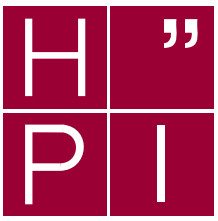
HASSO-PLATTNER-INSTITUT
für Softwaresystemtechnik an der Universität Potsdam



Ressourcenpartitionierung für Grid-Systeme

Matthias Lendholt

Technische Berichte Nr. 8
des Hasso-Plattner-Instituts
für Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik
an der Universität Potsdam

Nr. 8

Ressourcenpartitionierung für Grid-Systeme

Matthias Lendholt

Potsdam 2005

Bibliografische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Die Reihe *Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam* erscheint aperiodisch.

Herausgeber: Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam

Redaktion: Matthias Lendholt
Email: matthias.lendholt@gmx.de

Editor: Peter Tröger
Email: peter.troeger@hpi.uni-potsdam.de

Vertrieb: Universitätsverlag Potsdam
Postfach 60 15 53
14415 Potsdam
Fon +49 (0) 331 977 4517
Fax +49 (0) 331 977 4625
e-mail: ubpub@rz.uni-potsdam.de
http://info.ub.uni-potsdam.de/verlag.htm

Druck: allprintmedia gmbH
Blomberger Weg 6a
13437 Berlin
email: info@allprint-media.de

© Hasso-Plattner-Institut für Softwaresystemtechnik an der Universität Potsdam, 2005

Dieses Manuskript ist urheberrechtlich geschützt. Es darf ohne vorherige Genehmigung der Herausgeber nicht vervielfältigt werden.

Heft 8 (2005)
ISBN 3-937786-72-4
ISSN 1613-5652

Ressourcenpartitionierung für Grid-Systeme

matthias.lendholt@gmx.de

Grid Computing Frameworks werden im Allgemeinen über ihre Anwendungsschwerpunkte in *Data Grids*, *Computational Grids* und *Resource Grids* kategorisiert. Unter Betrachtung der den Frameworks zugrunde liegenden und von ihnen verwendeten Computersystemen, können *Computational Grids* in die beiden Bereiche *Cluster Computing* und *Idle Time Computing* differenziert werden. Während beim *Cluster Computing* dedizierte Rechnersysteme für *Grid Computing* verwendet werden, sind es beim *Idle Time Computing* ungenutzte Ressourcen auf Arbeitsplatzrechnern mit Desktopbetriebssystemen, die für *Grid Computing* herangezogen werden. Ein faires Nebeneinander von Jobs des Grid-Computings sowie der Tätigkeiten des Anwenders ist in *Idle Time Computing* nur dann gegeben, wenn sichergestellt wird, dass eine gerechte Ressourcenaufteilung in allen Situationen erfolgt, bei der sowohl die Interessen des lokalen Anwenders als auch des Grid-Computing-Jobs gewahrt werden. Zur effizienten Steuerung der Ressourcenvergabe ist ein Framework erforderlich, das eine zentrale Kontrolle über die am *Idle Time Computing* partizipierenden Knoten ermöglicht. Der Zugriff erfolgt dabei sowohl von administrativer Seite, um beschränkende Maßnahmen durchzuführen, als auch durch das Grid-System um Mindestgarantien sicherzustellen.

Diese Arbeit beschäftigt sich mit der Entwicklung eines derartigen Frameworks, um auf heterogenen Arbeitsplatzrechnern eine zentrale Ressourcenkontrolle zu ermöglichen.

Inhaltsverzeichnis

Inhaltsverzeichnis 2
Abbildungsverzeichnis 3
Tabellenverzeichnis 4
Abkürzungsverzeichnis 5
1 Einleitung 7
1.1 Motivation 7
1.2 Aufgabenstellung 7
1.3 Aufbau der Arbeit 8
2 Fachliches Umfeld 9
2.1 Grid Computing 9
2.2 Idle Time Computing 11
3 Ressourcenpartitionierung 13
3.1 Ressourcenpartitionierung und Scheduling 13
3.2 Ressourcenpartitionierung in Idle Time Computing 13
3.2.1 Administrative Kontrolle der Ressourcen 13
3.2.2 Mindestgarantien zur Sicherstellung der Lauffähigkeit 16
3.3 Vorhandene Lösungen und verwandte Arbeiten 17
3.3.3 Ressourcenmanagement im Globus Toolkit 18
3.3.4 Ressourcenmanagement in Condor 19
3.3.5 Das GARA Framework 20
3.3.6 Distributed Resource Management Application API 21
3.3.7 Grid Resource Allocation Agreement Protocol 21
3.3.8 SNAP 21
3.3.9 OGSI-Agreement 22
3.3.10 PBS/Maui und andere Batch Systeme 22
3.3.11 Wide Area Distributed Computing 22
3.3.12 Weitere verwandte Arbeiten 22
3.3.13 Überlegungen zum Ressourcenbedarf 23
4 Ressourcenkontrolle in Desktopbetriebssystemen 25
4.1 Ressourcen 25
4.2 Ressourcenpartitionierung unter Linux 28
4.2.14 Festplattenkapazität 28
4.2.15 Arbeitsspeicher 29
4.2.16 Netzwerkbandbreite 30
4.2.17 CPU Kapazität 31
4.3 Ressourcenpartitionierung unter Microsoft Windows XP 34
4.3.18 Festplattenkapazität 34
4.3.19 Arbeitsspeicher 34
4.3.20 Netzwerkbandbreite 35
4.3.21 CPU Kapazität 35
4.4 Fazit 35
5 Entwurf und Design des Frameworks 37
5.1 Entwurfsgrundlage - Komponentenarchitektur 37
5.2 Architektur 38
5.2.22 Broker Server 39
5.2.23 Broker Software auf den Clients 40
5.2.24 Schnittstellen 41
5.3 Komponententypmodell 41
5.4 Ressourcenidentifikation und Ressourcenzuordnung 44

5.5 Anbindung ans Grid 47
5.6 Datenbankentwurf 48
5.7 XML RPC Schnittstelle 51
6 Implementierung 53
6.1 Broker Server 54
6.1.25 Datenbank 55
6.1.26 Komponenten 55
6.1.27 Webserverskripte 56
6.1.28 Database Crawler 56
6.2 Komponenten 60
6.2.29 Network 61
6.2.30 DSCP 61
6.2.31 Quota 62
6.2.32 FSF 62
6.3 Linux Client 63
6.3.33 Broker Client 65
6.3.34 Komponenten 66
6.4 Windows Client 67
6.5 Abhängigkeiten 68
7 Evaluierung 69
7.1 Durchgeführte Test- und Anwendungsfälle 69
7.1.35 Test der Clientsoftware 69
7.1.36 Test der Serversoftware 70
7.1.37 Durchführbare Anwendungsfälle 70
7.2 Evaluierung der Frameworkimplementierung 72
7.2.38 Performanz der Serverimplementierung 72
7.2.39 Zuverlässigkeit der Partitionierungen 73
7.3 Evaluierung von Traffic Shaping mit iproute2 74
7.3.40 Szenario 1 75
7.3.41 Szenario 2 77
7.3.42 Szenario 3 78
7.3.43 Fazit 81
8 Zusammenfassung und Ausblick 82
8.1 Fazit 82
8.2 Weiterentwicklung 82
8.2.44 Weiterentwicklungen auf Entwurfsebene 82
8.2.45 Weiterentwicklungen auf Implementierungsebene 84
Anhang A: Details zum Datenbankentwurf 85
Anhang B: Bildschirmfotos der Weboberfläche 88
Anhang C: Quellcode Linux Broker Client 92
Anhang D: Inhalt des beiliegenden Datenträgers 95
Quellenverzeichnis 96

Abbildungsverzeichnis

Abbildung 1: Abgrenzung von *Idle Time* -, *Cluster* - und *Grid Computing* 11
Abbildung 2: Schematisches Abarbeitungsmodell von *Grid Computing* 12
Abbildung 3: GRAM im Kontext globaler und lokaler Scheduler 18
Abbildung 4: Ressourcenkontrolle im GT3 19

Abbildung 5: Frameworkarchitektur 38
Abbildung 6: Komponententypmodell 42
Abbildung 7: Datenbankentwurf 49
Abbildung 8: Architektur der Implementierung 54
Abbildung 9: CGI Skripte und ihre Anwendungsfälle 56
Abbildung 10: Aktivitäten des Database Crawlers 58
Abbildung 11: Reservierungszustände 59
Abbildung 12: Zustände von Knoten und Reservierungen 60
Abbildung 13: Komponentenmodell des Linux Clients 64
Abbildung 14: Klassendiagramm des Linux Clients 65
Abbildung 15: Programmlauf mit Debugausgabe des Linux Broker Clients 66
Abbildung 16: Programmlauf mit Debugausgabe des Windows Clients 67
Abbildung 17: Verschiedene DSCPs in IP Headern 73
Abbildung 18: Zwei unpartitionierte, parallele Datenströme 74
Abbildung 19: Szenario 1 mit CBQ 75
Abbildung 20: Szenario 1 mit HTB 76
Abbildung 21: Szenario 2 mit CBQ 77
Abbildung 22: Szenario 2 mit HTB 77
Abbildung 23: Szenario 3 mit CBQ 79
Abbildung 24: Szenario 3 mit HTB 80
Abbildung 25: Bildschirmfoto – Startseite 88
Abbildung 26: Bildschirmfoto – Komponentendefinition 89
Abbildung 27: Bildschirmfoto – Neuen Knoten anlegen 89
Abbildung 28: Bildschirmfoto – Knotenliste des Wurzelknotens 90
Abbildung 29: Bildschirmfoto – Knotenliste 90
Abbildung 30: Bildschirmfoto – Reservierung anlegen 91
Abbildung 31: Bildschirmfoto – Detailansicht 92
Abbildung 32: Inhalt des beiliegenden Datenträgers 95

Tabellenverzeichnis

Tabelle 1: Betrachtung der Festplattenkapazitätstypen 27
Tabelle 2: Betrachtung der Arbeitsspeicherkapazitätstypen 27
Tabelle 3: Betrachtung der Netzwerkkapazitätstypen 27
Tabelle 4: Betrachtung der Prozessorkapazitätstypen 27
Tabelle 5: Ressourcenpartitionierungsmöglichkeiten unter Linux und Windows XP 36
Tabelle 6: Beschreibung der Datenbanktabellen 50
Tabelle 7: Interface BrokerClient 51
Tabelle 8: Beschreibung der Schnittstellenmethoden 51
Tabelle 9: Boolesche Zustandsvariablen 59
Tabelle 10: Parameter der Komponente Network 61

Tabelle 11: Reservierungsparameter der Komponente Network 61
Tabelle 12: Reservierungsparameter der Komponente DSCP 62
Tabelle 13: Parameter der Komponente Quota 62
Tabelle 14: Reservierungsparameter der Komponente Quota.....62
Tabelle 15: Parameter der Komponente FSF 63
Tabelle 16: Reservierungsparameter der Komponente FSF.....63
Tabelle 17: Testfälle des Linux-Clients 70
Tabelle 18: Testfälle des Windows-Clients.....70
Tabelle 19: Anwendungsfälle zur administrativen Beschränkung.....71
Tabelle 20: Anwendungsfälle zur Garantie von Mindestressourcen.....71
Tabelle 21: Datenbankdetails 88

Abkürzungsverzeichnis

API	Application Program(ming) Interface
CBQ	Class Based Queueing
CGI	Common Gateway Interface
CIDR	Classless Inter-Domain Routing
CORBA	Common Object Request Broker Architecture
CPAN	Comprehensive Perl Archive Network
CPU	Central Processing Unit
DBMS	Database Management System
DCOM	Distributed Component Object Model
DPSS	Distributed Parallel Storage System
DRMAA	Distributed Resource Management Application API
DRMS	Distributed Resource Management System
DSCP	Differentiated Services Code Point
DSRT	Dynamic Soft Real Time CPU Scheduler
ER	Entity Relationship
FK	Foreign Key
FLOPS	Floating-Point Operations Per Second
FMC	Fundamental Modelling Concepts
FSF	Fixed Swap File
GARA	{Globus / General purpose} Architecture for Reservation and Allocation
GGF	Global Grid Forum
GID	Group ID
GRAAP	Grid Resource Allocation Agreement Protocol
GRAM	Grid Resource Allocation and Management
GT3	Globus Toolkit 3
HTB	Hierarchical Token Bucket
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	Identification
IETF	Internet Engineering Task Force
IP	Internet Protocol
ITC	Idle Time Computing

LAN	Local Area Network
LARTC	Linux Advanced Routing and Traffic Control
LSF	Load Sharing Facility
MAC	Media Access Control
MB	Megabyte
MHE	Master Hosting Environment
MJS	Managed Job Service
MIPS	Million Instructions Per Second
MMJFS	Master Managed Job Factory Service
MOPS	Millions of Operations Per Second
MSDN	Microsoft Developer Network
NAT	Network Address Translation
NFS	Network File System
NTFS	New Technology File System
NTSS	NT Scheduling Server
NWS	Network Weather Service
OGSA	Open Grid Services Architecture
OGSI	Open Grid Services Infrastructure
PAM	Pluggable Authentication Modules
PBS	Portable Batch System
PID	Process ID
PK	Primary Key
QOS	Quality of Service
RPC	Remote Procedure Call
RRD	Round Robin Database
RSL	Resource Specification Language
RSVP	Resource Reservation Protocol
RT	Real-Time
SDK	Software Development Kit
SID	Session ID
SLA	Service Level Agreement
SNAP	Service Negotiation and Acquisition Protocol
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protokoll
SOSS	Solaris Scheduling Server
TOS	Type of Service
TC	Traffic Control
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UHE	User Hosting Environment
UID	User ID
UML	Unified Modelling Language
WADC	Wide Area Distributed Computing
WG	Working Group
WMI	Windows Management Instrumentation
WS	Web Service
WSDL	Web Services Definition Language
WSRF	Web Service Resource Framework
XML	eXtensible Markup Language

1 Einleitung

1.1 Motivation

Die der Arbeit zugrunde liegende Problematik ist die bisher unzureichend gelöste Frage der Ressourcenpartitionierung beim *Idle Time Computing*. Hierunter versteht man den Teilbereich von *Grid Computing*[1], bei dem keine dedizierte Clusterhard- und software sondern ungenutzte Ressourcenkapazitäten von Arbeitsplatzrechnern für das *Grid Computing* verwendet werden. Die normalerweise auf Arbeitsplatzrechnern eingesetzten Betriebssysteme sind nicht primär für verteiltes Rechnen entwickelt worden und es existieren nur begrenzte Möglichkeiten zur Ressourcenpartitionierung, d.h. dem Begrenzen aber auch dem Garantieren von Ressourcen. Während vor allem für große Forschungseinrichtungen und Institute das *Cluster Computing* im Rahmen von *Grid Computing* die vorherrschende Rolle spielt, so bietet sich für Universitäten und Unternehmen mit einer sehr großen Anzahl von Arbeitsplatzrechnern die Möglichkeit des Idle-Time-Computings, um ungenutzte Kapazitäten für ressourcenintensive Berechnungen zu verwenden. Der Begriff *Idle Time Computing* verdeutlicht bereits in sich die ersten Ansätze, die in dieser Richtung gemacht wurden, nämlich das Nutzen der Kapazitäten ausschließlich dann, wenn seitens des lokalen Anwenders keine Aktivität mehr durchgeführt wird. Aufgrund der stets steigenden Leistung der verwendeten Computerhardware ergibt sich, dass auch während Aktivität seitens des lokalen Anwenders besteht, große Teile der zur Verfügung stehenden Ressourcen ungenutzt bleiben, da die vom lokalen Anwender ausgeführten und benutzten Programme diese nicht benötigen. Es bietet sich daher an, Arbeitsplatzrechner nicht nur während vollständiger Leerlaufphasen, sondern auch während sie aktiv von einem lokalen Anwender verwendet werden, für *Idle Time Computing* zu nutzen. Dieses sollte jedoch nur geschehen, wenn zum einen der Schutz des lokalen Anwenders bzw. seiner Anwendungen vor den ressourcenintensiven Prozessen des Grid-Computings garantiert werden kann und wenn zum anderen aber auch die Möglichkeit besteht, dass sich Jobs im *Idle Time Computing* Mindestgarantien sichern können, um ihre eigene Lauffähigkeit zu garantieren. Beides, also sowohl die administrative Beschränkung des Ressourcenverbrauchs aber auch die Möglichkeit Mindestanforderungen zu garantieren, kann nur dann sinnvoll erfolgen, wenn hierfür ein geeignetes Framework zur netzweiten Ressourcenpartitionierung verwendet wird. Derartige Frameworks sind derzeit jedoch ausschließlich für spezielle Clusterhard- und software vorhanden.

1.2 Aufgabenstellung

Ziel der Arbeit ist der Entwurf eines Frameworks, das Ressourcenparititonierung im Rahmen von *Idle Time Computing* auf heterogenen Arbeitsplatzrechnern ermöglicht.

Im Kontext dieser Arbeit werden dabei folgende Punkte berücksichtigt:

[1] Eine ausführlichere Betrachtung von *Grid Computing* erfolgt in Kapitel 2.1 .

- Recherche und Analyse über vorhandene und verwandte Lösungen im Cluster- und Grid-Umfeld.
- Recherche, Analyse, Evaluierung und Testen von Mechanismen zur Ressourcenpartitionierung auf Arbeitsplatzbetriebssystemen.
- Entwurf eines Frameworks zur Ressourcenpartitionierung.
- Proof-of-concept Implementierung von Teilaspekten des entworfenen Frameworks.
- Evaluierung des implementierten Frameworks.

Beim Entwurf des Frameworks werden folgende Punkte beachtet:

- Berücksichtigung heterogener Betriebssystemlandschaften.
- Verwendung betriebssystemnaher Schnittstellen zur Ressourcenpartitionierung, um eine spätere Verwendung ohne die Installation zusätzlicher komplexer Softwarekomponenten zu ermöglichen.
- Berücksichtigung folgender Anforderungen:
    - Administrative Beschränkung der Ressourcenauslastung und des Ressourcenverbrauchs.
    - Ermöglichung von Mindestgarantien für Jobs des Idle-Time-Computings zur Sicherstellung ihrer Lauffähigkeit.
    - Koordination der Ressourcen-Partitionierung auf den einzelnen Knoten („Brokering“).

