher Shell) das Skript ausgewertet werden soll [5]. Dabei ist Bash-Shell angewendet.
Die Eingabe zur gespeicherten Daten wird per Befehl read ausgeführt (vgl. Zeile 3). Dabei wird die Pfade des lokalen Rechners eingegeben. Die Suche zur Dateien aus der eingegebener Pfade wird zuerst mit dem Befehl *find* ausgeführt. Der Befehl *find* wird benutzt um die Dateien auf dem Unix/Linux-Systeme aufzufinden indem *find* die Start-Pfade von Verzeichnis traversiert.

Das zurückgegebene Dateiliste wird als die Parameter für die For-Schleife eingegeben (vgl. Zeile 12). Dann wird jedes Element bzw. Datei in der Liste durchgearbeitet. Die Größe der Datei wird mithilfe von Befehl stat ermittelt (vgl. Zeile 16). Das Programm gzip im Linux-System komprimiert die einzelne Daten weiter (vgl. Zeile 19). Dann wird die Größe der neue komprimierte Datei ermittelt um die geschaffte Speicherkapazität zu berechnen (vgl. Zeile 25). Die Hilfsvariable sumfile bestimmt wie viele Dateien komprimiert wurden (vgl. Zeile 28).

[5] http://www.bin-bash.de/scripts.php

Quelltext 6.8: Komprimieren aller Dateien

```
1  #!/bin/bash
2  # Eingabesbeispiel: Path="/home/irobot/Desktop/FTPSource/home/irods/FTPResc2/"
3  read -p "Pfade von Collection/ Datei zur Komprimierung : " Path
4  # Deklariert die Variablen win, origsize, zipsize
5  declare -i win
6  win=0
7  declare -i origsize
8  declare -i zipsize
9  sumfile=0
10 temp='.gz'
11 # Suchen alle Dateien aus eingegebener Pfade mithilfe von Schleife aus Ergebnis von Befehl find durch
12 for f in $(find $Path -iname '*.*' )
13 do
14         echo "$sumfile : FileName : $f"
15         # ermittelt die Groesse der Datei bevor Komprimierung
16         let origsize=`stat -c "%s" $f`
17         echo " -> Original Size = $origsize "
18         # komprimiert die Datei mit Befehl gzip
19         gzip $f
20         # ermittelt die Groesse der Datei nach der Komprimierung
21         echo " -> New Zip Name : $f$temp"
22         let zipsize=`stat -c "%s" $f$temp`
23         echo " -> Zip Size = $zipsize "
24         # Berechnet die geschaffte Speicherkapazitaet
25         let "win += origsize - zipsize"
26         echo " -> Win : $win"
27         # Bestimmt die Anzahl der komprimierte Dateien
28         let sumfile++
29 done
30 echo "Glueckwunsch,$win freie Speicherkapazitaet gewonnen. "
31 echo "Anzahl der ZipFile : $sumfile"
```

6.2.2 Erweiterung Ansatz

1. Mit benutzerdefinierter Speicheranforderung Die Implementierung basiert auf die Verfahren (vgl. mit 4.1.2) mit der Operation Komprimierung die Dateien bis Speicheranforderung erfüllt wird (vgl. Quelltext 6.9).

Beschreibung
Die Bedingung vergleicht die geschaffte Speicherkapazität mit eingegebener Speicheranforderung (vgl. Zeile14). Falls die geschaffte Speicherkapazität die Speicheranforderung schon überschritten hat, wird die Arbeit dann abgebrochen.

Erweiterung Ansatz mit Sortierung Funktion

Die Implementierung kann auch mit zusätzlicher Sortierung der Dateien erfolgen. Der zweiter Ansatz erweitert den ersten Ansatz indem die Sortierung-Funktion 'sort' in die Zeile11 hinzugefügt wird. Die Funktion 'cut -f 2' wählt 2 Spalte aus, die Größe der Daten für die Sortierung benötigt. Die Daten werden dann nacheinander komprimiert. Die Verbesserung von Zeile11 in oberer Implementierung kann wie folgt geändert werden:

fileset = ‘ find $Path -type f -exec du -sb {} \; | sort -nr | cut -f 2 ‘
for f in $fileset

Quelltext 6.9: Komprimieren unter Berücksichtigung der Speicherbedarf

```
1  #!/bin/bash
2  read -p "Bedarf nach freier Speicherkapazitaet : " memRe
3  echo " Speicheranforderung $memRe"
4  read -p "Pfade von Collection/ Datei zur Komprimierung : " Path
5  declare -i win
6  win=0
7  declare -i origsize
8  declare -i zipsize
9  sumfile=0
10 temp='.gz'
11 for f in $(find $Path -iname '*.*' )
12 do
13  # Vergleich die geschaffte Speicherkapazitaet mit der Speicheranforderung
14  if test $win -lt $memRe
15          then
16                  echo "$sumfile : FileName : $f"
17                  let origsize = `stat -c "%s" $f`
18                  echo " -> Original Size= $origsize "
19                  gzip $f
20                  echo " -> Neue Zip-Name : $f$temp"
21                  let zipsize = `stat -c "%s" $f$temp`
22                  echo " -> Zip Groesse = $zipsize "
23                  let "win += origsize - zipsize"
24                  echo " -> Geschaffte Speicherkapazitaet: $win"
25                  let sumfile++
26  fi
27 done
28 if test $win -lt $memRe
29          then
30                  let "fehlt = memRe - win"
31                  echo "Leider, $win freie Speicherkapazitaet geschafft. Es fehlt noch $fehlt Byte"
32                  echo "Anzahl der ZipFile: $sumfile"
33          else
34                  echo "Glueckwunsch,$win freie Speicherkapazitaet gewonnen. "
35                  echo "Anzahl der ZipFile: $sumfile"
36 fi
```