1.3 Aufbau der Arbeit

Der strukturelle Aufbau der Arbeit richtet sich nach den im vorigen Kapitel genannten Punkten. Kapitel 2 geht kurz auf das fachliche Umfeld der Arbeit ein, in Kapitel 3 wird sich ausführlich mit Ressourcenpartitionierung in Grid Computing, speziell in Bezug auf *Idle Time Computing*, befasst und es werden existierende Lösungen analysiert. In Kapitel 4 erfolgt eine Untersuchung, welche Ressourcenpartitionierungsmöglichkeiten seitens der Betriebssysteme Linux und Windows XP in Hinblick auf die Verwendbarkeit in einem Framework zur Ressourcenpartitionierung in Grid-Systemen bestehen. Der Entwurf eines solchen Frameworks erfolgt in Kapitel 5 , daran anschließend die prototypische Implementierung in Kapitel 6 und eine anschließende Evaluierung in Kapitel 7 . Zum Schluss erfolgt in Kapitel 8 ein Rückblick auf Geleistetes sowie eine Vorschau auf zukünftige Weiterentwicklungen des Frameworks.

2 Fachliches Umfeld

2.1 Grid Computing

Die hier vorgestellte Arbeit bewegt sich im Bereich des Grid-Computings, ein in den letzten Jahren sowohl in der Forschung als auch in der kommerziellen Welt sehr stark in den Vordergrund getretenes Thema. Der Begriff *Grid Computing* ist dabei nicht immer klar umrissen. Es existieren verschiedene Definitionen und Ansichten, auch geprägt durch Marketinginteressen diverser Firmen, die den Begriff auf eigene Produkte anwenden, obwohl diese wenig bis gar nichts mit der Begriffsdefinition von *Grid Computing* aus der Forscherwelt verbindet.

Ian Foster[2] definiert *Grid Computing* in „What is the Grid? A Three Point Checklist" ([10]) über die folgenden Anforderungen:

> *[…] a Grid is a system that:*
>
> 1. *coordinates resources that are not subject to centralized control ...[…]*
> 2. *... using standard, open, general-purpose protocols and interfaces ...[…]*
> 3. *... to deliver nontrivial qualities of service.[…]*

Rajkumar Buyya[3] beschreibt *Grid Computing* in „Grids and Grid technologies for wide-area distributed computing" ([02]) wie folgt:

> *Grids enable the sharing, selection, and aggregation of a wide variety of resources including supercomputers, storage systems, data sources, and specialized devices […] that are geographically distributed and owned by different organizations for solving large-scale computational and data intensive problems in science, engineering, and commerce. Thus creating virtual organizations […] and enterprises […] as a temporary alliance of enterprises or organizations that come together to share resources and skills, core competencies, or resources in order to better respond to business opportunities or large-scale application processing requirements, and whose cooperation is supported by computer networks. The concept of Grid computing started as a project to link geographically dispersed supercomputers, but now it has grown far beyond its original intent. The Grid infrastructure can benefit many applications, including collaborative engineering, data exploration, high-throughput computing, and distributed supercomputing.*

Sehr ausführlich wird das gesamte Themengebiet in *dem* Standardwerk zu *Grid Computing* aufgezeigt: Ian Foster, Carl Kesselman: The Grid, [12]. Während dieses Buch mehr eine Zusammenfassung der bereits vorliegenden Ergebnisse ist, sind es die beiden folgenden Papers, die für die Entwicklung von *Grid Computing* richtungweisend waren und es auch heute noch sind:

[2] Ian Foster, Professor an der Universität von Chicago, ist neben Carl Kesselman und Steven Tuecke einer der führenden Forscher auf dem Gebiet *Grid Computing*. Eine umfassende Übersicht zu seinen Publikationen kann seiner Homepage entnommen werden: http://www-fp.mcs.anl.gov/~foster/

[3] Rajkumar Buyya, *Senior Lecturer* und Direktor des *Grid Computing and Distributed Systems Laboratory* an der Universität von Melbourne.

- The Anatomy of the Grid, [15]

  Aus dem Abstract: *[…] Next, we present an extensible and open Grid architecture, in which protocols, services, application programming interfaces, and software development kits are categorized according to their roles in enabling resource sharing. We describe requirements that we believe any such mechanisms must satisfy and we discuss the importance of defining a compact set of intergrid protocols to enable interoperability among different Grid systems. […]*

- The Physiology of the Grid, [14]

  Aus dem Abstract: *[…] We present an Open Grid Services Architecture that addresses these challenges. Building on concepts and technologies from the Grid and Web services communities, this architecture defines a uniform exposed service semantics (the Grid service); defines standard mechanisms for creating, naming, and discovering transient Grid service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities. The Open Grid Services Architecture also defines, in terms of Web Services Description Language (WSDL) interfaces and associated conventions, mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management and notification. […]*

  Die in diesem Paper beschriebene *Open Grid Services Architecture* (OGSA[4]) definiert, was *Grid Services* sind. Sie beschreibt keine Implementierungsansätze oder technischen Details, sondern versteht sich als Standardarchitektur für Grid-Applikationen.

Auch wenn der Begriff und die dahinter stehenden Konzepte von *Grid Computing* erst in den letzten Jahren an Bedeutung gewonnen haben, so sind viele Konzepte, auf denen es beruht, nicht neu, sondern wurden in Clustersystemen realisiert und finden seit Jahren ihren Einsatz. Neu an *Grid Computing* ist die Kombination der drei von Ian Foster aufgezählten Bedingungen in einem System. Neben oftmals dann doch wieder proprietären[5] Lösungen sind es vor allem das Global Grid Forum (GGF) sowie das Globus Toolkit, die eine führende Rolle im Bereich *Grid Computing* eingenommen haben.

Das Global Grid Forum[6] besteht aus *Working* und *Research Groups*, die sich mit den unterschiedlichsten Themenbereichen im Zusammenhang mit *Grid Computing* auseinandersetzen und Standardisierungsprozesse voranbringen. In Kapitel 3.3.6 und 3.3.7 wird kurz auf die Ergebnisse zweier *Research Groups* eingegangen. Das GGF beschreibt sich im Internet selbst wie folgt:

[4] Globus Alliance, OGSA Homepage: http://www.globus.org/ogsa/

[5] Proprietär, da nicht alle gestellten Anforderungen erfüllt werden. Besonders häufig werden eigene, mit anderen Frameworks nicht kompatible Schnittstellen verwendet.

[6] Global Grid Forum Homepage: http://www.ggf.org/

> *The Global Grid Forum (GGF) is a community-initiated forum of thousands of individuals from industry and research leading the global standardization effort for grid computing. GGF's primary objectives are to promote and support the development, deployment, and implementation of Grid technologies and applications via the creation and documentation of "best practices" - technical specifications, user experiences, and implementation guideline*

Das Globus Toolkit, ein Produkt der Globus Alliance[7], ist eine Implementierung der *Open Grid Services Infrastructure* (OGSI). Selbstbeschreibung der Globus Alliance:

> *The Globus Alliance is a research and development project focused on enabling the application of Grid concepts to scientific and engineering computing. […] Globus Alliance software development has resulted in the Globus Toolkit, a set of services and software libraries to support Grids and Grid applications. The Toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability.*

Bei der eben erwähnten *Open Grid Services Infrastruckture* (OGSI) handelt es sich um eine formale technische Spezifikation ([39]) der in OGSA beschriebenen Konzepte, vor allem der *Grid Services*, die eine Weiterentwicklung der *Web Services* sind. In Zusammenarbeit der Globus Alliance mit IBM erfolgte eine Weiterentwicklung von OGSI hin zum *Web Services Resource Framework* (WSRF[8]), dieses wird im Globus Toolkit 4 zur Anwendung kommen.

2.2 Idle Time Computing

In Kapitel 1.1 erfolgte bereits eine erste Differenzierung von *Grid Computing* in die beiden Bereiche Cluster- und *Idle Time Computing*. Basis der Trennung ist dabei die Kombination aus verwendeter Hard- und Software und der damit verbundene Einsatzbereich der betrachteten Rechner. In *Cluster Computing* werden Systeme benutzt, die ausschließlich zur Abarbeitung von zentral vergebenen Jobs verwendet werden. *Idle Time Computing* findet dagegen auf Arbeitsplatzrechnern statt, hier ist die Abarbeitung von fremden, d.h. nicht vom lokalen Anwender vergebener Jobs nur sekundär, primär stehen die Ressourcen dem lokalen Anwender zur Verfügung.

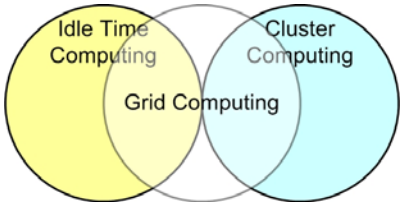


Abbildung 1: Abgrenzung von *Idle Time* -, *Cluster* - und *Grid Computing*

Grid Computing ist in beiden Domänen vertreten, sowohl im *Idle Time Computing* als auch im *Cluster Computing*, Kriterium ist, dass die drei in Kapitel 2.1 aufgelisteten Anforderungen erfüllt werden. *Idle Time Computing* wird derzeit vom *Wide Area Distributed Computing* (WADC) beherrscht, hierbei dient das Framework zumeist dem Berechnen eines Problems eines klar erkennbaren Auftraggebers und bietet keine Möglichkeit, für andere Berechnungen genutzt zu werden. *Cluster*

[7] Globus Alliance Homepage: http://www.globus.org/
[8] Globus Alliance, Web Service Resource Framework Homepage: http://www.globus.org/wsrf/

Computing ist vor allem geprägt durch so genannte Batch-Systeme, diese erfüllen aufgrund ihrer starken Zentralität sowie der proprietären Protokolle gleich zwei der drei erwähnten Kriterien nicht.

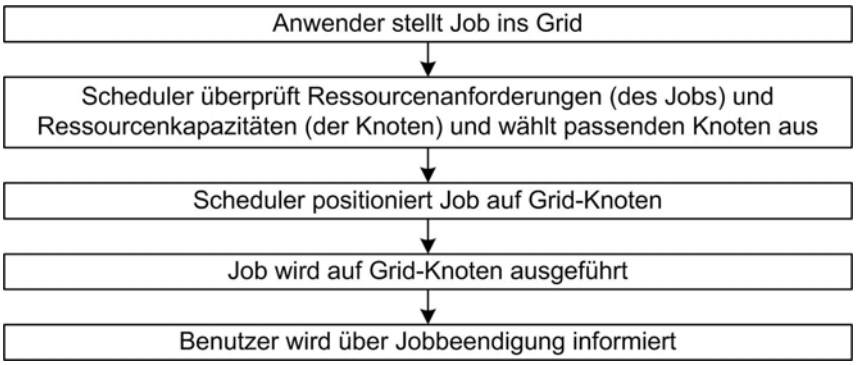


Abbildung 2: Schematisches[9] Abarbeitungsmodell von *Grid Computing*

Das in der Abbildung dargestellte Abarbeitungsmodell gilt für *Grid Computing* im Allgemeinen. Auf dieser Abstraktionsebene lassen sich keine Unterschiede zwischen *Cluster Computing* und *Idle Time Computing* finden.

In der Darstellung wird als Standardjobplatzierungsvorgehensweise ein Vergleich zwischen benötigten Ressourcen des Jobs und vorhandenen Ressourcen der Knoten durchgeführt. Dies ist das derzeit gängigste Verfahren, die durch Idle Time Computing entstehende Problematik wird in Kapitel 0 aufgezeigt.

[9] Der Ressourcenabgleich wird nicht in allen Systemen durchgeführt. Je nach verwendetem Grid-System müssen der Abbildung weitere Elemente hinzugefügt werden. Sicherheitsaspekte wie Authentifizierung sind in der Abbildung nicht dargestellt.

3 Ressourcenpartitionierung

In diesem Kapitel werden der Begriff Ressourcenpartitionierung sowie seine Bedeutung für *Idle Time Computing* erörtert. Anschließend erfolgt eine Betrachtung über vorhandene Möglichkeiten und Forschungsarbeiten zum Thema Ressourcenpartitionierung in *Idle Time Computing*. Eine Konkretisierung des Begriffs Ressource ist in diesem Kapitel noch nicht notwendig, sondern erfolgt in Kapitel 4 bei der Analyse der Ressourcenpartitonierungsmöglichkeiten in Desktopbetriebssystemen.

3.1 Ressourcenpartitionierung und Scheduling

Unter Ressourcenpartitionierung versteht man die Möglichkeit, Ressourcen anhand festgelegter Kriterien aufzuteilen und bestimmten Verbrauchern und Konsumenten zuzuweisen. Ressourcenpartitionierung bewegt sich im gleichen Umfeld wie der sehr umfangreiche Themenkomplex Scheduling, jedoch auf einer anderen Ebene der Ressourcenkontrolle. Scheduling ist die Zuordnung von gegebenen Ressourcen zu Aufträgen unter Erstellung von Zeitplänen. Die Reihenfolge ist dabei nicht beliebig sondern richtet sich nach den von den Aufträgen gestellten Anforderungen und wird vom Scheduler ermittelt. Als Beispiel seien CPU Scheduler erwähnt, welche einzelnen Prozessen und Threads Zeitslots gewährleisten, in denen sie auf der CPU arbeiten können. Scheduler erstellen konkrete Ablaufpläne, wie, wann und wo welcher Verbraucher welche Ressource erhält, jedoch sind sie dabei räumlich und zeitlich begrenzt. Ressourcenpartitionierung erfolgt auf einer höheren Ebene, sie beschäftigt sich nicht mit der detaillierten Planung von Ablaufplänen, sondern mit der Verteilung von Ressourcenkontingenten an bestimmte Verbraucher bzw. Verbrauchergruppen und zwar über die Grenzen von lokalen Schedulern hinweg.

3.2 Ressourcenpartitionierung in Idle Time Computing

Ressourcenpartitionierung in *Idle Time Computing* wird von zwei Problemfeldern beherrscht, die in den beiden folgenden Unterkapiteln analysiert sowie ihre Ist- und Soll-Zustände aufgezeigt werden.

3.2.1 Administrative Kontrolle der Ressourcen

Problemdiskussion

Die Installation und Einrichtung eines *Idle Time Computing* Frameworks innerhalb einer Arbeitsplatzdomäne, sei es in einem Unternehmen oder innerhalb der Studentenpools an einer Universität, birgt für alle Beteiligten ein hohes Maß an Risiko. Neben der Gefahr, dass die dem *Idle Time Computing* zugrunde liegende Software ein unerlaubtes Eindringen in den Rechner oder die unerlaubte Erlangung von administrativen Rechten auf einem Rechner ermöglicht, ist es vor allem der teilweise Verlust der Kontrolle über die einzelnen Rechner, der sehr schwer wiegt. Ein *Idle Time Computing* Framework wird, verteilt über alle dafür vorgesehenen Rechner eines Netzes, installiert und bietet anschließend die Möglichkeit, innerhalb

des Frameworks Jobs auszuführen. Hierbei ist in erster Linie niemandem bekannt, auf welchen und auf wie vielen Knoten der jeweilige Job ausgeführt wird und welche Ressourcen dafür verwendet werden. Auch wenn innerhalb des Frameworks bekannt ist, welcher Job von welchem Benutzer in Auftrag gegeben wurde, so ist dieses nur nützlich, wenn eine regelmäßige Überprüfung der Aktivitäten innerhalb des ITC Frameworks von administrativer Seite erfolgt. Eine solche Überprüfung ist zeitaufwendig und es kann nicht erwartet werden, dass sich die Administration, welche den meisten Aufwand und den geringsten Nutzen bei der Installation eines ITC Frameworks hat, um eine regelmäßige Inspektion der ITC Aktivitäten kümmern wird.

Problematisch an dieser Situation sind vor allem zwei Punkte: Zum einen die Komplexität der Jobs, die im ITC bearbeitet werden sollen und zum anderen die Komplexität des ITC Frameworks an sich. *Grid Computing* und somit auch ITC wird vor allem zur Bearbeitung ressourcenintensiver Jobs verwendet. Ressourcenintensiv bedeutet in den meisten Fällen, dass es vor allem einen hohen Bedarf an Rechenleistung gibt, der sich zum einen auf den Faktor Zeit auswirkt, also auf die Laufzeit des Jobs, der sich aber ebenfalls auf Anzahl der parallel laufenden Prozesse eines Jobs auswirkt, also auf die Anzahl der Maschinen, die den Job gleichzeitig ausführen. Im Bereich Video-Rendering und Raytracing bedeutet ressourcenintensiv aber auch, dass große Datenmengen lokal auf den einzelnen Knoten produziert und anschließend über das Netzwerk zwecks Zusammenführung verschickt werden. Unabhängig vom konkreten Ressourcentyp lässt sich feststellen, dass Jobs des Idle-Time-Computings häufig eine oder mehrere Ressourcen über einen längeren Zeitraum verbrauchen. Hierbei ist sowohl der administrativen Seite als auch oftmals den Anwendern des ITC Frameworks unbekannt, welchen Ressourcenbedarf einzelne Jobs haben. Die Komplexität der ITC Frameworks macht es zudem schwierig vorherzusagen, auf welchen Knoten welche Jobs wann und zu welcher Uhrzeit ausgeführt werden. Es ist auch möglich, dass abgebrochene Jobs auf einer anderen Maschine weitergeführt oder von vorne gestartet werden.

Der Kontrollverlust über die Ressourcen wird vor allem deshalb als ein solcher empfunden, da zwar weiterhin Ressourcen lokal kontrolliert werden können und dem Administrator die Möglichkeit bestehen bleibt, einzelne Prozesse zu beenden oder Maschinen abzuschalten, jedoch existiert keine netzweite Möglichkeit Ressourcen zu kontrollieren, während dagegen ein ITC Framework die Möglichkeit hat, netzweit Ressourcen zu belegen und zu blockieren. Die Installation eines ITC Frameworks wird also nur dann von administrativer Seite erfolgen, wenn entweder ein Vertrauen in die Software vorliegt oder wenn andererseits die Möglichkeit besteht, diese Software bzw. die von ihr genutzten Ressourcen des Netzes mit einem sehr geringen Aufwand zu kontrollieren.

Der unkontrollierte Ressourcenverbrauch macht sich auf verschiedene Art und Weise bemerkbar. Sofern im ITC Framework keine Protokollierung erfolgt, welcher Nutzer wie viel Kapazität beansprucht, bietet das Framework die Möglichkeit, mehr oder weniger unkontrolliert Jobs einzuspeisen und diese zu starten. Da es sich oftmals um sehr rechenintensive Prozesse handelt, ist es nicht unwahrscheinlich, dass diese über einen längeren Zeitraum laufen und große Ressourcenmengen verbrauchen bzw. belegen, ohne dass dieses für den Grid-Anwender erkennbar ist. Der Einsatz von *Grid Computing* kann somit zu einer Selbstbedienungsmentalität führen, d.h. dem (gewollt oder ungewollt) gewissenlosen Verbrauch von enormen Kapazitäten durch häufiges und wiederholtes Starten von Jobs. Die Auswirkungen solcher Handlungsweisen sind erkennbar: Große Ressourcenmengen werden verbraucht, was sich sowohl auf die einzelnen Maschinen wie auch auf dem Netz

bemerkbar macht und die Arbeit der lokalen Anwender stört oder sogar verhindert. Diese sind bei der bisherigen Betrachtung des Ressourcenverbrauchs noch nicht beachtet worden, obwohl sie es sind, die den höchsten Preis für die Installation und Verwendung eines ITC Frameworks zu zahlen haben. ITC verwendet zur Abarbeitung der Jobs lokale Arbeitsplatzrechner und auch wenn es innerhalb des ITC Frameworks Methoden und Verfahren gibt, den Ressourcenverbrauch derart zu gestalten, dass Prozesse des lokal arbeitenden Anwenders nur gering beeinflusst werden, so besteht doch immer die Möglichkeit, dass durch exzessive Anwendung von ITC dieses nicht mehr gewährleistet ist. Beispielsweise können durch Jobs des ITC' lokale Festplatten bis zur Kapazitätsgrenze beschrieben werden, was zur Folge hat, dass dem Anwender bzw. dem Betriebssystem kein Auslagerungsplatz zur Verfügung steht. Das unkontrollierte Senden hoher Datenmengen durch Jobs des ITC' auf mehreren Knoten ist ein anderes Beispiel, in diesem Fall können zentrale Netzressourcen wie NFS Server für andere Anwendungen blockiert werden. Zudem ist es heutzutage üblich, dass auf Arbeitsplatzrechnern viele verschiedene Hintergrunddienste (z.B. Email-Client) aktiv sind und diese können durch das ITC ebenfalls beeinträchtigt werden.

Zusammenfassend kann gesagt werden, dass seitens der Administration der Bedarf besteht, den Ressourcenverbrauch vom *Idle Time Computing* als Ganzes zu kontrollieren und im Bedarfsfall zu begrenzen, um zum einen die Kontrolle über das Netz und über die einzelnen Rechner im Netz nicht zu verlieren, um andererseits einen Schutz der lokalen Anwender und ihrer Prozesse vor dem Ressourcenverbrauch des ITC' zu erreichen.

Soll-Zustand

Um die gewünschte Begrenzung des Ressourcenverbrauchs zu realisieren, bedarf es eines domänenweiten[10] Mechanismus', der der Administration die Möglichkeit gibt, zentrale Ressourcenpartitionierung durchzuführen. Sofern ein derartiger Mechanismus nicht vom *Grid Framework* mitgeliefert wird, ist ein eigenständiges Framework notwendig, welches unter anderem die folgenden Anwendungsszenarien[11] unterstützt:

1. Beschränkung der verwendeten Festplattenbelegung um sicherzustellen, dass genügend Kapazität für Programmdaten anderer Anwendungen sowie für zukünftige Installationen vorhanden ist. Ebenfalls dient dieses dem Schutz vor einem vollständigen Belegen der Festplatte, was unter Umständen zu einer Beeinträchtigung der Systemstabilität führen kann.
2. Beschränkung des verwendeten Arbeitsspeichers[12] um die Lauffähigkeit des Systems und anderer Anwendungen sicherzustellen.
3. Beschränkung der durch einen Job auf einer Anlage verwendeten CPU Zeit um anhaltende Dauerlast zu vermeiden und um außer Kontrolle geratene Endlosprozesse zu verhindern.
4. Beschränkung der lokal verwendeten Bandbreite beim Senden von Daten um Netzwerkverkehr anderer Anwendungen, besonders von Anwendungen eines lokalen Benutzers, nur gering zu beeinflussen.

[10] Domänenweit meint den Wirkungskreis der im Text erwähnten Administration.
[11] Es handelt sich bei der Auflistung nur um eine Auswahl von Anwendungsszenarien. Gleiches gilt für die noch kommende Aufzählung in Kapitel 0.
[12] Je nach Szenario wahlweise physikalischer oder virtueller Speicher.

5. Beschränkung des netzweiten Datenverkehrs um zu verhindern, dass der von Jobs des Grid-Computings erzeugte Netzwerkverkehr keine Netzelemente wie NFS Server oder Netzwerkknoten (z.B. Switches, Router) blockiert.

Ist-Zustand

Derzeit wird von keinem Grid-System die Möglichkeit zur administrativen Ressourcenpartitionierung mitgeliefert und es ist auch kein eigenständiges Framework für heterogene Systemlandschaften vorhanden.

3.2.2 Mindestgarantien zur Sicherstellung der Lauffähigkeit

Problemdiskussion

Jedes Programm bzw. jeder Prozess hat eine Anzahl von Mindestkriterien, die erfüllt sein müssen, damit die Lauffähigkeit des Programms gewährleistet ist. Diese Mindestkriterien sind aufteilbar in funktionale und nichtfunktionale Anforderungen. Funktionale Anforderungen bzw. Mindestkriterien sind harte Kriterien, was bedeutet, dass das Programm nicht lauffähig ist, wenn sie nicht garantiert werden können. Nichtfunktionale Anforderungen bzw. Mindestkriterien müssen dagegen nicht erfüllt sein, um die Lauffähigkeit zu garantieren, jedoch müssen sie erfüllt sein, um das Programm in einem bestimmten Kontext verwenden zu können. Video-Streaming über ein Netzwerk ist auch dann möglich, wenn die zur Verfügung stehende Bandbreite geringer als die benötigte ist, doch ist auf Empfängerseite kein für den Menschen angenehmes Betrachten des Videos mehr möglich. Gleiches gilt für Anwendungen auf Rechnern mit zu geringer Rechenleistung: Die Anwendungen sind lauffähig aber für das subjektive Empfinden sind sie zu langsam und ein erträgliches Arbeiten ist nicht möglich. Für Programme und Jobs, die innerhalb eines *Idle Time Computing* Frameworks ausgeführt werden, gelten die gleichen Vorraussetzungen. Sie haben funktionale Mindestkriterien die erfüllt sein müssen, um die Lauffähigkeit sicherzustellen und sie haben nichtfunktionale Mindestkriterien, die erfüllt sein sollten. Zur zweiten Kategorie gehört vor allem die Forderung, dass Jobs die nötigen Ressourcen bekommen, um in angemessener Zeit abgearbeitet werden zu können. Diese Forderung gründet auf der hinter *Idle Time Computing* stehenden Idee, nämlich der Nutzung fremder Ressourcen, um Probleme in geringerer Zeit lösen zu können. Verlängert sich die Ausführzeit von Jobs, bedingt durch Ressourcenknappheit, ist im negativsten Fall kein Gewinn für den Anwender erkennbar, da ein lokales Ausführen auf seinem eigenen Rechner weniger Zeit verbraucht hätte. Hiermit in Zusammenhang steht auch, dass die Verzögerung eines Jobs, bei der parallelen Ausführung von untereinander kommunizierenden Jobs, bedeuten kann, dass andere Jobs ebenfalls verzögert werden. Da die Anforderung nach einer angemessenen Ausführzeit nur dann garantierbar ist, wenn der Ressourcenverbrauch und –bedarf im Voraus bestimmbar ist, dient diese Forderung vor allem der Garantie von Mindestressourcen, um ein stetes Weiterarbeiten der Jobs zu gewähren. Dem gegenüber verhält sich die Situation bei harten Mindestanforderungen anders, hier können klare Aussagen gemacht werden, um festzustellen, unter welchen Bedingungen Jobs lauffähig sind oder nicht.

Um die Lauffähigkeit von Jobs sicherzustellen, besteht folglich der Bedarf nach einem Mechanismus, im *Idle Time Computing* minimale Ressourcenverfügbarkeit zu garantieren.

Soll-Zustand

Entweder wird die Möglichkeit des Garantierens von Mindestressourcen durch das jeweilige Grid-System eingeräumt oder dieses erfolgt durch externe Mechanismen. Wird eine eigenständige Lösung verwendet, sollte diese jedoch über das Grid-System angesteuert und kontrolliert werden. Nur dann kann eine automatische Reservierung der Mindestressourcen erfolgen. Besteht keine Verbindung zwischen Grid-System und dem Ressourcenreservierungsmechanismus, so muss das Reservieren manuell vom Grid-Anwender, der den Job ins Grid gespeist hat, erfolgen. Ein solches Vorgehen ist nicht praktikabel, zumal für den Anwender nicht ersichtlich ist, auf welchem Knoten sein Job im Grid platziert wird. Folgende Anwendungsszenarien sollten unter anderem unterstützt werden:

1. Forderung nach einer bestimmten verfügbaren lokalen Festplattenkapazität für während der Laufzeit des Jobs erzeugte temporäre Daten.
2. Forderung nach einer bestimmten Menge von verfügbarem Arbeitsspeicher.
3. Forderung nach verfügbarer Netzwerkbandbreite um bestimmte Datenmengen in begrenzter Zeit zu transferieren.
4. Allgemeine Forderung nach garantierten Minimalanteilen an allen Ressourcen. Diese Forderung gründet darauf, dass ein Job nicht vollständig zum Erliegen kommen soll, selbst wenn seitens lokaler Aktivitäten eine sehr hohe Ressourcenauslastung erreicht wird.

Ist-Zustand

In mehreren Distributed-Computing-Systemen findet ein Abgleich der Jobanforderungen mit den Ressourcenkapazitäten der jeweiligen Knoten statt. Hieraus wird eine Menge von, für den Job geeigneten, Knoten ermittelt, woraus je nach System ein beliebiger oder ein optimaler Knoten ausgewählt wird, auf dem dann anschließend der Job ausgeführt wird. Problematisch hierbei ist, dass die Angaben zu den Knoten meistens statisch sind und nur die absoluten Ressourcenkapazitäten widerspiegeln, nicht jedoch die aktuelle Auslastung. Im Bereich *Cluster Computing* ist dieses Verfahren zum Erreichen der Mindestgarantien ausreichend, da auf den für das *Cluster Computing* dedizierten Systemen keine anderen Anwendungen laufen und der Job im Allgemeinen vollen Zugriff auf alle Ressourcen hat. Beim *Idle Time Computing* verhält es sich jedoch anders: Aufgrund der Tätigkeiten des lokalen Anwenders sind die durch das Vergleichen der statischen Informationen ermittelten Ressourcenkapazitäten nicht für den Job des ITC' reserviert, sondern werden unter allen Prozessen aufgeteilt. Es werden folglich keine Mindestgarantien erreicht. Mit Ausnahme des GARA Frameworks (Betrachtung erfolgt in Kapitel 3.3.5 .) ist kein System zum Erwirken von Ressourcengarantien in heterogenen Systemen bekannt.

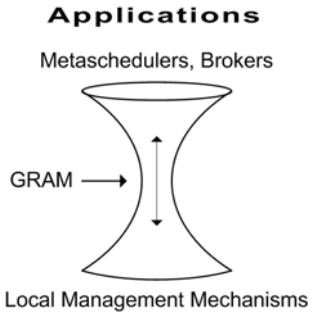
3.3 Vorhandene Lösungen und verwandte Arbeiten

In diesem Kapitel werden Lösungen vorgestellt und analysiert, die Ressourcenmanagement, Ressourcenpartitionierung und Ressourcen-Brokering im Kontext von *Grid Computing* in heterogenen Netzen ermöglichen bzw. damit im Zusammenhang stehen. Neben Mechanismen innerhalb von ITC Frameworks werden eigenständige Frameworks sowie anderweitige Ansätze im Themenumfeld untersucht. Es kann nicht ausgeschlossen werden, dass bei der Recherche nicht alle

Entwicklungen verfolgt wurden: *Grid Computing* ist ein inzwischen sehr weit verbreitetes Forschungsthema an vielen Universitäten und Instituten weltweit und es besteht eine kaum noch überschaubare Menge von Forschungsarbeiten, die sich mit Ressourcenbehandlung in *Grid Computing* beschäftigt.

3.3.3 Ressourcenmanagement im Globus Toolkit

Wie bereits in Kapitel 2 beschrieben, handelt es sich beim Globus Toolkit um ein Grid-Framework, das für den plattformunabhängigen Einsatz sowohl auf Arbeitsplatzrechnern als auch in Clustern konzipiert ist. Für die Ressourcenkontrolle innerhalb von Globus ist der *Grid Allocation and Manangement* (GRAM) Service zuständig. GRAM dient als netzweite Schnittstelle zur Ressourcenreservierung und zum Ressourcenmanagement zwischen globalen Metaschedulern und Ressourcenbrokern auf der einen und lokalen Schedulern auf der anderen Seite. GRAM selbst bietet weder die Fähigkeit Ressourcen zu partitionieren noch zu schedulen oder zu brokern, sondern dient ausschließlich als API. Das Globus Toolkit beinhaltet keine lokalen Scheduler und auch keine Mechanismen zur globalen Ressourcenpartitionierung, sondern liefert mit GRAM eine Schnittstelle für externe Lösungen wie Condor[13], PBS[14], LSF[15]. Die folgende Grafik veranschaulicht die Position von GRAM innerhalb der Globus Architektur.



Abbildung[16] 3: GRAM im Kontext globaler und lokaler Scheduler

Vladimir Silva beschreibt GRAM in [36] folgendermaßen:

> *The Globus Resource Allocation Manager (GRAM) is at the core of the Globus remote program execution infrastructure. GRAM allows you to run jobs remotely, using a set of WSDL client interfaces to submit, monitor, and terminate a job. A job request is written in the resource specification language (RSL) and processed by the Managed Job Service. In GT3, GRAM is divided into two big containers: the Master Hosting environment (MHE) and the User Hosting environment (UHE). The MHE is in charge of authorizing the GRAM request against the grid-map file and starting or pinging the UHE. It then uses a redirector to forward the request to the UHE process which, in turn, creates a Managed Job Service (MJS). MJS submits the job into the back-end scheduling system.*

[13] Condor und dessen Mechanismen zur Ressourcenpartitionierung werden in Kapitel 3.3.4 besprochen.
[14] Ressourcenkontrolle im Batchsystem PBS wird in Kapitel 3.3.10 aufgezeigt.
[15] Platform LSF ist ein kommerzielles Batch-Processing-System mit vergleichbarer Funktionalität wie Condor. Projekthomepage: http://www.platform.com/products/LSF/.
[16] Abbildung nach: http://www-unix.globus.org/toolkit/docs/3.2/gram/key/index.html.

Als Schnittstellensprache verwendet GRAM die über XML Schema definierte *Ressource Specification Language* (RSL).

Die folgende Grafik zeigt den strukturellen Aufbau eines Globus Grids in Bezug auf Ressourcenkontrolle.

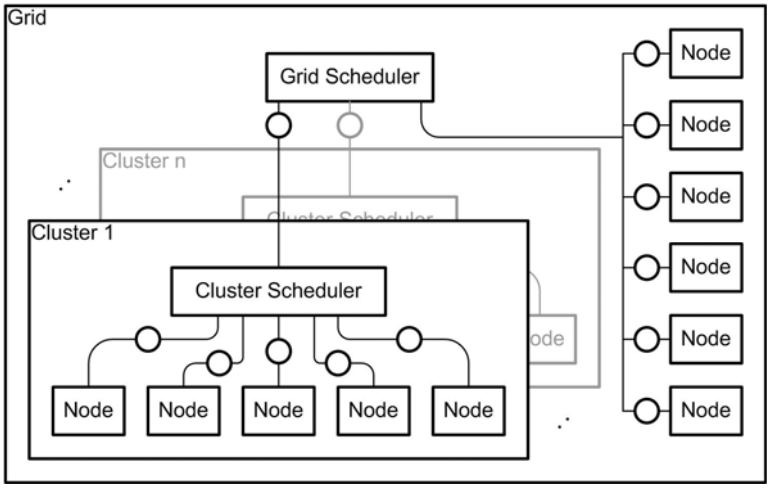


Abbildung 4: Ressourcenkontrolle im GT3

In der Abbildung sind mehrere Cluster-Systeme[17] zu erkennen, die zusammen mit dem übergeordneten Globus System das Grid bilden. Globus kommuniziert zu den jeweiligen Cluster-Schedulern über GRAM mittels RSL, es müssen hierbei nicht die gleichen Clustersysteme verwendet werden. Die Verwendung eines Clustersystems ist in Globus nicht zwingend notwendig. Es ist ebenso möglich, Knoten direkt als Clients zu verwenden. Im Allgemeinen erfolgt die Verwendung von Cluster-Systemen, da diese integrierte Scheduler und andere Jobplatzierungsmechanismen mitliefern.

Derzeit wird GRAM ausschließlich zum Ansprechen von externen Cluster-Schedulern verwendet, diese bekommen von Globus Grid-Jobs übergeben und platzieren sie (von Globus unabhängig) auf Knoten in dem von ihnen verwalteten Cluster.

3.3.4 Ressourcenmanagement in Condor

Condor[18] ist ein an der Universität von Wisconsin-Madison entwickeltes System zum verteilten Rechnen in heterogenen Netzen. Der Schwerpunkt liegt hierbei auf *High Troughput Computing*, d.h. dem Bereitstellen von hoher Rechenleistung über einen langen Zeitraum zur Abarbeitung von Batch-Jobs. Condor ist ein klassisches *Idle Time Computing Framework*, das die ungenutzten Ressourcen von Arbeitsplatzrechnern verwendet, um Batch-Jobs auszuführen, diese jedoch sofort unterbricht, sobald Aktivität seitens eines lokalen Nutzers vorliegt. Das Einsatzspektrum von Condor reicht dabei von der Verwaltung kleinerer Workstationpools bis hin zur Bündelung der Rechenleistung von mehreren Tausend Arbeitsplatzrechnern. Interessant in diesem Zusammenhang ist die Frage, inwieweit Condor als *Grid Framework* angesehen werden kann oder nicht. Aufgrund der

[17] Im obigen Zitat als *back-end scheduling system* bezeichnet. Hierunter fallen die erwähnten Systeme Condor, PBS und LSF.