7 Demonstration und Bewertung auf Performance

In Kapitel 4 wurden die Eigenschaften der entwickelten, abstrakten Lösungsansätze hinsichtlich der Kriterien aus Kapitel 3.2 bereits diskutiert. In Kapitel 6 wurden die Verfahren per iRODS und Shell implementiert. In diesem Kapitel wollen wir uns daher auf die Demonstration von implementierter Verfahren (vgl. 7.1) sowie Bewertung die Performance, Effizienz dieser Implementierungen (vgl. 7.2) konzentrieren.

7.1 Demonstration mit iRODS

Die Demonstration simuliert in der Kontext, dass der Wissenschaftler drei unterschiedliche Speicherressourcen (StorageResc2, GridResc2, FTPResc2). Jede Speicherressource hat unterschiedliche frei verfügbare Quota. Besonders, am Speicher-Ressource StorageResc2 sind die besetzte Daten über maximale Quota bereits überschritten. Dafür möchte der Wissenschaftler 20 Terabyte (20.10^6 Megabyte) freie Speicherkapazität schaffen, dadurch die folgende Aktionen mit entsprechender implementierte mit iRODS Verfahren nacheinander ausgeführt werden:

1. Komprimierung einzelne große Dateien am Speicher-Ressource StorageResc2,

2. Komprimierung weitere mehrere Dateien am Speicher-Ressource StorageResc2,

3. Löschen die Dateien mit bereits einer Replikation am Speicher-Ressource StorageResc2,

4. Verschieben mehrere Dateien nach anderen Ressourcen (GridResc2, FTPResc2) bis die noch gebrauchte Speicheranforderung erfüllt wird.

Die experimentierte Daten mit unterschiedlichen Formaten, Größe sind in folgenden Speicherressourcen in iRODS-Umgebung (beginnt mit der Pfade /tempZone2/home/irods/..) wie folgt simuliert:

StorageResc2 *Verfügbare/ Max Quota:* ***-6.202.448 (überschritten)/ 30.000.000*** *(Megabyte)= 30TB*
Storage1/ Archiv1 (500 Dateien archiv1-1.ar1.. archiv1-500.ar500),
Storage1/ Archiv2 (1000 Dateien archiv2-1.1.. archiv2-1000.1000),
Storage2 (500 Dateien storage2-1.st1.. storage2-500.st500),

Storage3 (100 Dateien storage3-1.txt .. storage3-100.txt),
bigfile.big (Große Datei 1,5 TB),

GridResc2 *Verfügbare/Max Quota:* ***7913660/ 10.000.000*** *(Megabyte)= 10TB*
Grid1 (200 Dateien grid1-1.gd1.. grid1-200.gd200),
Grid2 (400 Dateien grid2-1.gi1.. grid2-400.gi400),
Grid3 (500 Dateien grid3-1.gr1.. grid3-500.gr500),
Storage3 (20 Dateien storage3-1.txt .. storage3-20.txt Replikation von StorageResc2),

FTPResc2 *Verfügbare/Max Quota:* ***6099383/ 10.000.000*** *(Megabyte)= 10TB*
FTP1 (300 Dateien ftp1-1.ft1.. ftp1-300.ft300),
FTP2 (400 Dateien ftp2-1.ft1.. ftp2-400.ft400),
Storage3 (20 Dateien storage3-21.txt .. storage3-40.txt Replikation von StorageResc2).

Wegen der beschränkten Hardware-Ressourcen werden die experimentierte Daten mit Faktor 10^6 mal (von Megabyte auf Byte) reduziert.

7.1.1 Komprimieren

1.Demo: Komprimieren einzelner Datei
Der Vorgang beginnt mit der eingegebene von Benutzer Pfade zur Datei (z.B. /tempZone2/home/irods/StorageResc2/bigfile.big). Dabei *tempZone2* ist der Name von Ressource, *home* ist der Wurzel Ordner für alle registrierte Benutzer, *irods* ist Benutzername und *bigfile.big* ist die Datei zu komprimieren.

Dabei ist eine aufgenommene Test-logdatei von abgelaufenem Programm (vgl. Quelltext 6.1):

```
Verzeichnis-Pfade: /tempZone2/home/irods/StorageResc2 und die Datei: bigfile.big
1. Physikalische Pfade: /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/bigfile.big
2. Originale Größe: 1.572.853 Bytes
3. iRODS Path is /tempZone2/home/irods/StorageResc2/bigfile.big
4. DATA ID: 182.964
Komprimierung erfolgreich
********** Neue Zip-Datei ermitteln ************
– Neue iRODS Pfade: /tempZone2/home/irods/StorageResc2/bigfile.big.gz
– Neue physikalische Pfade: /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/bigfile.big.gz
– Neue Name: bigfile.big.gz
– Neue DATA ID: 182.965
– Neue Größe: 3.175 Bytes
Moment geschaffte Speicherkapazität 1.569.678 Bytes
```

Das Ergebnis ist dass die physikalische Datei komprimiert wurden. Sie befindet sich in gleichem Ort wie vorher und wird im neuen Komprimierungsformat umgewandelt (.gz).