[18] Condor Projekthomepage: http://www.cs.wisc.edu/condor/

Beschränkung auf lokale Domänen wird Condor nicht als Grid-System angesehen. Inzwischen bietet Condor zwei Möglichkeiten, innerhalb eines Globus Netzes als lokaler Cluster-Scheduler zu arbeiten. Eine Beschreibung der Grid-Anbindungsmöglichkeiten kann in [08], Kapitel 5 nachgelesen werden.

Ressourcenmanagement erfolgt bei Condor mittels Matchmaking, einem Mechanismus der überprüft, auf welchem Knoten der jeweilige Job lauffähig ist und welches der am Besten geeignete Knoten für den Job ist. In [31], Bericht 4, Kapitel 2.4.1 wird Matchmaking wie folgt beschrieben wird:

> *Erreicht wird dies durch den Mechanismus der ClassAds. Hier gibt jede beteiligte Entität in einer so genannten ClassAd bekannt, welche (Muss- und Kann-) Ansprüche sie an die potentiell ihr zuzuordnenden Entitäten hat, und welche Eigenschaften, die die Gegenseite interessieren könnte, sie selbst aufweist. Diese ClassAds werden dann an einem zentralen Ort im System (dem ClassAd-Dämonen) gesammelt, und regelmäßig untereinander durch den Matchmaker verglichen. Ergeben sich dabei Paare, wird das den beteiligten Entitäten mitgeteilt, so dass diese aushandeln können, ob sie wirklich zueinander passen, um dann – im Fall von Jobs und Ressourcen – den Job auf die Ressource zu übertragen und auszuführen. Der Matchmaker selber versteht dabei nichts von der Semantik der verglichenen Angaben zwischen Entitäten, er vergleicht letztlich nur Attribut-Wert-Paare, aus denen die ClassAds bestehen.*

Etwas weniger Allgemein und auf den Fall der Ressourcenkontrolle bezogen bedeutet dies, dass sowohl Ressourcenkonsumenten (Jobs) als auch Ressourcenprovider (Knoten) dem Matchmaker ClassAds zur Verfügung stellen, in denen sie mittels Key/Value Paaren ihren Bedarf bzw. ihre Kapazität mitteilen. Anhand dieser Attributtabellen erfolgt die Zuweisung von Jobs auf Knoten. Dabei ist zu beachten, dass keine automatische Erkennung des Ressourcenbedarfs erfolgt, sondern der Absender des Jobs diese manuell erstellen muss. Auf der anderen Seite werden beim Installieren von Condor die relevantesten Key/Value Paare erkannt und eine entsprechende ClassAd erstellt. Das Konzept beinhaltet nicht die Beschränkung oder das Reservieren von Ressourcen, sondern der Mechanismus dient dem Finden von zueinander passenden Paaren. Zur Laufzeit von Condor-Jobs erfolgt auf den jeweiligen Knoten keine Beschränkung der Ressourcen und der Job hat vollen Zugriff auf die vom Betriebssystem gewährten Ressourcen. Einzige Ausnahme ist, dass im Regelfall (sofern es nicht anders in der lokalen ClassAd definiert ist) ein Job nur dann ausgeführt wird, wenn keine Aktivität seitens des lokalen Anwenders vorliegt.

3.3.5 Das GARA Framework

Das GARA Framework wurde entwickelt, um end-to-end[19] Quality of Service innerhalb des Globus Toolkits zu ermöglichen. Hierfür stellt es Mechanismen zur Ressourcenreservierung bereit und garantiert anschließend dem Job die von ihm reservierten Ressourcen. Ressourcenreservierung ist dabei sowohl ad-hoc als auch im Voraus möglich. Auf unterster Ebene der GARA Architektur stehen dabei Ressourcenmanager, welche für jeweils eine konkrete Ressource auf einem Knoten zuständig sind. Folgende Ressourcen werden von GARA unterstützt:

- Netzwerkbandbreite mittels *Differentiated Services*
- CPU Anteile mittels des *Dynamic Soft Realtime CPU Schedulers* (DSRT), in Kapitel 0 erfolgt eine Betrachtung des DSRT

[19] Der Begriff *end-to-end Quality of Service* bedeutet, dass Ressourcengarantien von einem Kommunikationsendpunkt bis zum anderen aufgebaut werden und für die Dauer der Verbindung gehalten werden.

- Zugriff auf DPSS (Distributed Parallel Storage System)

Ursprünglich wurde das GARA Framework für Globus 1 entwickelt, daran beteiligt waren Ian Foster, Volker Sander und Alain Roy. Eine Portierung[20] für Globus 2 hat es von Elisabetti Ronchieri am italienischen CNAF Institut gegeben, für Globus 3 liegt keine Portierung vor. Die Projekthomepage wurde seit Sommer 2000 nicht mehr verändert und laut Aussagen von Ian Foster und Volker Sander wird das Projekt nicht weiter verfolgt und verwendet, lediglich Volker Sander benutzt die ursprüngliche Version am Forschungszentrum Jülich für eigene Forschungszwecke. Dass das GARA Framework trotzdem an dieser Stelle behandelt wird, hat vor allem zwei Gründe: Zum einen war es der bisher einzige ernsthafte Ansatz zur Realisierung eines Ressourcenbrokers in Globus und es ist bedauerlich, dass er nicht weiter verfolgt wurde, zum anderen sind im Umfeld von GARA diverse Forschungsarbeiten entstanden: Mittels GARA wurden mehrere Verfahren im Bereich co-allocation[21] sowie Netzwerk *Quality of Service* evaluiert. Hervorzuheben sind die Dissertation von Alain Roy ([32]), die neben einer ausführlichen Evaluation von GARA auf andere *end-to-end Ressource Reservation* Systeme eingeht, sowie die Dissertation von Volker Sander ([34]), die sich insbesondere mit der mathematisch-theoretischen Analyse von aggregierten Netzwerkströmen beschäftigt und zur Evaluierung der Theorien GARA verwendet.

3.3.6 Distributed Resource Management Application API

Die DRMAA ist eine von der gleichnamigen *Global Grid Forum Working Group* entwickelte API zur Absetzung und Steuerung von Jobs in *Distributed Resource Management* Systemen. Im Jahr 2003 wurde die Version 1.0 der Spezifikation ([20]) veröffentlicht, derzeit findet sie noch in keinem Grid-System Verwendung.

3.3.7 Grid Resource Allocation Agreement Protocol

Das GRAAP ist ein von der gleichnamigen *Global Grid Forum Working Group* in Entwicklung befindliches Protokoll zur Kommunikation zwischen höheren Diensten (wie Super-Schedulern) und lokalen Schedulern. Das Protokoll hat das Entwurfsstadium noch nicht verlassen, detaillierte Informationen können der *Working Group* Homepage[22] entnommen werden.

3.3.8 SNAP

Eine Hauptanforderung von Grid-Systemen ist das Aushandeln von Zugriffen auf fremde, d.h. außerhalb der eigenen Domäne liegenden, Ressourcenkapazitäten sowie das Management des Ressourcenzugriffs bzw. die Überwachung der Einhaltung der ausgehandelten *Service Level Agreements* (SLA). Das *Service Negotiation and Acquisition Protcol* [07] wurde hierzu entwickelt, jedoch wurde es bisher nur theoretisch behandelt, eine Validierung des entwickelten Modells steht noch aus.

[20] Die Portierung von GARA für Globus 2 sowie dazugehörige Dokumentation können unter folgender Internetadresse gefunden werden: http://www.cnaf.infn.it/joda/GARA/

[21] *Coordinated allocation* bedeutet, dass mehrere Ressourcen unterschiedlichen Typs gleichzeitig reserviert werden, um Engpässe bei nicht reservierten Ressourcen zu vermeiden.

[22] GGF, GRAAP WG Homepage: http://www.fz-juelich.de/zam/RD/coop/ggf/graap/graap-wg.html

3.3.9 OGSI-Agreement

Ein OGSI-kompatibles Modell zur Ressourcenaushandlung ist das *Agreement-based Grid Service Management Model* (OGSI-Agreement, [06]). Es geht vor allem auf die Anbindung von nicht Grid-spezifischen Komponenten sowie dem Aushandeln von *Service Level Agreements* ein. Eine Verwendung in Globus findet derzeit nicht statt, die nachfolgende *Web Services Agreement Specification* (WS-Agreement, [05]) hat das Entwurfsstadium noch nicht verlassen.

3.3.10 PBS/Maui und andere Batch Systeme

Das Portable Batch System (PBS) ist ein Stapelverarbeitungssystem, bei dem es möglich ist, verschiedene Scheduler wie den Maui Scheduler zu verwenden. Die Ressourcenkontrolle erfolgt seitens Maui, wobei ähnlich wie in Condor der Schwerpunkt auf Scheduling liegt. Maui erstellt Job Queues, verwaltet Prioritäten und findet passende Job-Knoten-Paare. Wie bei den meisten Batch Systemen dieser Art liegt der Schwerpunkt nicht auf *Idle Time Computing*, sondern es werden dedizierte Rechner fürs *Batch Processing* (*Cluster Computing*) verwendet. Aus diesem Grund wird auf Systeme dieser Art nicht weiter eingegangen. Anzumerken sei noch, dass PBS wie Condor und andere Batch-Processing-Systeme in Globus als *Site Scheduler* eingebunden werden kann.

3.3.11 Wide Area Distributed Computing

Wie in Kapitel 2.2 erörtert, gehören WADC Systeme nicht zur Familie der Grid-Systeme, funktionieren jedoch meistens nach dem *Idle Time Computing* Prinzip, weshalb sie an dieser Stelle erwähnt werden. Mit Ausnahme von CPU Leistung benötigen WADC Systeme im Allgemeinen keine nennenswerten Ressourcen, um lauffähig zu sein. Sie wurden primär geschaffen, um komplexe mathematische oder physikalische Berechnungen durchzuführen. Ihre Ressourcenkontrolle sieht weitestgehend so aus, dass sie maximale CPU Kapazität beanspruchen und gleichzeitig mit der niedrigsten Priorität laufen, sodass andere Prozesse vom betriebssystemeigenen CPU Scheduler bevorzugt und nur unwesentlich vom WADC beeinflusst werden.

3.3.12 Weitere verwandte Arbeiten

Wie bereits erwähnt, ist Ressourcenmanagement in *Grid Computing* ein weltweiter Forschungsschwerpunkt und es existiert eine Vielzahl von Papers, Forschungsarbeiten und Projekten. Neben den bereits genannten Dokumenten sei an dieser Stelle noch auf Folgendes hingewiesen:

Network Weather Service

Network Weather Service (NWS[23]) ist ein System zur Vorhersage von Ressourcenverfügbarkeit in Grid-Systemen. Beschreibung von der Projekthomepage:

> *The Network Weather Service is a distributed system that periodically monitors and dynamically forecasts the performance various network and computational resources can deliver over a given time interval. […] We have developed the NWS for use by dynamic schedulers and to provide statistical Quality-of-Service readings in a networked computational environment. […] and we have developed prototype*

[23] NWS Projekthomepage: http://nws.cs.ucsb.edu/

implementations for Globus and the Global Grid Forum (GGF) Grid Information System (GIS) architecture.

Tordsson: Resource Brokering for Grid Environments

Johan Tordsson beschäftigt sich in seiner Masterarbeit ([37]) mit dem Thema Ressourcen-Brokering in Grid-Systemen. Aus dem Abstract:

> *This thesis describes the development of a broker that bases resource selection on load and performance of the resources as well as characteristics of the application, aiming at minimizing the total time to delivery for each application. The total time to delivery includes the time required for input/output file transfer, batch queue waiting and application execution. The methods used to minimize the total time to delivery includes the usage of advance reservations, characterization of resource performance by the use of benchmarks and estimation of data transfer times from network performance predictions.*

Ressourcen-Brokering in UNICORE

Bei UNICORE[24] handelt es sich um ein vom deutschen Forschungsministerium initiiertes Grid-System. In [01] erfolgt eine Beschreibung der Mechanismen zum Ressourcenmanagement, aus dem Abstract:

> *This paper describes the Resource Discovery and Brokering system used within the UNICORE Grid System and describes how it may be used for selection of resources within a general computational grid. By having a focus on describing systems in terms of a simple standardized information model, it is possible to quickly deploy a UNICORE grid engine on a resource provider while allowing external agents to understand what software is available and what system resources are available to run that software.*

SNMP

Außerhalb der Domäne von Grid Computing sei noch auf das *Simple Network Management Protocol* ([23]) hingewiesen, welches speziell für die Verwaltung und Überwachung von Netzelementen (Router und Switches) konzipiert wurde.

3.3.13 Überlegungen zum Ressourcenbedarf

Die hier vorgestellten Lösungen wenden sich vor allem zwei Themenfeldern der Ressourcenpartitionierung zu:

1. Mechanismen zum Auffinden passender Ressourcenverbraucher/Ressourcenprovider-Paare.
2. Protokolle, Schnittstellen und Modelle zur Ressourcenreservierung sowie zum Aushandeln von *Service Level Agreements* und dem anschließenden Management dieser.

Als gemeinsame Ausgangslage muss in beiden Fällen der Ressourcenbedarf des Jobs bekannt sein. Prinzipiell gibt es hierzu zwei Möglichkeiten: Entweder die Anforderungen müssen seitens des Anwenders, also desjenigen, der den Job ins Grid stellt, spezifiziert werden oder es muss eine automatische Analyse des Jobs und seiner benötigten Ressourcen vorgenommen werden. Gegen die erste Möglichkeit

[24] UNICORE Projekthomepage: http://www.unicore.org/

spricht, dass oftmals nur sehr ungenaue Vermutungen vorliegen, welche Ressourcen in welcher Menge ein Job benötigt. Entweder weiß es der Anwender nicht bzw. er weiß nicht, wie er es in Erfahrung bringen soll oder er unternimmt nicht die notwendigen Handlungen, um dieses in Erfahrung zu bringen. In beiden Fällen wird der Anwender, so er denn seitens des Systems gezwungen ist Angaben zu machen, nur eine grobe Schätzung abgeben können und versucht sein, immer einen höheren Bedarf anzumelden, um zu verhindern, dass der Job aus Mangel an Mindestgarantien nicht lauffähig ist. Ein Anwender von *Grid Computing* sieht vor allem die Möglichkeit, vom Grid anonyme Ressourcen zur Abarbeitung seiner Programme zu bekommen. Dieses möchte er möglichst erreichen ohne im Voraus festlegen zu müssen, welche Ressourcen er benötigt, denn sie sollen ihm zur Laufzeit zur Verfügung stehen[25]. Bei der zweiten Möglichkeit die benötigten Ressourcen zu ermitteln, bedarf es eines automatischen Verfahrens, das den Job analysiert und daraus den Ressourcenbedarf gewinnt. Die bisherigen Forschungsergebnisse auf diesem Gebiet zeigen, dass es bei komplexeren Programmen nicht möglich ist, den gesamten Ressourcenbedarf im Voraus zu ermitteln. Zusammenfassend ist festzustellen, dass der genaue Ressourcenbedarf von Jobs in *Grid Computing* oftmals nicht oder nur unzureichend bekannt ist.

[25] Um die gängige Analogie zwischen *Grid Computing* und Stromnetzen auszudehnen: So wie Stromnetze anonym Strom liefern, so bieten sie die Möglichkeit dieses zu tun, ohne dass vorher der Strombedarf bekannt geben werden muss. Der zweite Teil der Analogie wird im Gegensatz zum ersten nicht allgemein anerkannt.

4 Ressourcenkontrolle in Desktopbetriebssystemen

In diesem Kapitel wird evaluiert, welche Ressourcen in Desktopbetriebssystemen partitioniert werden können und welche Mechanismen dafür vorhanden sind. Neben systeminternen Lösungen werden auch systemnahe Lösungen betrachtet oder Überlegungen zur Realisierung solcher angestellt. Die Evaluierung beschränkt sich auf die derzeit am Häufigsten eingesetzten Betriebsysteme Linux und Microsofts Windows XP.

4.1 Ressourcen

Ressourcen sind anhand ihrer Dimension in drei Typen aufteilbar:

a) [Kapazität]

   Diese Ressourcen haben keine zeitliche Dimension, d.h. sie sind absolut, wie zum Beispiel „64MB Arbeitsspeicher“.

b) [Kapazität] * [Zeit]

   Diese Ressourcen haben eine zeitliche Dimension und sind ein Produkt aus Ressourcenkapazität sowie der Dauer für die Ressource gegeben ist. Ein Beispiel wäre „500MB Festplattenkapazität für 4 Stunden“. Hierbei erfolgt im Allgemeinen eine exakte Festlegung der Kapazität und der Zeit.

c) [Kapazität] / [Zeit]

   Diese Ressourcen haben ebenfalls eine zeitliche Dimension, doch sagen sie etwas über die Kapazität pro Zeitdauer aus. Ein Beispiel wäre „400Kilobit/s Netzwerkbandbreite“.

In einem Computer, in einem Betriebssystem oder in einem Netzwerk von Computern gibt es verschiedene Ressourcenarten. In der folgenden Betrachtung wird eine Auswahl von primär erkennbaren Ressourcen aufgelistet:

1. Festplattenkapazität

   Die Betrachtung der Ressourcen findet auch auf Softwareebene statt, aus diesem Grund ist unter der Festplattenkapazität nicht die physikalische Kapazität eines Datenträgers zu verstehen, sondern die Kapazität einer gemounteten und vom System ansprechbaren Partition.

2. Speicherkapazität

   Vom Betriebssystem verwalteter Arbeitsspeischers. Entweder physikalischer oder virtueller Arbeitsspeicher.

3. Netzwerkkapazität

   Kapazität einer Verbindung zwischen zwei Knoten in einem Netzwerk. Die Kontrolle der verwendeten Netzwerkbandbreite kann derart erfolgen:

   - Allein auf Seite des Senders (Bandbreitenbegrenzung).
   - Gemeinsam von beiden Endpunkten der Verbindung (wie z.B. die Bestimmung der *window size* innerhalb von TCP).
   - Als ein netzweiter Mechanismus (wie *Integrated Services*[26] oder *Differentiated Services*[27]) unter Beteiligung aller Knoten längs der Verbindung.

4. CPU Kapazität

   Die Leistung einer CPU ist auf mehrere Arten bestimmbar, zum Beispiel als *Floating-Point Operations Per Second* (FLOPS) oder als *Million Instructions Per Second* (MIPS). Aufgrund der sehr heterogenen Hardware in Desktopsystemen sowie der stark veränderlichen und von laufenden Prozessen abhängigen verfügbaren CPU Kapazität, wird auf Metriken wie FLOPS oder MIPS verzichtet. Es werden in den folgenden Betrachtungen nur prozentuale CPU Anteile sowie die Zeit eines Prozesses auf der CPU betrachtet.

Ressourcenbeschränkungen wirken sich auf zwei Arten aus:

I. Blockierend

   Derartige Ressourcenbeschränkungen führen zum Anhalten des Prozesses, da dieser von der jeweiligen Ressource mehr benötigt, als für ihn zur Verfügung steht. In Kapitel 0 wurde unterschieden zwischen funktionalen und nichtfunktionalen Anforderungen von Programmen an Ressourcen. Forderungen nach einem größeren Ressourcenkontingent bei einer den Prozess blockierenden Ressource gehören in die Kategorie der funktionalen Anforderungen.

II. Verzögernd

   Derartige Ressourcenbeschränkungen führen zu einer Verzögerung des Prozesses. Die Ausführzeit ist häufig antiproportional zu der zur Verfügung stehenden Ressourcenkapazität. Forderungen nach einer größeren Kapazität, bei einer den Prozess beschränkenden Ressource, gehören in die Kategorie der nichtfunktionalen Anforderungen. Ausnahmen hiervon sind auftretende Timeouts, die zu einem Abbruch einer beteiligten Komponente führen. In derartigen Fällen ist die Forderung nach größerer Kapazität funktional.

In den folgenden Tabellen werden die genannten Ressourcentypen und –arten in Beziehung gesetzt und es wird bewertet, in wie weit sie für die anschließende Betrachtung relevant sind und welchem Auswirkungstypen sie entsprechen.

[26] *Integrated Services* (IntServ, [40]) ist ein Modell zur netzweiten Garantie von Dienstgüte bezüglich Bandbreite und Verzögerung. Als Protokoll wird RSVP ([41]) verwendet. Aufgrund des entstehenden hohen netzweiten Overheads wird IntServ derzeit kaum verwendet und es ist nicht davon auszugehen, dass es in Zukunft einen hohen Verbreitungsgrad erlangt.

[27] *Differentiated Services* (DiffServ, [03]) ist ein anderes Modell zur netzweiten Garantie von QoS, bei dessen Entwicklung versucht wurde, die Nachteile von IntServ zu vermeiden.

[Kapazität]	Die absolute Festplattenkapazität ist das aufgrund der leichten Messbarkeit sowie der Verständlichkeit das ideale Maß zur Beschränkung von Festplattenkapazität. → relevant, blockierend, Dimension: [Byte]
[Kapazität] * [Zeit]	Die Festplattenkapazität für eine Zeitdauer ist ein sinnvolles Maß zur Beschränkung von Festplattenkapazität, jedoch ergeben sich unter Desktopbetriebssystemen mehrere offene Fragen: Wird für den Anwender aus dem Dateisystem erkennbar, wie viel er von dieser Ressource verbraucht, was passiert nach Aufbrauchen der Ressource (Werden die Daten auf der Festplatte gelöscht?) und wie verhält sich das System bei längeren Auszeiten des Computers. → nur bedingt relevant, blockierend, Dimension: [Bytes] * [s]
[Kapazität] / [Zeit]	Festplattenkapazität pro Zeit, sehr schwer erfassbar und wenig aussagekräftig für reale Szenarien. → nicht relevant

Tabelle 1: Betrachtung der Festplattenkapazitätstypen

[Kapazität]	Die absolute Arbeitsspeicherkapazität ist wie bei der Festplattenkapazität das ideale Maß zur Beschränkung. → relevant, blockierend, Dimension: [Byte]
[Kapazität] * [Zeit]	Die Arbeitsspeicherkapazität für eine Zeitdauer ist sehr schwer messbar und für den Benutzer aus dem System heraus nicht erkennbar. Gleiche Problematik bei Desktopbetriebssystemen wie bei der Festplattenkapazität → nicht relevant
[Kapazität] / [Zeit]	Arbeitsspeicherkapazität pro Zeit, sehr schwer erfassbar und wenig aussagekräftig für reale Szenarien. → nicht relevant

Tabelle 2: Betrachtung der Arbeitsspeicherkapazitätstypen

[Kapazität]	Im Gegensatz zur Festplatten- oder Arbeitsspeicherkapazität ist die Netzwerkkapazität eine Durchsatzressource, bei der lediglich die Betrachtung pro Zeiteinheit möglich ist. → nicht relevant
[Kapazität] * [Zeit]	→ nicht relevant
[Kapazität] / [Zeit]	Netzwerkkapazität pro Zeit, Standardmaß für Netzwerkkapazität. → relevant, verzögernd, Dimension: [Byte] / [s]

Tabelle 3: Betrachtung der Netzwerkkapazitätstypen

[Kapazität]	Die absolute CPU Kapazität ohne zeitliche Dimension ist wenig aussagekräftig für reale Szenarien. → nicht relevant
[Kapazität] * [Zeit]	CPU Kapazität für eine Zeitspanne, definiert wie lange ein Prozess auf der CPU sein kann. → relevant, blockierend, Dimension: [s]
[Kapazität] / [Zeit]	CPU Kapazität pro Zeitspanne. Stehen Metriken wie MIPS oder FLOPS zur Verfügung, so ist die Dimension MIPS bzw. FLOPS, ansonsten wird ein prozentualer CPU Anteil angegeben. → relevant, verzögernd, Dimension: [Takte] / [s] bzw. in %

Tabelle 4: Betrachtung der Prozessorkapazitätstypen

In den beiden folgenden Kapiteln wird für die genannten und als relevant erachteten Ressourcenarten untersucht, inwieweit sie mit systemeigenen und systemnahen Mechanismen partitioniert werden können.

4.2 Ressourcenpartitionierung unter Linux

4.2.14 Festplattenkapazität

Quota

Die Festplattenkapazität kann bei Linux mittels Quota[28] kontrolliert werden. Begrenzungen können für einzelne Benutzer (identifiziert über die UID) oder für Gruppen (identifiziert über die GID) gesetzt werden. Sowohl die Gesamtgröße des von einem Benutzer oder einer Gruppe belegten Festplattenplatzes, als auch die Anzahl der Dateien pro Benutzer oder Gruppe kann begrenzt werden. Für beide Arten von Begrenzungen gibt es jeweils zwei Sorten von Begrenzungen:

1 *Hard Limit*
   Eine *Hard Limit* Begrenzung kann nicht überschritten werden. Alle weiteren Schreibversuche auf dem mit Quota geschützten Mountpoint werden fehlschlagen.

2 *Soft Limit*
   Eine *Soft Limit* Begrenzung kann für eine Zeitspanne (*Grace Period*) temporär überschritten werden.

Quota ist in allen aktuellen Linuxdistributionen enthalten und ist sehr ausgereift. Standardmäßig erfolgt das setzen von Begrenzungen über einen Editor, zur Steuerung über die Kommandozeile kann das Quotatool[29] verwendet werden.

Garantierte Festplattenkapazität

Mit Quota ist es möglich, den verwendeten Festplattenplatz pro Mountpoint zu begrenzen, nicht jedoch Festplattenplatz zu garantieren. Dieses ist mit folgendem Workaround möglich:

1. Erzeugen einer Datei mit der gewünschten Kapazität.
2. Formatieren dieser Datei mit einem Dateisystem (zum Beispiel ext2).
3. Mounten der Datei über ein *Loop Device*, zum Beispiel in ein Homeverzeichnis.

Abzüglich der Verwaltungsbytes ist die gesamte Größe der Datei nutzbar und reserviert. Nachteil dieser Lösung ist allerdings, dass die Größe der Datei auch die maximale Kapazität des Mountpoints definiert.

Beispiel zum Anlegen, Formatieren und Mounten einer Datei (`file.img`) mit 200 Megabyte Größe:

```
# dd if=/dev/zero of=file.img bs=1M count=200
# mk2efs file.img
# mount –t ext2 –o loop file.img /home/foo
```

[28] Der Begriff Quota bezeichnete ursprünglich das Konzept zur Begrenzung der verwendeten Festplattenkapazität, jedoch ist es auch der Name des Linux-Projekts: http://sourceforge.net/projects/linuxquota/
[29] Quotatool Homepage: http://quotatool.ekenberg.se/

4.2.15 Arbeitsspeicher

Zur Begrenzung des Arbeitsspeichers gibt es kein vergleichbares Programm wie Quota, jedoch gibt es verschiedene Ansätze, um Ressourcenkontrolle zu ermöglichen.

Ulimit

Ulimit ist ein in der Bash eingebauter Mechanismus zur Begrenzung von Ressourcen, die von Prozessen, die aus der aktuellen Shell (Bash) gestartet werden, verwendet werden können. Eine der begrenzbaren Ressourcen ist der Speicherverbrauch[30]. Versucht ein Prozess eine mittels Ulimit gesetzte Begrenzung zu überschreiten, so wird ihm dieses verwährt und der Prozess bricht ab. Folgendes ist dabei festzustellen:

- Ulimits müssen in jeder Shell, in der sie wirken sollen, gesetzt werden.
- Ulimits müssen vor dem Prozessstart gesetzt werden, laufende Prozesse können nicht begrenzt werden.
- Ulimits können benutzerspezifisch in der jeweiligen `.profile` oder für alle Benutzer gleich in `/etc/.profile` definiert werden, jedoch haben sie jeweils nur Auswirkungen pro Shell, d.h. die Summe des Ressourcenverbrauchs, mehrerer über verschiedene Shells gestarteter Prozesse, ist nicht limitiert.
- Per Cronjob gestartete Prozesse werden, wie experimentell herausgefunden wurde, nicht durch in `.profile` Dateien gegebene Ulimit-Beschränkungen begrenzt. Um hier eine obere Schranke zu realisieren, muss das aufgerufene Binary in einer eigenen Shell laufen, d.h. über ein Shellskript, in dem Ulimit-Begrenzungen gesetzt sind, gestartet werden.
- Die mittels Ulimit gesetzten Begrenzungen können nicht zur Laufzeit des Prozesses verändert werden.
- Eigene Messungen haben ergeben, dass beim Forken eines mit Ulimit-Begrenzungen aufgerufenen Programms anschließend beide Prozesse die gleiche unmodifizierte Begrenzung erfahren, sodass sich effektiv der maximale Ressourcenverbrauch verdoppelt hat. Sich forkende Programme sind folglich mit Ulimit nicht kontrollierbar.

PAM Limits

Eine weitere Möglichkeit zur Begrenzung des Speicherverbrauchs ist die Verwendung der *Pluggable Authentication Modules* ([29], Kapitel 6.12). Das PAM-System ist auf den meisten aktuellen Distributionen aktiv und bietet unter Anderem die Möglichkeit, den Ressourcenverbrauch eines Benutzers bzw. der Prozesse eines Benutzers zu beschränken. Konfiguriert werden die Beschränkungen in der Datei `/etc/security/limits.conf`[31]. Wie bei Ulimit existieren harte und weiche Begrenzungen und es sind ähnliche Beobachtungen festzustellen:

- Begrenzungen gelten pro Benutzerlogin. Es gibt keinen Mechanismus, der Begrenzungen pro Benutzer (der mehrfach angemeldet ist) insgesamt realisiert.

[30] Mit Ulimit ist es möglich den virtuellen Speicher und den physikalischen Speicher zu begrenzen.
[31] Eine Beispielkonfiguration der Datei: ftp://ftp.lugmoe.de/lugmoe/downloads/configs/limits.conf

- Zur Laufzeit von Prozessen können Begrenzungen nicht modifiziert werden.
- Per Cronjob gestartete Prozesse werden nachweislich nicht von diesem Mechanismus erfasst.
- Es ist lediglich möglich, den physikalischen Speicher zu begrenzen nicht jedoch den virtuellen Speicher.
- Geforkte Prozesse erben nicht die Begrenzung des Mutterprozesses sondern die Begrenzung gilt (im Gegensatz zum Verhalten bei Ulimit) für beide Prozesse in Summe.

Fazit

Die Möglichkeit den Arbeitsspeicher zu beschränken ist nur begrenzt gegeben. Besonders störend ist, dass zur Laufzeit eines Prozesses die Begrenzung nicht verändert werden kann. Geht man im Grid-Kontext davon aus, dass auf jedem Knoten ein Dämon, der die eigentlichen Jobs startet, läuft, so ergibt sich sowohl bei Ulimit als auch bei PAM, dass dieser Dämon neu gestartet werden muss, um die Begrenzung zu verändern. Bei beiden Ansätzen gibt es ferner keine Möglichkeit, geforkten Prozessen separate Begrenzungen zukommen zu lassen. Insgesamt ist festzuhalten, dass beide Ansätze nur teilweise zufrieden stellen können. Arbeitsspeicher zu reservieren ist in beiden Fällen nicht möglich und es gibt auch außerhalb dieser beiden Ansätze keine praktikable Möglichkeit, dieses auf normalen Desktop-Linux-Systemen zu tun.

4.2.16 Netzwerkbandbreite

Traffic Shaping mit iproute2

In Linux werden diverse Netzwerkfeatures durch das iproute2-Paket[32] realisiert, darunter NAT, RSVP und *Differentiates Services*. Des Weiteren ist es mittels iproute2 möglich, Netzwerkbandbreitenpartitionierung durchzuführen, d.h. die verwendete Bandbreite zu begrenzen aber auch zu garantieren. Die Partitionierung kann für beliebige Datenströme durchgeführt werden. Bedingung ist, dass der jeweilige Datenstrom von anderen unterscheidbar ist und herausgefiltert werden kann. Einfache Filter richten sich nach dem verwendeten Protokoll, nach der Sender- / Empfängeradresse oder nach Sender- / Empfängerport. Mittels iptables können Pakete nach diversen Kriterien gefiltert und anschließend markiert werden. Diese Paketmarkierung kann wiederum über iproute2 herausgefiltert werden, sodass alle Filtermechanismen von iptables zur Klassifizierung von Datenströmen verwendet werden können. Iptables ermöglichen das Filtern nach folgenden Kriterien:

- Diversen Feldern im TCP Header
- Diversen Feldern im UDP Header
- ICMP Type
- MAC Source Adress
- Identifizierung des Paketabsenders anhand folgender Charakteristik:
    - User ID (UID)
    - Group ID (GID)
    - Process ID (PID)

[32] iproute2 Homepage: http://developer.osdl.org/dev/iproute2/

- Session ID (SID)

In iproute2 sind verschiedene Scheduler und *Queuing Disciplines* zur Behandlung von Netzwerkpaketen implementiert. Bandbreitenbegrenzung aber auch Bandbreitengarantien lassen sich mit *Classful Queueing Disciplines* erwirken. Es werden hierbei alle Pakete anhand von Filterkriterien in vorgegebene Klassen sortiert, denen jeweils eine Datenrate zugeordnet ist. Diese Datenrate ist zum einen eine garantierte Datenrate, zum anderen kann sie (je nach Konfiguration) auch als Maximum wirken. *Traffic Shaping* mit iproute2 ist nicht unbedingt trivial, sondern aufgrund der inneren Mechanismen und der Syntax relativ komplex.

Für technische Details wird auf [22] verwiesen[33], hierbei besonders auf das Kapitel 9 (*Queueing Disciplines for Bandwidth Management*), darin auf Kapitel 9.5 (*Classful Queuing Disciplines*) sowie auf Kapitel 12 (*Advanced filters for (re-)classifying packets*). Eine Einführung in die Thematik bietet der „Traffic Shaping mit Linux" Artikel im Linux Enterprise Magazin, Ausgabe Juli 2004 ([09]).

In Kapitel 7.3 erfolgt eine Evaluierung der *Traffic Shaping* Fähigkeiten von iproute2.

Differentiated Services

In Kapitel 4.1 wurde kurz auf die Mechanismen zur netzwerkweiten Bandbreitenkontrolle eingegangen. Bei *Differentiated Services* ([03]) werden alle Pakete im IP-Header, im *Type of Service* (TOS) Feld markiert und anhand dieser Markierung, dem *Differentiated Services Code Point* (DSCP), erfahren die Pakete unterschiedliche Behandlung im Netz seitens der beteiligten Netzelemente. Im Allgemeinen erfolgt die Markierung auf dem ersten aktiven Netzelement längs der Verbindung, es ist jedoch auch möglich, bereits auf Senderseite die Pakete zu markieren, dieses erfolgt unter Linux über iptables. Die Möglichkeiten zum Filtern der Pakete sind hierbei die gleichen, wie bei der Kombination mit iproute2. Gefilterte Pakete können anschließend mit einem DSCP versehen werden. In diesem Beispiel werde alle Pakete des Users `test1` mit `42` versehen:

```
iptables –t mangle –A OUTPUT –m owner –uid-owner 501 –j DSCP –set-dscp 42
```

Der User kann dabei entweder über die UID oder über den Namen identifiziert werden, ebenso kann der DSCP-Wert in Dezimal oder Hexadezimal angegeben werden.

4.2.17 CPU Kapazität

Wie in Kapitel 4.1 dargestellt, kann CPU Kapazität über zwei Ansätze beschränkt werden: Zum einen über die maximale Zeit, die ein Prozess auf der CPU arbeitet und zum anderen über einen prozentualen Wert, den ein Prozess maximal an Leistung innerhalb einer Zeitspanne verbrauchen darf. Ersteres wird in Kapitel 0 analysiert. Die zweite Begrenzungsart kann nur über einen Scheduler erreicht werden, jedoch ist in Linux kein CPU Scheduler integriert, der dieses ermöglicht. In den Kapiteln 0, 0 und 0 werden drei Scheduler in Hinblick auf diese Fähigkeit untersucht.

Begrenzung der maximalen Laufzeit mit PAM oder Ulimit

Sowohl mittels PAM als auch über Ulimit[34] kann die maximale Rechenzeit eines Prozesses begrenzt werden, es ergeben sich bei beiden Mechanismen die gleichen

[33] Zum Verständnis dieser Arbeit ist es jedoch nicht notwendig, die angegebene Literatur zu bemühen.