Nach der Komprimierung muss man die physikalische Datei mit neuem Typ wieder ins iCAT Metadaten registrieren. Es existiert noch die Informationen (Metadaten Name, Format) über die alte logische Name (bigfile.big) im iCAT. Eine überschriebene Registrierung von neuem Datentyp auf altes Datentyp oder Trennung von altem Verweis der Datei (eng. unlink) ist bei der angewendeter Version iRODS 3.1 nicht möglich. Dafür braucht man die Registrierung von Datei mit neuem physikalischen Namen und neuem logischen Name ins iCAT. Der alte logische Name ist in diesem Fall noch da, erscheint als Fehler weil die alte physikalische Name nicht mehr existiert.

Die neue Version iRODS 3.2 *Micro-Service msiDataObjUnlink mit Option unreg* kann den alten Verweis auf logische Name im iCAT entfernen, ermöglicht die neue Registrierung mit gleichen Name wieder ins iCAT.

Das Ergebnis der Komprimierung ist oft positiv, dh. die neue Größe der Daten nach der Komprimierung ist kleiner als die originale Größe der Daten vor der Komprimierung. Die alte Größe der Datei reduziert von 1572853 Bytes (1,5 Megabytes) noch auf 3175 Bytes. Die folgende Rechnung zeigt die Komprimierungsverhältnis

$$\text{Komprimierung Faktor[13]} = \frac{DieGroessevorderKomprimierung}{DieGroessenachderKomprimierung} = \frac{1.572.853}{3.175} \sim 496$$

Die Komprimierung hat die originale Größe der Datei fast 496 mal reduziert, und damit gewinnt der Wissenschaftler **1.569.678** Bytes $\sim$ 1,5 MB freie Speicherkapazität.

2.Demo: Komprimieren mehrerer Dateien
Die automatische gleichzeitige Bearbeitung von mehreren Dateien wäre die bessere Lösung und bringt freie Speicherkapazität. Nach dem Ablauf des Programms bekommt der Benutzer die freie Speicherkapazität aus den gesamten bearbeiteten Daten. Der geschaffte Speicherplatz wird gleich der Differenz der Größe der Summe aller bearbeiteter Daten (z.B. in allen Unterverzeichnisse von Storage2) vor und nach der Komprimierung sein. Die Komprimierung wird auf dem Verzeichnis Storage2 von StorageResc2 angewendet.

Dabei ist eine aufgenommene Test-logdatei von abgelaufenem Programm (vgl. Quelltext 6.2):

```
Name : storage2-1.st1
– iRODS Pfade: /tempZone2/home/irods/StorageResc2/Storage2/storage2-1.st1
– /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/Storage2/storage2-1.st1
– Größe: 57 Bytes
– DATA ID: 266629
Komprimierung erfolgreich
********** Neue Zip-Datei ermitteln ************
– Neue iRODS Pfade: /tempZone2/home/irods/StorageResc2/Storage2/storage2-1.st1.gz
– Neue physikalische Pfade: /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/Storage2/storage2-
1.st1.gz
– Neue Name: storage2-1.st1.gz
– Neue DATA ID: 268353
– Neue Größe: 79 Bytes
Moment geschaffte Speicherkapazität: -22 Bytes
Insgesamt Gewinn freie Speicherkapazität: -22 Bytes


Name : storage2-10.st10
– iRODS Pfade: /tempZone2/home/irods/StorageResc2/Storage2/storage2-10.st10
– /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/Storage2/storage2-10.st10
```

– Größe: 453 Bytes
– DATA ID: 266815
Komprimierung erfolgreich
********** Neue Zip-Datei ermitteln ***********
– Neue iRODS Pfade: /tempZone2/home/irods/StorageResc2/Storage2/storage2-10.st10.gz
– Neue physikalische Pfade: /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/Storage2/storage2-10.st10.gz
– Neue Name: storage2-10.st10.gz
– Neue DATA ID: 268354
– Neue Größe: 86 Bytes
Moment geschaffte Speicherkapazität: 367 Bytes
Insgesamt Gewinn freie Speicherkapazität: 345 Bytes

...

Name : storage2-99.st99
– iRODS Pfade: /tempZone2/home/irods/StorageResc2/Storage2/storage2-99.st99
– /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/Storage2/storage2-99.st99
– Größe: 4.369 Bytes
– DATA ID: 266705
Komprimierung erfolgreich
********** Neue Zip-Datei ermitteln ***********
– Neue iRODS Pfade: /tempZone2/home/irods/StorageResc2/Storage2/storage2-99.st99.gz
– Neue physikalische Pfade: /home/irobot/Desktop/iRODS/Vault/home/irods/StorageResc2/Storage2/storage2-99.st99.gz
– Neue Name: storage2-99.st99.gz
– Neue DATA ID: 268852
– Neue Größe: 116 Bytes
Moment geschaffte Speicherkapazität: 4.253 Bytes
Insgesamt Gewinn freie Speicherkapazität: 5.450.162 Bytes
Anzahl der komprimierte Dateien : 500
Operation erledigt

Nach dem Ablauf von Programm wurden 500 Dateien von Verzeichnis Storage2 komprimiert und der Benutzer gewinnt in diesem Fall **5.450.162 Bytes** $\sim$ 5,4 MB freie Speicherkapazität. Auffällig ist das Ergebnis bei erster Datei, der Komprimierungsfaktor stellt eine negative Leistung (dh. die Größe der Datei nach der Komprimierung ist noch größer als vorher) dar.