[34] Beide Mechanismen wurden bereits in den Kapiteln 0 und 0 auf ihre Fähigkeiten zum Begrenzen des verwendeten Arbeitsspeichers untersucht.

Nachteile, wie auch bei der Begrenzung des Arbeitsspeichers. Allgemein ist die Begrenzung der Rechenzeit im Kontext von Grid Computing zweifelhaft, wenn auf jedem Client ein zeitlich unbegrenzter Dämon laufen muss. Auch wenn dessen Verbrauch an Prozessorleistung nur sehr gering ist, so lässt sich keine obere Grenze definieren. Folglich bietet es sich an, nur die von dem Dämon gestarteten Jobs zu begrenzen. Dies hieße allerdings, dass sie zumindest bei Ulimit in ein Shellscript gekapselt werden müssten. Die in das Grid eingespeisten Jobs sollten betriebssystemunabhängig sein, das Kapseln der Jobs muss folglich vom Grid-System erfolgen. Dieses zu realisieren, würde einen Eingriff in die Mechanismen des Grid-Clients bedeuten, was nicht wünschenswert ist. Insgesamt sind beide Möglichkeiten nicht praktikabel, denkbar wäre dagegen die Entwicklung eines eigenen Programms, das über `/proc`, `ps` oder `top` die Laufzeit der zu überwachenden Prozesse ermittelt und bei Bedarf die jeweiligen Prozesse über `kill` beendet.

Cap Processor Usage

Cap Processor Usage[35] ist ein Kernel-Patch, der den internen CPU Scheduler dahingehend modifiziert, dass es möglich ist, Prozessen eine obere Begrenzung des relativen CPU Verbrauchs zuzuweisen. Die Zuweisung erfolgt mit dieser Syntax:

```
$ cap <task pid> <CPU percentage>
```

Eigene Messungen haben ergeben, dass es bei mehreren Prozessen mit gleicher Priorität nicht möglich ist, über diesen Mechanismus einen höheren CPU Anteil zu garantieren, sondern der Prozess erhält wie alle anderen $^1/_N$ % CPU Leistung (sofern $^1/_N$ kleiner als die angegebene Begrenzung ist). Mit *Cap Processor Usage* ist folglich nicht möglich, Garantien zu erreichen, sondern ausschließlich Beschränkungen können gesetzt werden. Forkt sich ein Prozess, so haben anschließend beide Prozesse die gleiche Beschränkung und in Summe hat sich der maximale CPU Verbrauch verdoppelt. Besonders problematisch erweist sich hier, dass, wenn die Summe der vergebenen Prozessoranteile 100 übersteigt, der Scheduler nicht mehr korrekt arbeitet und Begrenzungen nicht mehr realisiert. Auch nach Beendigung eines oder mehrerer Prozesse, sodass in Summe nicht mehr als 100% vergeben sind, arbeitet der Scheduler nicht länger korrekt und erst nach Neustart des Rechners können wieder verlässliche Begrenzungen erreicht werden. Um mit *Cap Processor Usage* Garantien zu erreichen, müssten alle anderen laufenden Prozesse begrenzt werden. Hierzu wäre ein Programm notwendig, das alle gerade aktiven Prozesse überwacht und Begrenzungen setzt. Außerdem wäre ein Programm notwendig, um sich forkende Prozesse zu überwachen und die vergebene Beschränkung nach dem Forken auf die beiden Kinder zu beschränken, bzw. um nach der Beendigung eines Kindes dem anderen Prozess wieder den gesamten Anteil zukommen zu lassen.

Wie bereits erwähnt, handelt es sich bei *Cap Processor Usage* um einen Kernel-Patch, dieser wird allerdings nicht mehr gewartet und gepflegt. Patches sind nur bis Kernel-Version 2.4.19 vorhanden, neuere Kernels, sei es 2.4.20 oder die 2.6er Reihe werden nicht unterstützt. Getestet wurde *Cap Processor Usage* mit Kernelversion 2.4.18. Letztendlich muss noch erwähnt werden, dass ein Kernel-Patch keine ideale Lösung im Kontext von *Grid Computing* ist. Der Aufwand, einen Kernel-Patch auf allen verwendeten Knoten anzuwenden, wäre extrem hoch und es ist unwahrscheinlich, dass dieses angewendet werden würde.

[35] *Cap Processor Usage* Homepage: http://www.tls-technologies.com/CPU/cpu-intro.html

DSRT

Der *Dynamic-Soft-Real-Time Scheduler*[36] ist ein Projekt an der Universität von Illionois unter Leitung von Klara Nahrstedt. Auf der Projekthomepage wird DSRT wie folgt beschrieben:

> *The dynamic soft real time scheduler (DSRT) 2.0 is a user-level dynamic scheduler that can schedule soft real time (RT) periodic and aperiodic tasks.*

Im Gegensatz zu *Cap Processor Usage* ist DSRT weder ein Kernelpatch noch muss er in den Kernel kompiliert werden, sondern wird im normalen Benutzermodus von root gestartet. Mit dem Scheduler ist es möglich, Prozessen *soft real-time* Garantien zu gewährleisten aber auch Prozessen bestimmte Anteile der CPU Leistung (in %) zuzuweisen. Der Scheduler ist sowohl für Solaris und Linux als auch für Microsoft Windows NT vorhanden, allerdings ist die Projekthomepage seit Oktober 2000 nicht mehr verändert worden und nach Aussagen von Klara Nahrstedt gibt es keine Weiterentwicklung. Das Installieren von DSRT gestaltet sich als sehr komplex und ist augrund der inzwischen veralteten Ausgangslage auf aktuellen Systemen ohne aufwendiges Nachinstallieren diverser Bibliotheken nicht möglich.

Solaris Scheduling Server

Der *Solaris Scheduling Server* (SOSS, [38]) ist ein auf Forschungsarbeiten von Andreas Polze beruhender und von Peter Tröger entwickelter CPU Scheduler zur Partitionierung der Prozessorkapazität. Treibende Kraft bei der Entwicklung ist der Einsatz von *Idle Time Computing* auf Workstations. Der Scheduler ermöglicht, einem Prozess feste Garantien zu geben, während Hintergrundlast, z.B. seitens *Grid Computing*, diesen Prozess nicht beeinflusst, sondern die nicht vergebene Prozessorkapazität verwendet. Wie beim DSRT können mit SOSS *soft real-time* Garantien realisiert werden. Insgesamt sind Ansatz und Anwendungsszenarien vergleichbar mit denen von DSRT, jedoch ist der SOSS in seiner derzeitigen Form auf einen kontrollierbaren Thread beschränkt und somit noch nicht geeignet für eine allgemeine Partitionierung der CPU Kapazität. Eine Portierung von Solaris auf Linux ist noch nicht durchgeführt worden.

Fazit

Derzeit ist in Linux keine umfassende Lösung zur Partitionierung von CPU Kapazität vorhanden und es existiert auch keine systemnahe Lösung, die die gewünschten Anforderungen realisiert. Die vorgestellten Scheduler sind nicht dazu geeignet, auf Arbeitsplatzrechnern unter einfachem Installationsaufwand und mit gewünschter Präzision ihre Arbeit zu verrichten. Die Begrenzung der insgesamt verbrauchten CPU Zeit ist, wie bereits diskutiert, zwar möglich, allerdings nur unter den oben aufgeführten Einschränkungen. Zusammenfassend kann festgestellt werden, dass ohne die Weiterentwicklung einer der bestehenden Lösungen oder einer Neuentwicklung, keine sinnvolle und anwendungsorientierte CPU Partitionierung möglich ist.

[36] DSRT Homepage: http://cairo.cs.uiuc.edu/software/DSRT-2/dsrt-2.html

4.3 Ressourcenpartitionierung unter Microsoft Windows XP

4.3.18 Festplattenkapazität

Wie unter Linux so ist auch unter Windows XP ein Quota Mechanismus für NTFS Laufwerke integriert, jedoch kann nur der verwendete Speicherplatz begrenzt werden, nicht die Anzahl der Dateien. Ebenfalls wie unter Linux gibt es eine Unterscheidung zwischen harten und weichen Begrenzungen, wobei weiche Garantien nur eine Benachrichtigungsschwelle sind und nicht wie in Linux mit einer *Grace Period* kombiniert werden können. Quotas sind in Windows XP auf zwei verschiedene Arten setzbar: Zum einen im Explorer über das Kontextmenü des jeweiligen Laufwerks sowie zum anderen über das Kommandozeilentool `fsutil`. Die betriebssysteminterne WMI Schnittstelle für Quota, der *Disk Quota Provider*, ist jedoch nur in Windows 2000 Server Edition, nicht jedoch in Windows XP vorhanden, hierzu [26]:

> *Windows XP and earlier: The Disk Quota provider is not available.*

Einen Mechanismus zum Reservieren bzw. Garantieren von Festplattenkapazität gibt es unter Windows XP nicht.

4.3.19 Arbeitsspeicher

Seit Windows 2000 ist es möglich, den Ressourcenverbrauch von Prozessen in Windows über *Job Objects* zu kontrollieren. *Job Objects* werden in [27] erklärt:

> *A job object allows groups of processes to be managed as a unit. Job objects are nameable, securable, sharable objects that control attributes of the processes associated with them. Operations performed on the job object affect all processes associated with the job object.*

Job Objects können über eine C++ API erzeugt und konfiguriert werden. Folgendes Vorgehen zur Partitionierung ist notwendig:

1. Anlegen eines neuen *Job Objects* mit `CreateJobObject`
2. Zuweisen von Prozessen mit `AssignProcessToJobObject` zu dem *Job Object*
3. Limitieren des *Job Objects* mit `SetInformationJobObject`

Mittels *Job Objects* können sowohl der virtuelle Arbeitsspeicher als auch die Prozessorzeit limitiert werden, jedoch können jeweils keine Garantien mit *Job Objects* erzeugt werden. Es ist allerdings möglich, die *working size*, d.h. den physikalischen Speicher, sowohl zu garantieren als auch zu limitieren.

Neben der C++ API existiert in Windows XP keine andere Möglichkeit, auf Job Objects zuzugreifen. In Microsofts *Windows 2000 Datacenter Server* können mit *Process Control*[37] so genannte *Process Execution Rules* und *Process Group Execution Rules* erzeugt werden, diese partitionieren entweder Prozesse oder *Process Groups*. Die *Execution Rules* zu setzen ist entweder über ein graphisches Interface oder über über das Kommandozeilentool `proccon` möglich. Lediglich die Kommandozeilenvariante ist neben der C++ API für ein Framework eine praktikable Zugriffsmöglichkeit, jedoch ist `proccon` nach [25] nicht in Windows XP enthalten. Folglich können in Windows XP nur über die erwähnte C++ API die gewünschten Partitionierungen gesetzt werden.

[37] Eine ausführliche Beschreibung der *Process Control* Möglichkeiten unter *Windows 2000 Datacenter Server* kann [24] entnommen werden.

4.3.20 Netzwerkbandbreite

Eine Netzwerkbandbreitenbeschränkung- oder partitionierung wie unter Linux ist mit Windows XP nicht möglich. Integrierte QoS Mechanismen setzen auf RSVP und sind damit keine stand-alone Lösung. Auch der im Internet häufig erwähnte QoS Paketplaner kann nicht dazu verwendet werden, den Datenstrom von Prozessen zu beschränken. Es besteht die Möglichkeit, dass Prozesse mit der in Windows integrierten QoS API kommunizieren und dieser API den zu kontrollierenden Datenstrom übergeben, jedoch kann von außerhalb der Datenstrom nicht kontrolliert werden, lediglich ein direktes Ansprechen der API (innerhalb des Programms) ermöglicht dieses. Es ist somit keine geeignete Lösung zur Bandbreitenpartitionierung für Grid-Jobs gegeben.

4.3.21 CPU Kapazität

Wie bereits in Kapitel 4.3.19 beschrieben, existiert in Windows XP die Möglichkeit, über *Job Objects* den Ressourcenverbrauch von Prozessen zu limitieren, womit auch die Begrenzung der maximalen CPU Zeit eines Prozesses bzw. eines *Job Objects* vorhanden ist. Die Limitierung des relativen CPU Anteils ist auch möglich, allerdings kann kein festes Quantum eingestellt werden, sondern lediglich die Scheduling-Klasse für das *Job Object* kann festgelegt werden, [28]:

> *If […], this member is the scheduling class for all processes associated with the job. […] The valid values are 0 to 9. Use 0 for the least favorable scheduling class relative to other threads, and 9 for the most favorable scheduling class relative to other threads. By default, this value is 5. To use a scheduling class greater than 5, the calling process must enable the SE_INC_BASE_PRIORITY_NAME privilege.*

Es ist folglich keine Möglichkeit zur exakten Festlegung des Quantums bzw. des prozentualen CPU Anteils in Windows XP gegeben. Auch sind keine ausgereiften und mit der entsprechenden Funktionalität ausgestatteten sowie für den breiten Einsatz geeigneten Scheduler existent. Unter funktionalen Gesichtspunkten betrachtet, sind sowohl der DSRT (bereits in Kapitel 0 behandelt) sowie der NTSS (Windows NT Portierung vom SOSS, Kapitel 0) denkbar, doch beide Scheduler sind, wie bereits geschildert, derzeit nicht für den praktikablen Einsatz geeignet.

4.4 Fazit

Mechanismen zur Ressourcenpartitionierung sind sowohl in Windows XP und Linux vorhanden, jedoch werden von beiden Betriebssystemen nicht alle Ressourcen unterstützt. Für Linux ist neben den systeminternen Mechanismen ein größeres Spektrum an externen Lösungen vorhanden, während Windows mit den *Job Objects* eine C++ API anbietet, die es ermöglicht, ohne Kommandozeilenaufrufe sondern direkt über diese Schnittstelle Partitionierungen vorzunehmen. Nachteilig hieran ist jedoch, dass die Verwendung einer bestimmten Programmiersprache (zumindest für eine Kapselung) erzwungen wird. Aus den durchgeführten Untersuchungen ergibt sich die folgende Tabelle von Ressourcenpartitionierungsmöglichkeiten für die beiden untersuchten Betriebssysteme:

		Linux	Windows XP
Festplattenkapazität	Limitieren	möglich	möglich
	Garantieren	bedingt möglich	nicht möglich
Dateianzahl limitieren		möglich	nicht möglich
Arbeitsspeicher	Limitieren	bedingt möglich	möglich
	Garantieren	nicht möglich	bedingt[38] möglich
Netzwerkbandbreite	Limitieren	möglich	nicht[39] möglich
	Garantieren	möglich	nicht möglich
DSCP setzen		möglich	nicht möglich
CPU Zeit limitieren		bedingt möglich	möglich
CPU Anteil	limitieren	bedingt[40] möglich	bedingt[41] möglich
	garantieren	nicht möglich	bedingt möglich

Tabelle 5: Ressourcenpartitionierungsmöglichkeiten unter Linux und Windows XP

[38] Wie in Kapitel 4.3.19 beschrieben, können Mindestanforderungen für den physikalischen Speicher gesetzt werden, nicht jedoch für den virtuellen.

[39] Unter Windows XP ist es teilweise möglich Netzwerkbandbreite zu partitionieren, jedoch nur über RSVP, welches andere, ebenfalls RSVP unterstützende Netzwerkkomponenten erfordert oder unter Verwendung spezieller APIs. In beiden Fällen ist keine gewünschte Partitionierungsmöglichkeit gegeben.

[40] Experimente haben gezeigt, dass *Cap Processor Usage* (Kapitel 0) die vorgenommene Begrenzung immer realisiert, sofern das Programm nicht durch eine zu hohe Gesamtsumme an vergebnen Prozessoranteilen ‚verwirrt' wird. Die Einsetzbarkeit ist jedoch aufgrund des benötigten Kernel-Patches nur bedingt gegeben.

[41] Es ist möglich, die Scheduling-Klasse eines *Job Objects* bzw. der enthaltenen Prozesse festzulegen, jedoch können damit keine absoluten Angaben (CPU Anteil, Quantum) gesetzt werden.

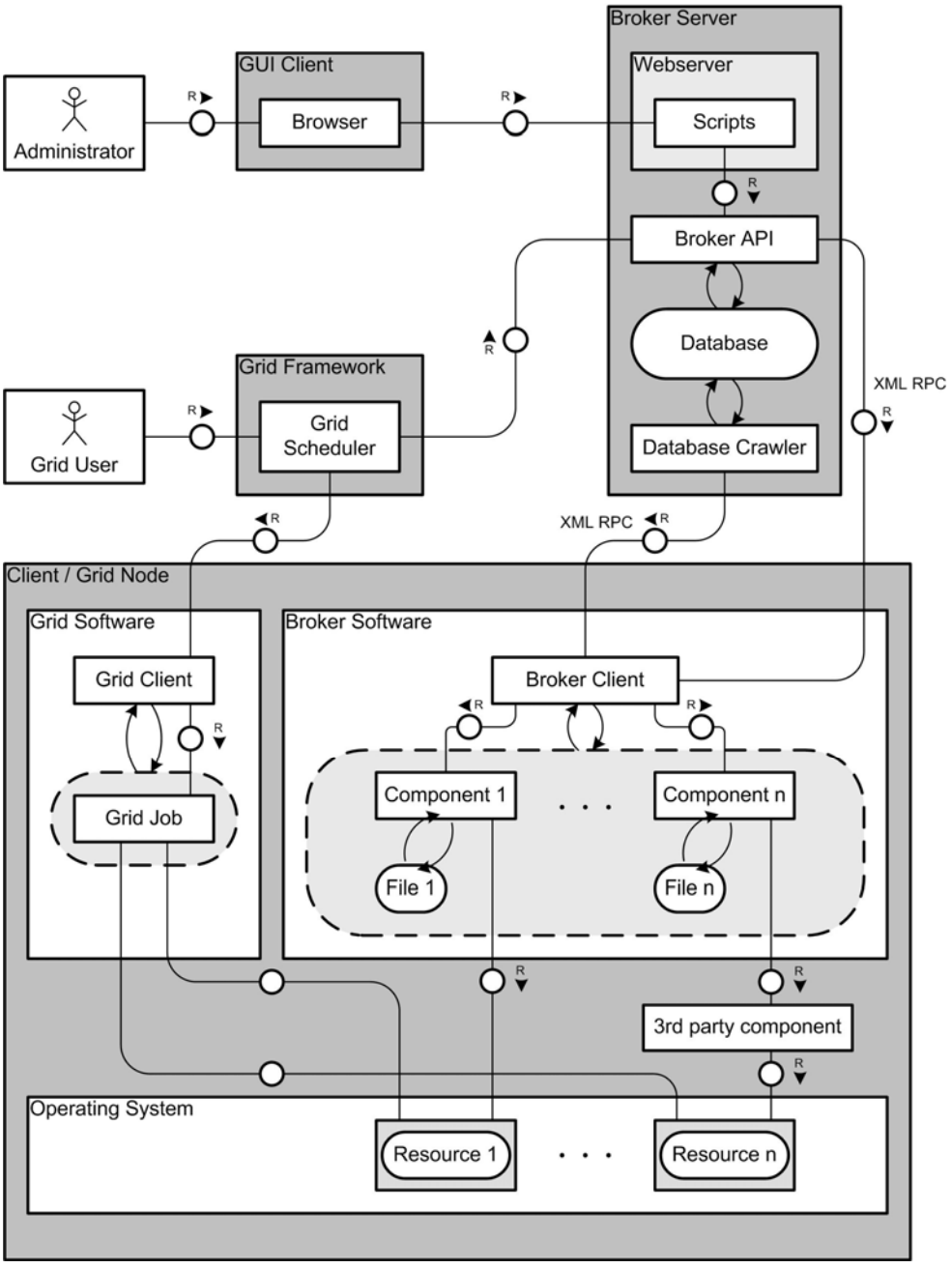
5 Entwurf und Design des Frameworks

Kapitel 4 hat die prinzipielle Möglichkeit zur Ressourcenpartitionierung auf Desktopbetriebssystemen aufgezeigt. Sofern noch keine systemeigene Lösung vorliegt, besteht die Gegebenheit, externe Lösungen zu verwenden. Sind auch derzeit nicht für alle Ressourcen Mechanismen zur Partitionierung vorhanden, so ermöglichen es die bestehenden, ein Framework zu realisieren, mit dem viele der in Kapitel 3.2 aufgezeigten Anwendungsszenarien gelöst werden können.

5.1 Entwurfsgrundlage - Komponentenarchitektur

Dieses Kapitel beschreibt die Grundlage, auf der die Architektur des Frameworks entwickelt wurde. Zentrales Konzept hierbei ist die Unabhängigkeit des Frameworks von den einzelnen Ressourcen. Das Framework soll die Möglichkeit bieten, beliebige Ressourcen auf beliebigen Knoten zu partitionieren, ohne dass das Framework Kenntnisse über die Ressourcen und ihre spezifischen Eigenschaften haben muss. Diese Forderung gründet sich aus der Tatsache, dass nicht absehbar ist, welche Ressourcen zukünftig mit Hilfe des Frameworks verwaltet werden und dass somit eine größere Flexibilität des Frameworks erreicht wird. Auch wird hiermit eine zu große Spezialisierung einzelner Akteure auf die jeweiligen Ressourcentypen verhindert. Grundsätzlich handelt es sich um eine Client-Server Architektur, bei der ein zentraler Server Ressourcenpartitionierung auf den von ihm verwalteten Knoten (Clients) durchführt. Zur Realisierung der Trennung zwischen Funktionalität und Ressourcenart, wird sowohl auf Server- als auch auf Clientseite eine Komponentenarchitektur verwendet. Hierbei entspricht eine Komponente einer partitionierbaren Ressource. Serverseitig erfolgt die gesamte Datenhaltung in einer Datenbank. Um im Datenbankdesign die Unabhängigkeit von Ressourcenarten zu erhalten, existiert für jede partitionierbare Ressource eine Komponentendefinition innerhalb der Datenbank. Diese Komponentendefinition stellt alle Informationen zum Partitionieren der spezifischen Ressource bereit. Daneben werden innerhalb der Datenbank die Knoten (Clients) verwaltet, sowie die Zuordnung von Komponenten und Knoten und die eigentlichen Reservierungen bzw. Beschränkungen. Die Zuordnung zwischen Komponenten und Knoten gibt hierbei an, welche Ressourcen auf dem jeweiligen Knoten partitionierbar sind. Auf Clientseite ermöglicht die Komponentenarchitektur einen modularen Aufbau, der im Idealfall dazu führt, dass Komponenten zur Laufzeit des Basisclients, welcher für die Kommunikation mit dem Server zuständig ist, nachgeladen werden können. Das Kommunikationsprotokoll zwischen Server und Client ist ebenfalls ressourcenunabhängig.

5.2 Architektur



Abbildung[42] 5: Frameworkarchitektur

[42] Diagrammtyp: FMC Block Diagramm. Informationen zur Notation der drei FMC Diagrammtypen können der FMC Homepage entnommen werden: http://fmc.hpi.uni-potsdam.de/

Abbildung 5 veranschaulicht die Architektur des Frameworks im Kontext von *Grid Computing*. Zu sehen ist zum einen das Framework, bestehend aus dem Server (*Broker Server*) sowie der Client Software (*Broker Software*), zum anderen sind Elemente des Grids abgebildet, darunter der *Grid Scheduler* sowie die auf den Clients laufende Grid Software, bestehend aus dem *Grid Client* sowie einem möglichen *Grid Job*[43]. Während der vom Grid auf den Client gestartete Job als Ressourcenverbraucher agiert, ist der Zugriff des Frameworks auf die Ressourcen partitionierend, d.h. durch das Framework werden die verschiedenen Ressourcen partitioniert. Gemäß den Anforderungen erfolgt der Zugriff auf das Framework von administrativer Seite und es existiert eine Schnittstelle zum Grid. Durch diese beiden Zugriffsmöglichkeiten werden die in Kapitel 0 und 0 gezeigten Anwendungsfallgruppen realisiert.

Im Folgenden werden die einzelnen Bestandteile des Frameworks sowie die Schnittstellen unter ihnen genauer beschrieben.

5.2.22 Broker Server

Datenbank

Innerhalb der Datenbank werden, wie bereits erwähnt, nicht nur Knoten und ihre entsprechenden Partitionierungsinformationen gespeichert, sondern zusätzlich, um Unabhängigkeit von den einzelnen Ressourcen zu erreichen, Informationen zu den jeweiligen Komponenten. Eine detaillierte Beschreibung der Datenbankstruktur erfolgt in Kapitel 5.5 .

Broker API

Ein Zugriff auf die Datenbank sollte nur erfolgen, wenn das notwendige Wissen für den Umgang mit den dort gespeicherten Informationen mitgebracht wird. Hierzu dient die Broker API, die nicht als reine Schnittstelle zu verstehen ist, sondern welche die über das Komponentenmodell[44] definierten und in der Datenbank gespeicherte Komponenteninformationen interpretiert und daraus die Logik bezieht und diese nach außen abbildet. Neben der Anbindung der für die Administration zuständigen Webskripte ist die Schnittstelle primär darauf ausgelegt, Zugriff seitens eines Grid-Schedulers zu ermöglichen. Hierzu muss die API alle notwendigen Methoden zur Informationsabfrage, zum Reservieren, zum Ändern und zum Aufheben von Reservierungen bereitstellen. Reservierungen und Vergleichbares werden von der Broker API nicht direkt durchgeführt, sondern in der Datenbank gespeichert. Die Interaktion der Broker API mit den Clients ist auf das Abfragen bestimmter nicht in der Datenbank gespeicherter Daten beschränkt. Was das für Daten sein können, wird innerhalb der Datenbank über die Komponentendefinitionen spezifiziert, denkbar wären zum Beispiel aktuelle Auslastungsinformationen oder die Übermittlung von vorhandenen Mountpoints beim Setzen von Quota-Beschränkungen.

Database Crawler

Aufgrund der Tatsache, dass Clients nicht erreichbar sein können, sei es weil sie ausgeschaltet sind oder die Clientsoftware nicht läuft, ist es nicht möglich, einen direkten Zugriff der Broker API auf die Clients für relevante Transaktionen zu

[43] Es handelt sich hierbei um eine vereinfachte, schematische Darstellung des Grid-Systems. Je nach verwendeter Grid-Software müssen weitere Komponenten hinzugefügt werden.

[44] Die Erläuterung des Komponentenmodells erfolgt in Kapitel 5.3 .

ermöglichen. Stattdessen wird beim Datenbankdesign[45] berücksichtigt, dass Clients nicht erreichbar sein können und eine spätere Synchronisierung von Reservierungen möglich sein muss. Die Synchronisierung wird vom Database Crawler durchgeführt, es ist neben der Broker API die einzige Komponente, die einen direkten Zugriff auf die Datenbank hat.

Webserver Skripte

Zur Administration des Frameworks und zur Durchführung von administrativen Beschränkungen scheint eine webbasierte Lösung angebracht. Es vermeidet die Entwicklung von proprietären Clients und ermöglicht eine netzweite Erreichbarkeit. Die Bezeichnung „Webserver Skripte“ muss hierbei nicht bindend sein, denkbar sind viele Lösungen von Applets über Java *Server Pages* bis hin zu Skriptlösungen mit Perl. Es ist irrelevant wo der Webserver läuft, der Einfachheit halber befindet er sich im Architekturmodell auf demselben Rechner wie der Broker Server, d.h. wie die Datenbank.

5.2.23 Broker Software auf den Clients

Der Entwurf spezifiziert nicht das Kommunikations- und Architekturmodell zwischen Broker Client und den einzelnen Komponenten. Für jeden Client kann hier eine eigenständige Lösung realisiert werden, da dies keine Auswirkung auf das Gesamtsystem hat.

Broker Client

Der Broker Client dient als Kommunikationspartner des Broker Servers und verwaltet die einzelnen Brokerkomponenten, das heißt er steuert sie an und leitet Nachrichten des Broker Servers an sie weiter. Darüber hinaus hat er keine Funktionalität.

Komponenten

Jede Komponente ist zuständig für die Partitionierung einer Ressource. Es lassen sich zwei Fälle unterscheiden:

- Persistente Partitionierungen bleiben beim Neustart des Systems bestehen, folglich ist beim Start der Komponente kein Handlungsbedarf vorhanden.
- Transiente Partitionierungen bleiben im Falle eines Neustarts nicht bestehen, sondern sie müssen von der Komponente erneut gesetzt werden, wenn die Komponente nach einem Systemstart erstmalig gestartet wird.

Komponenten, die zur zweiten Kategorie gehören, müssen clientseitig Daten speichern, um unabhängig vom Broker Server beim Systemstart die gewünschte Partitionierung herstellen zu können. Dieses Verhalten wird in der Architekturabbildung durch die zu jeder Komponente gehörende Datei veranschaulicht.

Wie beschrieben, werden systemeigene oder fremde Lösungen zur Ressourcenkontrolle verwendet, dementsprechend kommunizieren die Komponenten entweder direkt mit dem Betriebssystem oder mit einem Programm dritter Seite (in Abbildung 5 „3rd party component“ genannt).

[45] Das Datenbankdesign wird in Kapitel 5.5 vorgestellt.

5.2.24 Schnittstellen

Administration ↔ Broker Server

Der administrative Zugriff soll, wie bereits in Kapitel 0 beschrieben, über eine webbasierte Lösung realisiert werden. Als Kommunikationsprotokoll wird dementsprechend HTTP oder HTTPS verwendet.

Grid ↔ Broker Server

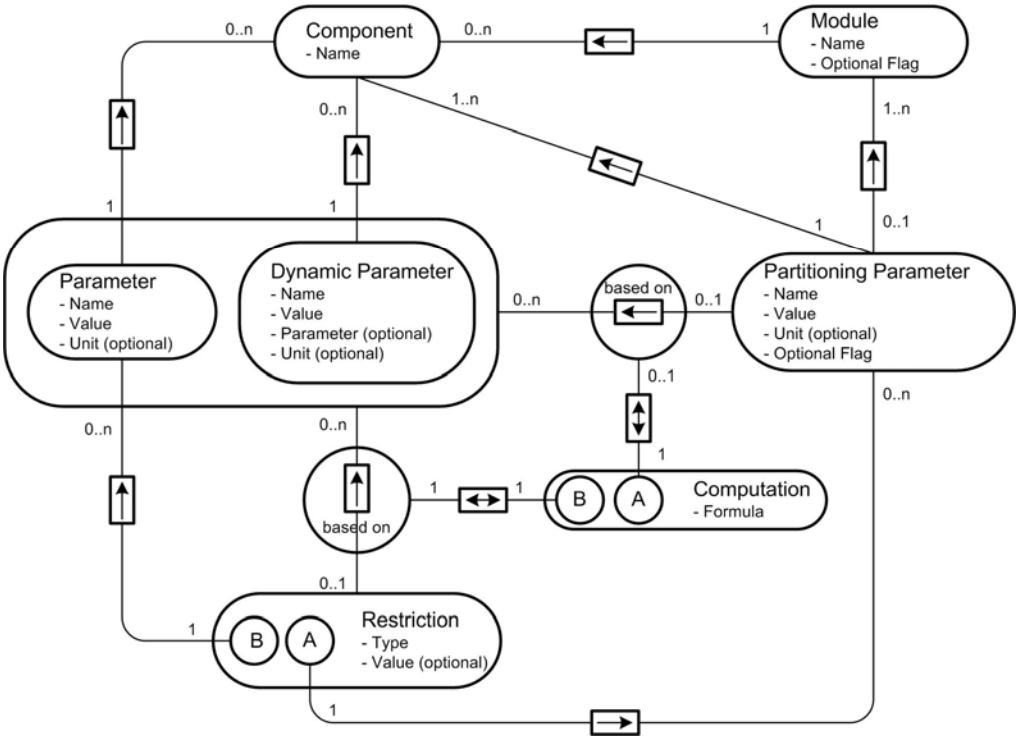
Als Schnittstelle bietet sich die in Kapitel 3.3.3 erwähnte *Resource Specification Language* (RSL) an. Die Analyse der Anbindung des Frameworks an ein Grid erfolgt in Kapitel 5.5 .

Broker Server ↔ Clients

Als Kommunikationsprotokoll für die Schnittstelle zwischen Broker Server und den Clients wird XML RPC verwendet. In Kapitel 5.7 erfolgt eine Betrachtung der Motivation, diese Technologie zu verwenden sowie eine detaillierte Beschreibung der einzelnen, für das Framework definierten, Funktionen.

5.3 Komponententypmodell

Um eine datenbankgestützte Speicherung der Komponentendefinition zu ermöglichen, bedarf es eines Metamodells, dem alle Komponenten folgen. Dieses Modell (im Folgenden Komponententypmodell genannt) wird in diesem Kapitel erläutert.



Abbildung[46] 6: Komponententypmodell

Das Komponententypmodell zeigt die im Folgenden beschriebenen Entitäten:

- *Component*

  Zentrales Element im Modell. Jede Komponente besitzt eine Menge an Konfigurationsparametern sowie eine Menge an Partitionierungssparametern. Daneben kann eine Komponente beliebig viele Module besitzen. Jede Komponente ist für einen Ressourcentyp zuständig, jedoch ist die Ausprägung der Ressourcenpartitionierung nicht auf jedem System gleich. Beispielsweise können unter Linux sowohl Dateianzahl als auch der belegte Festplattenplatz beschränkt werden, unter Windows jedoch nur der Festplattenplatz. Mit der Einführung von Modulen kann eine Unterteilung der Partitionierungseigenschaften einer Komponente erfolgen, sodass in der Datenbank nicht nur eine Zuordnung von Knoten und Komponenten erfolgt, sondern auch von Knoten und Modulen. Es ist somit möglich, die gleiche Komponente für verschiedene Ausprägungen zu verwenden. Im gegebenen Beispiel kann folglich die gleiche Komponente für Linux-Quota wie auch für Windows-Quota verwendet werden, obwohl der Funktionsumfang bei beiden Systemen unterschiedlich ist.

- *Module*

  Im Komponentenmodell dienen Module zur Gruppierung von Partitionierungssparametern. Nicht-optionale Module sind automatisch aktiv, d.h. sie müssen bei einer Reservierung verwendet werden, wenn die Komponente, zu der das Modul gehört, auf einem Knoten verwendet wird. Optionale Module können, müssen aber nicht, bei Verwendung der Komponente verwendet werden.

[46] Diagrammtyp: FMC ER Diagramm.

- *Partitioning Parameter*

  Um eine Partitionierung[47] durchzuführen, müssen unterschiedlich viele Angaben bereitgestellt werden. Sofern sie nicht als optional gekennzeichnet sind, erfordert eine Partitionierung das Belegen aller Partitionierungssparameter. Diese können, müssen aber nicht, einem Modul zugeordnet sein. Sind sie einem Modul zugeordnet, müssen sie nur bei aktiviertem Modul belegt werden, da sie andernfalls keinen Einfluss auf die Partitionierung haben. Partitionierungssparameter werden entweder frei gesetzt, oder sie beruhen auf einem dynamischen Parameter, d.h. die Komponente liefert eine Auswahlmöglichkeit an Werten, von denen einer gewählt werden kann (Beispiel: Alle Mountpoints mit aktiviertem Quota). Beruhen Partitionierungssparameter auf einem dynamischen Parameter, so kann es hierfür eine Berechnungsformel geben, welche aus dem Wert des dynamischen Parameters sowie anderen Eigenschaften die Grundlage für den Partitionierungssparameter generiert. Des Weiteren kann ein Partitionierungssparameter Begrenzungen (Restrictions) unterworfen sein.

- Konfigurationsparameter
  - *Parameter*

    Parameter gelten komponentenweit und werden nicht pro Reservierung gesetzt. Sie dienen zur Administration der Komponente auf dem Client und werden in der Regel nur sehr selten geändert, wie zum Beispiel die Angabe der maximalen Netzwerkbandbreite. Parameter können wie Reservierungsparameter Begrenzungen haben.
  - *Dynamic Parameter*

    Dynamische Parameter gelten ebenfalls komponentenweit, doch dienen sie nicht der Administration der Komponente, d.h. sie können nicht gesetzt werden, sondern die Komponente liefert laufzeitabhängige Informationen zurück. Es wird unterschieden zwischen dynamischen Parametern ohne Übergabeparameter[48] und dynamischen Parametern mit Übergabeparameter. Letztere liefern einen Wert zurück, der vom Übergabeparameter abhängig ist[49]. Dynamische Parameter können mehrere konkatenierte und per Semikolon getrennte Werte zurückgeben.