Die Implementierung mit Bash Shell 6.8 unterscheidet sich von der iRODS-Implementierung bei der Eingabe von absoluter Adresse von Start-Verzeichnis im Dateisystem. Jedoch liefern beide Implementierungen gleiche Ergebnisse.

7.1.2 Löschen

3.Demo: Löschen die Dateien mit Replikation
Die Größe der gewonnene Speicherkapazität ist abhängig von der angewendeten Operation. Bei der Komprimierung ist die Leistung vom Gewinn freier Speicherkapazität oft abhängig

von Datenformat, Komprimierungsprogramm und die geschaffte Speicherkapazität ist für den Speicherbedarf vom Benutzer nicht ausreichend. Durch das Löschen von Daten kann man mehr Speicherplatz gewinnen als durch Komprimieren aber man verliert die Daten falls die Daten keine Kopie haben.

In diesem Fall müssen die Dateien in StorageResc2, die Replikation in anderen Ressourcen GridResc2 und FTPResc2 haben, gelöscht werden um die freie Speicherkapazität in StorageResc2 zu schaffen. Dabei ist eine aufgenommene Test-logdatei von abgelaufenem Programm (vgl. Quelltext 6.4)

```
Datei mit Name storage3-1.txt hat eine Kopie in GridResc2
– Die Datei storage3-1.txt in StorageResc2 wird gelöscht
– Gewinn freie Speicherkapazität: 56 Bytes

...

Datei mit Name storage3-40.txt hat eine Kopie in FTPResc2
– Die Datei storage3-40.txt in StorageResc2 wird gelöscht
– Gewinn freie Speicherkapazität: 1.733 Bytes

Anzahl der gelöschter Dateien: 40
Insgesamt Gewinn freie Speicherkapazität: 35.780 Bytes
```

Der Wissenschaftler gewinnt in diesem Fall 35780 Bytes $\sim$ 35 KB mit insgesamt 40 Dateien, die gelöscht werden. Die Dateien wurden komplett in StorageResc2 permanent gelöscht aber der Wissenschaftler haben immer die Möglichkeit auf die Replikation der gelöschter Dateien in anderen Ressourcen (GridResc2, FTPResc2) zuzugreifen.

7.1.3 Verschieben

4.Demo: auf Multi-Ressource mit Quota
Die Komprimierung von Daten sowie das Löschen von Replikation können die Speicheranforderung an einem Ort nicht erfüllen, besonders wenn die Komprimierungsmethode wenig freie Speicherplätze schafft oder nur wenige Dateien eine Replikation woanders besitzen. In dem Fall würde der Wissenschaftler zu anderen Kontext wechseln, (eine / mehrere) Dateien nach einem anderem Ressource zu verschieben, um die freie Speicherkapazität weiter zu schaffen.

Die Speicherkapazität von einer Ressource wird in diesem Fall von Quota als Limit der Speichern von Daten beschränkt, und nach der Angabe der verfügbare freie Quota (GridResc2 hat noch $\sim$ 7,9 MB und FTPResc2 hat noch 6 MB), reicht eine Ressource nicht für den Rest der Speicheranforderung aber die Kombination von beider Ressourcen sind ausreichend. Dabei ist eine aufgenommene Test-log von abgelaufenem Programm (vgl. Quelltext 6.6)

Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-1000.1000 ist schon an StorageResource FTPResc2 verschoben
– Größe: 52.013 Bytes
– Gewinn: 52.013 Bytes
– TargetResource1 noch : 7.861.647 Bytes

...
Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-835.835 ist schon an StorageResource FTPResc2 verschoben
– Größe: 43.433 Bytes
– Gewinn: 7.922.018 Bytes
– **TargetResource1 noch : -8.358** Bytes
Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-834.834 ist schon an StorageResource GridResc2 verschoben
– Größe: 43.381 Bytes
– Gewinn: 7.965.399 Bytes
– TargetResource2 noch : 6.056.002 Bytes

...
Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-680.680 ist schon an StorageResource GridResc2 verschoben
– Größe: 35.373 Bytes
– Gewinn: 14.025.453 Bytes
– **TargetResource2 noch : -4.052 Bytes**
die freie Speicherkapazität **14.025.453** Bytes wurde geschafft
insgesamt : 7.922.018 Bytes an FTPResc2 verschoben
insgesamt : 6.103.435 Bytes an GridResc2 verschoben
die Anzahl der verschobene Dateien an FTPResc2 ist 166, an GridResc2 ist 155

Insgesamt gewinnt der Wissenschaftler aus 4 Verfahren : 1.569.678 Bytes(Komprimieren einzelner Daten) + 5.450.162 Bytes(Komprimieren mehrere Daten) + 35.780 MB(Löschen Replikation) + 14.025.453 Bytes(Verschieben)= 21.081.073 Bytes $\sim$ 21 MB $>$ Speicheranforderung 20.000.000 MB. Der Auftrag ist erfüllt. Eine Beobachtung zu oberer log-Datei zeigt dass die verfügbare Quotas in Ressourcen (GridResc2, FTPResc2) negativ ist, dh. die verfügbare Quotas sind momentan überschritten *(siehe Verbesserung 7.2.2)*

7.2 Bewertung auf Performance

Der Performance Test konzentriert sich auf die Aspekte Effizienz vom Kriterien 3.2 mit der Implementierungen iRODS-Komprimierung 6.3 und Shell-Komprimierung 6.9 (mit-/ohne Sortierung), iRODS-Verschieben 4.3.2 (mit/ohne Sortierung) ein. Dabei werden die folgenden Aspekte für die Bestimmung der Effizienz aus Kriterien wiederholt:

- Aufwand des Algorithmen im Verfahren und Implementierung bzgl. die freie geschaffte Speicherkapazität.
- Anzahl der bearbeiteten Daten, beeinflusst auf die Aufwände von Rechenkapazität.
- Die reale Laufzeit (die gesamte Laufzeit des Befehls mit implementierter Methode).

Die folgende aufgelistete Soll-Ergebnisse evaluiert die Effizienz der Verbesserungsansätze (Berücksichtigung des Speicherbedarfs) 4.1.2, (Ergänzung um die Sortierung) 4.1.3 über den Basis-Ansatz (Komprimieren alle Dateien) 4.1.1 und Verbesserungsansatz bzgl. Vermeiden das Überschritten von Quota (siehe 4.Demo 7.1.3), das von Basis-Ansatz (Verschieben) 6 verursacht wurde:

P1 Es soll nicht mehr Speicherplatz freigegeben werden als nötig.

P2 Die Anzahl der bearbeitete Daten sowie die reale Laufzeit des Programms soll effizient reduziert werden.

P3 Die Verteilung der verschobene Dateien müssen die freie verfügbare Quotas in anderen Ziel-Ressourcen für Speichern ausnutzen. Aber das Speicher-Überschritten im Quota darf wegen Verschieben nicht passieren.

7.2.1 Komprimierung

1.Testfall: Komprimierung mit benutzerdefinierter Speicheranforderung
Die Operation wird auf Ressource FTPResc2 (Anzahl der Dateien 700) gemessen. Die folgende Tabelle (vgl. 7.1)zeigt sich die Effizienz zwischen den Ansatz mit vordefinierter Speicheranforderung und den Basis-Ansatz.

Umgebung/ Implementierung	geschaffene Kapazität (Bytes)	bearbeitete Dateien	Laufzeit (s)
iRODS			
Basisansatz (A 5, I 6.2)	2 004 834	700	36, 87
Erweiterung (A 2, I 6.3)	1 502 340	551	27, 89
Bash Shell			
Basisansatz (A 5, I 6.8)	2 004 834	700	6, 64
Erweiterung (A 2, I 6.9)	1 504 151	494	4, 49

Tabelle 7.1: Vergleich der Effizienz von Implementierungen für die Komprimierung von Dateien bei einer Speicheranforderung von $1,5 \times 10^6$ Bytes. Die Verweise in der Spalte Implementierung beziehen sich auf die Algorithmen (A) und Implementierungen (I).

Die SQL-Abfrage in iRODS mit Dateiname (Quelltext 6.2, 6.3) ermöglicht die alphabetische Sortierung der zurückgegebener Reihenfolge der Dateien nach ihrem Namen. Die Reihenfolge der zurückgegebene Dateien beeinflussen das letztendliche Ergebnis, weil sie mit der Größe der Datei zusammenhängen, und die Größe der Datei entscheidet die geschaffte Speicherkapazität, besonders bei dem Ansatz unter Berücksichtigung des Speicherbedarfs. In diesem Fall hat das Ergebnis gezeigt, dass die vordefinierte Speicheranforderung aus Benutzer die Anzahl der bearbeiteter Daten (nur 551/700) beschränkt hat. Der Basis-Ansatz ohne vordefinierte Speicheranforderung liefert anstatt immer am höchsten Anzahl der Dateien 700 mit groß möglichst freier Speicherkapazität.

Beim Basis-Ansatz könnte die Rechenkapazität reduziert werden, weil es keine Vergleich-Bedingung zwischen die geschaffte Speicherkapazität und die Speicheranforderungen während der Datenverarbeitung gibt. Aber die reale Laufzeit ist insgesamt jedoch noch deutlich höher als erweiterter-Ansatz, weil alle Dateien bearbeitet (komprimiert) werden müssen.

Vergleich in der Tabellen von iRODS und Bash-Shell zeigt die unterschiedliche Ergebnisse, besonders bei der reale Laufzeit. Es ist abhängig davon wie die Daten abgefragt und zurückgegeben werden (iRODs per SQL von Datenbank) und (Shell per Befehl find von Unix/Linux-Dateisystem). Insgesamt haben beide Implementierungen mit erweitertem Ansatz die Effizienz über Basis-Ansatz nachgewiesen (vgl. 7.2).

2.Testfall: Komprimierung mit benutzerdefinierter Speicheranforderung auf sortierte Daten
Die folgende Tabelle zeigt sich den Unterschied zwischen mit/ohne Sortierung-Funktion bei den Implementierungen mit iRODS und Bash Shell.

Umgebung/ Implementierung	geschaffene Kapazität (Bytes)	bearbeitete Dateien	Laufzeit (s)
iRODS			
Ohne Sortierung (A 2, I 6.3)	1 502 340	551	27, 89
Mit Sortierung (A 3, I 6.1.2)	1 502 347	345	27, 31
Bash Shell			
Ohne Sortierung (A 2, I 6.9)	1 504 151	494	4, 49
Mit Sortierung (A 3, I 6.2.2)	1 502 347	345	4, 94

Tabelle 7.2: Vergleich der Effizienz von Implementierungen für die Komprimierung von Dateien bei einer Speicheranforderung von $1,5 \times 10^6$ Bytes mit zusätzlicher Sortierung.

Das Ergebnis hat gezeigt, dass die Anzahl der bearbeitete Dateien durch die Sortierung nach der Größe sehr reduziert wurden (vgl. P2 7.2). Es könnte auch passieren, dass die freie geschaffene Speicherkapazität mit Sortierung viel größer als die Methode ohne Sortierung ist. Der Grund davon ist abhängig von den bearbeiteten Daten. Die größte Datei wird zuerst bearbeitet und die geschaffene Speicherkapazität sind abhängig von den bearbeiteter Daten (Format, Größe) und den Kompress-Algorithmen (gzip).

Außerdem zeigen die Ergebnisse mit Sortierung von beiden Implementierung Bash Shell und iRODS identisch bei den geschaffene Speicherkapazität (1.502.347), und Anzahl der bearbeitete Dateien (345). Weil der Arbeitsvorgang immer sowohl bei iRODS als auch Shell zuerst auf die Dateien mit höchster Größe beginnt und dann absteigend bearbeitet.

7.2.2 Verschieben

3.Testfall: Verschieben von Daten im Ressource mit von Quota beschränkte Speicherkapazität
Ein Rückblick auf 7.1.3 erkennt man die negative Speicherkapazität beim Ziel-Ressourcen. Da zeigt sich, dass die maximale Speicherkapazität Quota während der Datenübertragung überschritten wird. Jedoch wird das System in diesem Fall so konfiguriert, dass die Datenübertragung nicht unterbrochen wird und die komplette erfolgreiche Datenübertragung von Quell-Ressource an Ziel-Ressource falls Quota überschritten ist. Anders wird die Datenübertragung unterbrochen bzw. abgebrochen und nur ein Teil von Datei fertig übertragen wird. Aber die gesamte Daten wird unvollständig übertragen und unbrauchbar betrachten. Die weitere Übertragung danach auf dem überschrittenem Quota Speicherraum ist nicht mehr möglich. Der Grund, warum Quota überschritten ist, liegt an die Größe der verschickten Daten und die verfügbare freie Speicherraum von Ziel-Ressourcen, die nicht immer übereinstimmen könnten.

Lösung mit optimierter Bedingung:
Die Änderung in der Überprüfung-Bedingung von Verfahren 4.3.2 mit der Implementierung 6.1.5 gewährleistet die Vermeidung von Überschritten in Quota, und optimierte das Verteilen von Daten von einem Quell-Ressourcen an mehreren Ziel-Ressourcen.

Das Ergebnis von zusammengefasster log-Dateien des abgelaufen Programm mit optimierter Bedingung:

Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-1000.1000 ist schon an StorageResource FTPResc2 verschoben
– Größe: 52.013 Bytes
– Gewinn: 52.013 Bytes
– TargetResource1 noch: 7.861.647 Bytes
...

Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-836.836 ist schon an StorageResource FTPResc2 verschoben
– Größe: 43.485 Bytes
– Gewinn: 7.878.585 Bytes
– **TargetResource1 noch: 35.075 Bytes**

Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-835.835 ist schon an StorageResource GridResc2 verschoben
– **Größe: 43.433** Bytes
– Gewinn: 7.922.018 Bytes
– TargetResource2 noch: 6.055.950 Bytes

...
Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-682.682 ist schon an StorageResource GridResc2 verschoben
– Größe: 35.477 Bytes
– Gewinn: 13.954.655 Bytes
– **TargetResource2 noch: 23.313** Bytes

Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-674.674 ist schon an StorageResource FTPResc2 verschoben

– **Größe: 35.061** Bytes
– Gewinn: 13.989.716 Bytes
–**TargetResource1 noch: 14** Bytes

Die Datei /tempZone2/home/irods/StorageResc2/Storage1/Archiv2/archiv2-448.448 ist schon an StorageResource GridResc2 verschoben
– **Größe: 23.309** Bytes
– Gewinn: 14.013.025 Bytes
– **TargetResource2 noch: 4** Bytes

die freie Speicherkapazität 14.013.025 Bytes wurde geschafft

insgesamt : 7.913.646 Bytes an FTPResc2 verschoben

insgesamt : 6.099.379 Bytes an GridResc2 verschoben

die Anzahl der verschobener Daten an FTPResc2 ist 166, an GridResc2 ist 155

Das Überschritten von Quota in diesem Fall wurde vermeidet. Die Prüfungsbedingung wurde optimiert, dass die Datenübertragung nur stattfindet wenn die Größe von verschiebenden Daten von Quell-Ressource (StorageResc2) kleiner als die verfügbare freie Speicherkapazität von Ziel-Ressourcen (FTPResc2, GridResc2). Beobachtung auf log-Datei erkennt man dass das Verschieben von Daten abwechseln zwischen zwei Ziel-Ressourcen nach verfügbare Quota dynamisch geändert wird, um die freie vorhandene Speicherkapazität Quotas am Ziel-Ressourcen auszunutzen. Wenn die Größe der Datei größer als verfügbare Quota am Ziel-Ressource ist, wird die Datei an anderen Ziel-Ressource verschoben.

Untere Tabelle zeigt den Vergleich zwischen beiden Verfahren. Die Anzahl der verschobener Dateien sind nicht geändert worden obwohl beide Ziel-Ressourcen nicht überschritten wurde und die geschaffte Speicherkapazitäten auch optimierter (nah der Speicheranforderung) geworden sind:

Umgebung/ Implementierung	geschaffene Kapazität (Bytes)	bearbeitete Dateien	Gesamte Laufzeit (s)
iRODS (A 6, I 6.7)			
Quell-Ressource	14 025 453	321	14, 99
1-Ziel-Ressource	7 922 018	166	
2-Ziel-Ressource	6 103 435	155	
iRODS neue If-Bedingung (A 4.3.2, I 6.1.5)			
Quell-Ressource	14 013 025	321	16, 85
1-Ziel-Ressource	7 913 646	166	
2-Ziel-Ressource	6 099 379	155	

Tabelle 7.3: Vergleich der Quota Zustand mit/ohne Änderung der Bedingung bei einer Speicheranforderung von $1,4 \times 10^6$ Bytes. Der Test wird auf die sortierte nach Größe sortierten Dateien angewendet

8 Fazit

Ziel dieser Arbeit war es, die Speicherkapazität in wissenschaftlicher Umgebung zu verwalten bzw. die Ausschöpfung der Speicherkapazität zu vermeiden. Um das Ziel zu erreichen, müssen die grundlegenden Operationen (Komprimieren, Löschen, Verschieben) und zugehörige Ansätze angewendet werden, dadurch werden die Daten ohne Verlust verändert, oder die Anzahl der gespeicherte Daten reduziert oder umgeordnet um neue freie Speicherkapazität für neue Daten zu schaffen.

Dabei wurde zunächst in Kapitel 2 die Fallbeispiele über bekannte Projekte wie CERN, C3Grid, Glues beobachtet um eine Überblick über Datenmanagement und Probleme zu bestimmen. Danach wird in Kapitel 3 die konkreten Probleme bezüglich dem Ziel der Arbeit geklärt und analysiert. Die Anforderungen bzgl. angestrebter Veränderung des Systems und die Kriterien wurden dabei zusammengefasst, die auf die nächste entwickelte Verfahren und Implementierung bei nächsten Kapitel orientieren.

In Kapitel 4 wurden die Verfahren von Basis bis zu erweiterten Maßnahmen entworfen. Die Basis-Ansätze konzentrieren auf die Einfachheit mit weniger benötigte Informationen und die erweiterten Ansätze entwickeln die Basis-Ansätze weiter durch die Betrachtungen weiterer Aspekte (Speicheranforderung, Sortierung, Replikation, Multi-Ressourcen). Kapitel 5 öffnet sich einen Auswahl zu heutigen Technologien mit dem Ziel welche Technologie geeignet ist für die Implementierung von Kapitel 4 sowie die Probleme, Anforderung und Kriterien in Kapitel 3. Dabei wurden viele Technologie im Ausblick beobachtet (Dropbox, dCache, OGSA-DAI, Cloud Amazon S3, Grid Globus Toolkit, iRODS, Shell). Data Grid iRODS und Bash-Shell wurden für die Implementierung ausgewählt.

In Kapitel 6 wurden die Verfahren mithilfe der ausgewählten Technologien iRODS und Bash Shell implementiert. Abschließend demonstriert Kapitel 7 mit der Implementierungen in Kapitel 6 in einer Situation, wie ein Wissenschaftler die freie Speicherkapazität nach seiner Anforderung Schritt für Schritt schaffen kann. Danach wird die Performance, Effizienz von Verfahren und Implementierung bewertet um die erwähnte Kriterien aus Kapitel 3 hervorzuheben. Die Arbeit ermöglicht noch weitere Ideen sowie offene Aspekte als vorhandene Herausforderungen in wissenschaftlicher Daten-intensive Umgebung und die Betrachtung auf weitere Verbesserung bzw. Nachbesserung in Zukunft.

Literaturverzeichnis

[1] Abadi, Daniel J.: *Data Management in the Cloud: Limitations and Opportunities.* Technischer Bericht, Yale University New Haven, CT, USA, 2009.

[2] Allcock, W., A. Chervenak, I. Foster, L. Pearlman, V. Welch und M. Wilde: *Globus Toolkit Support for Distributed Data-Intensive Science.* In: *in International Conference on Computing in High Energy and Nuclear Physics*, 2001.

[3] Antonioletti, Mario, Malcolm Atkinson, Rob Baxter, Andrew Borley, Neil Chue Hong, Brian Collins, Jonathan Davies, Neil Hardman, George Hicken, Ally Hume, Mike Jackson, Amrey Krause, Simon Laws, James Magowan, Jeremy Nowell, Norman W. Paton, Dave Pearson, Tom Sugden, Paul Watson und Martin Westhead: *OGSA-DAI: Two Years On.* In: *GGF10*, 2004.

[4] Branco, Miguel, Ed Zaluska, David de Roure, Mario Lassnig und Vincent Garonne: *Managing very large distributed datasets on a Data Grid.* Concurrency and Computation: Practice & Experience, 22:1338–1364, 2009.

[5] Brunner, Robert: *Working in the Bash shell An introduction.* NCSA Research Scientist, Assistant Professor of Astronomy, University of Illinois, Urbana-Champaign, May 2006.

[6] Dobrzelecki, Bartosz, Amrey Krause, Alastair C. Hume, Alistair Grant, Mario Antonioletti, Tilaye Y. Alemu, Malcolm Atkinson, Mike Jackson und Elias Theocharopoulos: *Integrating distributed data sources with OGSA–DAI DQP and Views.* Phil. Trans. R. Soc., 368:4133–4145, 2010.

[7] Foster, Ian: *What is the Grid? - a three point checklist.* GRIDtoday, 1, 2002.

[8] Foster, Ian: *Globus toolkit version 4: software for service-oriented systems.* In: *Proceedings of the 2005 IFIP international conference on Network and Parallel Computing*, 2005.

[9] Fuhrmann, Patrick: *dCache, the Overview.* http://www.dcache.org/manuals/dcache-whitepaper-light.pdf.

[10] Gottloeber, Stefan, Yehuda Hoffman und Gustavo Yepes: *Constrained Local UniversE Simulations (CLUES).* In: *High Performance Computing in Science and Engineering, Garching/Munich 2009*, Seiten 309 – 322, 2010.

[11] Gray, Jim, Liu David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt und Gerd Heber: *Scientific data management in the coming decade.* SIGMOD Rec., 34:34–41, January 2005.

[12] Kindermann, S., F. Schintke, B. Fritzsch und C3 Team: *A Collaborative Data Management Infrastructure for Climate Data Analysis.* Geophysical Research Abstracts, 14 (EGU2012), p. 10569, 2012. C3-Grid Poster.

[13] KODITUWAKKU, S.R. und U. S.AMARASINGHE: *COMPARISON OF LOSSLESS DATA COMPRESSION ALGORITHMS FOR TEXT DATA*. Indian Journal of Computer Science and Engineering, 1:416–425, 2010.

[14] Lamanna, Massimo: *The LHC computing grid project at CERN.* Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 534:1 – 6, 2004.

[15] Moore, Reagan: *Towards a Theory of Digital Preservation.* The International Journal of Digital Curation, 3:63–75, 2008.

[16] Palankar, Mayur, Adriana Iamnitchi, Matei Ripeanu und Simson Garfinkel: *Amazon S3 for science grids: a viable solution?* In: *Proceedings of the 2008 international workshop on Data-aware distributed computing*, 2008.

[17] Rajasekar, Arcot, Reagan Moore, Michael Wan und Wayne Schroeder: *Universal view and open policy: Paradigms for collaboration in data grids.* In: *Collaborative Technologies and Systems, 2009. CTS '09. International Symposium on*, 2009.

[18] Rajasekar, Arcot, Michael Wan, Reagan Moore, Wayne Schroeder, Sheau-Yen Chen, Lucas Gilbert, Chien-Yi Hou, Christopher A. Lee, Richard Marciano, Paul Tooby, Antoine de Torcy, Bing Zhu und Gary Marchionini: *iRODS Primer Integrated Rule-Oriented Data System.* Morgan & Claypool, 2010.

[19] Ramey, Chet und Brian Fox: *Bash Reference Manual.* Case Western Reserve University and Free Software Foundation, 4.2 Auflage, December 2010.

[20] Röblitz, Thomas, Harry Enke, Kristin Riebe, Bernadette Fritzsch und Jens Klump: *Vision of a Virtual Infrastructure for Storing and Processing Scientific Data.* Technischer Bericht, Chair Service Computing TU Dortmund University Germany and Leibniz-Institut für Astrophysik Potsdam (AIP) Germany and Alfred Wegener Institute for Polar and Marine Research Germany and Helmholtzzentrum Potsdam – German Research Center for Geosciences Germany, 2011.

[21] Scientific Data, High Level Expert Group on: *Riding the wave — How Europe can gain from the rising tide of scientific data.* A report to the European Commission, Oktober 2010.

[22] Ulbrich, U., H. Kupfer, I. Kirchner, T. Brücher, M. Stockhause, B. Fritzsch, K. Fieg und C. Kurz: *C3-Grid - ein Werkzeug für die Klimaforschung.* In: *DACH2007, Meteorologentagung*, 2007.

[23] Vinge, Vernor: *2020 Computing: The creativity machine.* Nature, 440:411, 2006.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 23. November 2012

Long Duc Phan