[47] Ob es sich bei der Partitionierung um eine Begrenzung oder eine Reservierung handelt, ist auf dieser Modellebene nicht relevant.

[48] Beispielsweise zum Herausfinden der Mountpoints mit aktiviertem Quota.

[49] Beispielsweise zum Herausfinden der Gesamtkapazität eines Mountpoints um sicherzustellen, dass die Begrenzung diese nicht überschreitet. Hierfür muss der Mountpoint identifiziert werden und sein Name als Übergabeparameter übergeben werden.

- *Restriction*
  Restriktionen dienen zum Begrenzen von Partitionierungsparametern und von nicht-dynamischen Parametern. Sie können entweder statisch sein, d.h. einen festen Wert haben oder auf einem Parameter beruhen. Im zweiten Fall sind sie mit einer Formel, die zur Berechnung des eigentlichen Werts notwendig ist, verknüpft. Beispielsweise wird bei der Reservierung von Netzwerkbandbreite eine obere Grenze berücksichtigt, die sich aus der maximalen Bandbreite sowie den schon erfolgten Reservierungen ergibt.
- *Computation*
  Berechnungen dienen zum einen zur Ermittlung von Begrenzungen und zum andern zur Generierung der Wertemenge für, von Parametern abhängigen, Partitionierungsparametern.

Das Komponententypmodell hat wesentlichen Einfluss auf das noch folgende Datenbankmodell in Kapitel 5.5 .

5.4 Ressourcenidentifikation und Ressourcenzuordnung

Zur Ressourcenpartitionierung in *Idle Time Computing* ist es notwendig, den Ressourcenverbrauch der vom ITC gestarteten Jobs auf den Knoten zu identifizieren und ihn anschließend derart zu partitionieren, dass primär nur der Ressourcenverbrauch des ITC von der Partitionierung betroffen ist[50]. Grundsätzlich gibt es bei allen für das Framework in Frage kommenden Betriebssystemen verschiedene Arten von Ressourcenzuordnungen, von denen zwei[51] eindeutig sind:

- Prozessnummer
  Ressourcen können fast immer einem Prozess zugeordnet werden. Besonders offensichtlich ist diese Zuordnung bei Speicherverbrauch[52], CPU Auslastung sowie Netzwerkverkehr. Weniger eindeutig ist der Verbrauch von Festplattenspeicher. Lediglich schreibende Prozesse können im Moment des Schreibens identifiziert werden, nach Prozessbeendigung kann jedoch keine Zuordnung mehr getroffen werden.
- Benutzerkennung
  Alle Ressourcen, die eindeutig einem Prozess zugeordnet werden, können einem Benutzer zugeordnet werden, da jeder Prozess unter einer Benutzerkennung läuft. Daneben ist die für das Framework wichtige Ressource ‚Festplattenspeicher' einem Benutzer direkt zuordbar, da Dateien ebenfalls immer einem Benutzer zugeordnet sind.

Anhand dieser kurzen Betrachtung empfiehlt es sich, verwendete ITC Ressourcen über eine Benutzerkennung zu identifizieren und zu begrenzen. Bevor jedoch im Rahmen der Frameworkentwicklung eine Entscheidung getroffen werden kann, muss betrachtet werden, ob eine solche Identifikation innerhalb der in Frage kommenden Grid / ITC Systeme überhaupt sinnvoll und möglich erscheint.

Es gibt drei Möglichkeiten, wie innerhalb von Grid-Systemen die Zuordnung zwischen Grid-Anwendern und laufenden Grid-Prozessen erfolgen kann:

1. Verwendung der jeweiligen Anwenderkennung

[50] Natürlich hat die Partitionierung auch Auswirkungen auf anderen Ressourcenverbrauch, jedoch erfolgt dieses nur passiv durch das laufende System, nicht jedoch durch den Mechanismus des Frameworks.

[51] Neben der Benutzerkennung sowie der Prozessnummer gibt es noch weitere Identifikatoren wie z.B. Prozessgruppen oder Benutzergruppen.

[52] Hierbei ist zu beachten, dass *shared memory* jedoch nicht eindeutig zuordbar ist.

Hierbei laufen die Grid-Jobs auf den jeweiligen Knoten unter der Kennung des Anwenders, der den Job in das Grid gestellt hat. Um dieses zu ermöglichen, muss auf allen Knoten des Grids die Anwenderkennung bekannt sein.

2. Verwendung einer dedizierten Grid-Kennung

   Bei diesem Verfahren existiert auf den Knoten des Grids eine dedizierte Benutzerkennung, unter deren ID Prozesse und Jobs des Grid-Computings ausgeführt werden. Innerhalb des Grids sorgt ein interner Mechanismus für das Mapping von Grid-Anwender auf diese Benutzerkennung, um eine völlige Anonymität zu verhindern und um nachzuvollziehen, welcher Anwender für welche Jobs verantwortlich ist.

3. Verwendung einer anonymen Systemkennung

   Ähnlich wie bei der vorherigen Lösung laufen Jobs des Grid Computings nicht unter der Kennung des Anwenders, sondern es wird für alle Prozesse des Grid Computings eine anonyme Systemkennung wie `nobody` verwendet. Auch hierbei existiert ein Mapping-Mechanismus, mit dem es möglich ist, den für den jeweiligen Job zuständigen Anwender herauszufinden. Der wesentliche Unterschied im Vergleich zur Verwendung einer dedizierten Grid-Kennung ist, dass derartige Benutzerkennungen auch von anderen Systemdiensten und Programmen verwendet werden.

Im ersten Fall ist es nicht möglich, von einem Grid-System zu sprechen, gerade weil der Anwender auf allen Systemen bekannt sein muss. Dieses widerspricht der Idee von *Grid Computing*, nämlich dass sich der Anwender keine Gedanken machen muss, wo sein Job läuft, sondern dass das Grid ihm die benötigte Rechenkapazität zur Verfügung stellt. Können Jobs eines Anwenders nur auf Anlagen laufen, auf denen auch der Anwender bekannt ist, so ist keine freie Skalierung mehr möglich, da immer nur ein Teil der insgesamt verfügbaren Knoten für einen Benutzer zur Verfügung steht.

Beide andere Verfahren beinhalten gewisse Risiken. Auch wenn es über Mechanismen des ITC Systems möglich ist, Anwender und deren Jobs zuzuordnen, so ist es von außerhalb nicht möglich, diese Zuordnung zu erkennen. Für Administratoren von Netzen, in denen das ITC läuft, stellt dieses ein großes Problem dar. Sie müssen in der Lage sein, zu erkennen, wem welcher Job bzw. Prozess gehört, um im Falle von zu großem Ressourcenverbrauch oder illegaler Aktivitäten (z.B. Rootkits) den Benutzer ausfindig zu machen. Im Falle der Verwendung einer dedizierten Grid-Kennung ist zumindest erkennbar, dass der jeweilige Prozess zum *Grid Computing* gehört. Werden dagegen Jobs und Prozesse auf eine anonyme Systemkennung, die auch von anderen Prozessen verwendet wird, gemappt, so ist auch dieses nicht erkennbar.

Insgesamt bietet es sich trotz der Sicherheitsbedenken an, für *Grid Computing*, eine dedizierte Benutzerkennung zu verwenden, unter der ausschließlich Jobs und Prozesse des Grids laufen. Noch idealer wäre eine Trennung zwischen dem *Grid Client* / *Grid Daemon* und den eigentlichen Jobs des Grid Computings.

Beim Globus Toolkit erfolgt ein Mapping von Grid-Anwendern zu lokalen Benutzerkonten über die Datei `/etc/grid-security/grid-mapfile`. Identifiziert werden Grid-Anwender über ein Zertifikat, dieses wird für jeden Grid-Anwender auf eine lokale Kennung abgebildet. Diese Kennung kann entweder für jeden Anwender verschieden sein oder es kann eine dedizierte Grid-Kennung bzw. eine anonyme Systemkennung wie `nobody` verwendet werden. Sehr kritisch muss hierbei betrachtet werden, dass für jeden potentiellen Grid-Anwender ein Eintrag vorliegen und somit

jeder Anwender auf jedem Knoten bekannt sein muss. Es stellt sich die Frage, inwieweit dieses den Vorstellungen von *Grid Computing* widerspricht. Bei Condor[53] können Jobs ebenfalls wahlweise unter der Benutzerkennung des Jobabsenders oder über eine allgemeine Kennung laufen, jedoch muss der Jobabsender nicht auf jedem System bekannt sein ([08], Kapitel 3.3.5). Die Identifizierung des Jobabsenders erfolgt bei Condor über die UID_DOMAIN. Als allgemeine Kennung wird per Grundeinstellung `nobody` benutzt, jedoch kann auch eine beliebige andere Kennung verwendet werden ([08], Kapitel 3.7.1).

Unter der Vorraussetzung, dass es einen eindeutigen Benutzer gibt, dessen alleinige Funktion es ist, Jobs / Prozesse des Grid-Computings durchzuführen, bietet die Zuordnung der Ressourcen nach Userkennung mehrere Vorteile gegenüber der Zuordnung nach Prozess ID. Auf diese Weise ist es leicht möglich, die Summe aller verwendeten Ressourcen des jeweiligen Ressourcentyps zu bilden und ebenso ist es möglich, den maximalen Verbrauch bzw. die minimale Zusicherung für einen Benutzer zu treffen. Die Partitionierung ist unabhängig von der Anzahl der parallel laufenden Threads oder Prozesse des ITC, da die Schranken für einen Benutzer gesetzt werden. Nachteilig an dieser Lösung ist, dass nur für alle Prozesse / Jobs zusammen Partitionierungen getroffen werden können, doch da im Allgemeinen davon auszugehen ist, dass es pro Knoten nur einen aktiven ITC Job gibt, sollte dieses kein Problem darstellen.

Anders sieht es aus, wenn keine eindeutige Grid-Kennung existiert. Werden alle Prozesse des Grid Computing auf eine anonyme Systemkennung gemappt, ergibt sich das gleiche Problem, als wenn die Jobs unter der Kennung des Anwenders laufen: Es ist nicht möglich, zu erkennen, ob ein Prozess zum ITC gehört oder ob es ein davon unabhängiger Prozess ist. In diesem Fall kann keine Erkennung aufgrund der User ID erfolgen, sondern die Erkennung muss anhand der Prozess ID erfolgen. Ein nicht unerhebliches Problem ist jedoch die Zuordnung von Prozessen zum *Grid Computing*, da die Prozess ID keine Information beinhaltet, von wem oder von was der Prozess ausgeführt wurde. Folglich muss die Zuordnung auf einer anderen Ebene erfolgen, entweder auf Basis des Namens des ausgeführten Prozesses oder anhand der Elternprozesse. Generell kann gesagt werden, dass eine Lösung über den Namen nicht allgemeingültig sein wird, da nicht vorhersehbar ist, welche Prozesse zu welcher Zeit neue Kindprozesse erzeugen. Demzufolge wird eine Lösung, im folgenden Observer genannt, benötigt, die zu jeder Zeit alle PIDs der ITC Prozesse kennt. Verschiedene Möglichkeiten, wie der Observer von neuen PIDs erfahren soll, sind denkbar:

- Das Grid-System informiert den Observer über neue Prozesse, was bedeuten würde, dass das Grid-System modifiziert werden müsste. Eine Veränderung des Grid-Systems sollte im Rahmen eines flexiblen, unabhängigen und einfachen Frameworks nicht erfolgen, diese Lösung ist deshalb nicht anwendbar.
- Das Betriebssystem informiert unter Angabe der Eltern PID den Observer, wenn ein neuer Prozess gestartet wird. Jedoch unterstützen nicht alle in Frage kommende Betriebssysteme eine derartige Funktionalität[54].

[53] Bei Condor handelt es sich, wie in Kapitel 3.3.4 diskutiert, nicht um ein Grid-System. Aufgrund der gleichen Problematik wie bei *Grid Computing* wird es an dieser Stelle mit dem GT3 verglichen.

[54] Windows XP bietet mit dem *NT Kernel Logger* (einer *Event Tracing Session*) die Möglichkeit, dem System einen Callback zu übergeben, der beim Starten und Beenden von Threads und Prozessen benachrichtig wird. Vergleichbares ist bei Linux nicht vorhanden.

- Ein ständig laufendes Programm fragt regelmäßig die Prozesstabelle ab und erkennt neue Prozesse des ITC' anhand der Eltern PID. Diese neuen Prozesse werden dem Observer mitgeteilt.

Sofern das Betriebssystem nicht den zweiten Ansatz unterstützt, muss der dritte Lösungsvorschlag realisiert werden, um eine stetige Überwachung der laufenden Prozesse zu ermöglichen.

Es erscheint insgesamt wünschenswert, dass im Grid eine dedizierte Kennung zur Ausführung der Jobs existiert und dass die Identifizierung der vom ITC verwendeten Ressourcen mittels User ID erfolgt. Bei dieser Lösung können auch Ressourcen partitioniert werden, die nicht über eine Prozess ID identifizierbar sind. Andererseits muss berücksichtigt werden, dass nicht alle Ressourcen pauschal über eine User ID identifiziert werden können, sondern dass manche Mechanismen nur über die Prozess ID wirken[55]. Für diese Mechanismen wird auch unter Verwendung einer dedizierten Kennung ein Programm zur Prozesserkennung bzw. zur Prozessgruppierung benötigt. Die Identifizierung der betroffenen Prozesse ist jedoch gerade wegen der dedizierten Benutzerkennung sehr leicht realisierbar.

Bei der weiteren Realisierung des Frameworks werden die in diesem Kapitel angestellten Überlegungen berücksichtigt und es wird von einer dedizierten Benutzerkennung fürs *Grid Computing* ausgegangen.

5.5 Anbindung ans Grid

Derzeit gibt es nicht *das eine* Grid, sondern es existieren mehrere Frameworks, unter denen keine völlige Kompatibilität herrscht. Es ist deshalb nicht möglich, eine generelle Anbindung an beliebige Grid-Systeme zu realisieren. Schwerpunkt bei der bisherigen Betrachtung war das Globus Toolkit, sodass sich dieses Kapitel auch auf die Einbindung bezüglich GT3 konzentriert.

Ressourcenmanagement in GT3 wurde in Kapitel 3.3.3 aufgezeigt. Wie in Abbildung 4 (Seite 19) dargestellt, wird GT3 im Allgemeinen als übergeordnetes Grid-System unter Verwendung von mehreren, jeweils auf eine lokale Domäne beschränkten, Cluster-Systemen eingesetzt[56]. Je nachdem ob GT3 in Verbindung mit Cluster-Systemen eingesetzt wird oder ob nicht, ergeben sich andere Anbindungsanforderungen.

Bei der Verwendung eines Cluster-Systems muss bedacht werden, dass der Jobplatzierungsvorgang nicht vom GT3 durchgeführt wird, sondern vom Scheduler des verwendeten Cluster-Systems. Bei vielen Cluster-Schedulern wird ein Ressourcenabgleich zwischen Bedarf und vorhandener Kapazität durchgeführt, jedoch ohne Ressourcen zu reservieren[57]. Um eine Integration des in dieser Arbeit entwickelten Frameworks zu erreichen, muss der jeweilige Cluster-Scheduler über RSL (oder eine vergleichbar standardisierte Schnittstelle) auf die Broker API zwecks Ressourcenreservierung zugreifen und die Ergebnisse der Abfragen bzw. der Reservierungsversuche in die Entscheidung bezüglich der Jobplatzierung einfließen lassen. Um dieses zu erreichen, bedarf es einer Anpassung des Schedulers des verwendeten Cluster-Systems.

[55] In Kapitel 4 erfolgte die Betrachtung der Ressourcenpartitionierungsmöglichkeiten unter Windows und Linux. Während Ressourcen wie Festplattenkapazität oder Netzwerkbandbreite über die User ID identifiziert werden können, sind Speicherverbrauch und CPU Auslastung meistens nur direkt für einen Prozess einstellbar.

[56] Domäne bezeichnet hier den Einsatzbereich eines Cluster-Systems. Dieser ist beschränkt auf eine lokale Administrationsdomäne wie z.B. ein Rechenzentrum an einer Universität.

[57] Für den allgemeinen Fall erfolgte die Beschreibung in Kapitel 0, für Condor in 3.3.4 .

Wird kein Cluster-System verwendet, erfolgt das Scheduling der Jobs innerhalb des Globus Systems, jedoch ist die Informationssituation diesbezüglich etwas unklar. In der Architekturbeschreibung von GT3 [21] wird kein interner Scheduler erwähnt, sondern es wird lediglich folgende Aussage bezüglich der Jobplatzierung getroffen:

> *MJS submits the job into a back-end scheduling system*

Um welches *back-end scheduling system* es sich handelt, wenn GT3 ohne unterliegende Cluster-Systeme verwendet wird, ist aus der vorhandenen GRAM Dokumentation auf den Globus-Webseiten nicht ersichtlich. Fest steht, dass RSL derzeit nur zum Anbinden externer Scheduler verwendet wird, so dass ein Ansprechen des Frameworks ohne Veränderungen innerhalb von GT3 nur dann möglich ist, wenn das Framework neben der Ressourcenpartitionierung auch die anderen Aufgaben eines Schedulers übernimmt und das *GRAM Job Manager Scheduler interface* implementiert. Es ist jedoch nicht das Ziel des hier vorgestellten Frameworks, einen Job-Scheduler zu implementieren. Folglich muss festgestellt werden, dass es ohne Veränderungen an einem bestehenden und mit GT3 verwendbaren Job-Scheduler nicht möglich ist, Ressourcengarantien mit diesem Framework durch GT3 zu erreichen.

Ungeachtet der Problematik, welch Scheduler oder andere Komponente mit dem hier vorgestellten Framework kommuniziert, muss die Broker API eine standardisierte Schnittstelle anbieten, um einen Zugriff von außerhalb zu ermöglichen. Hierbei bietet sich die RSL (Kapitel 3.3.3) oder das noch in Entwicklung befindliche GRAAP (Kapitel 3.3.7) an.

5.6 Datenbankentwurf

Basierend auf dem Komponentenmodell aus Kapitel 5.3 erfolgt in diesem Kapitel der Datenbankentwurf. Ihm zu Grunde liegt die Entscheidung (beruhend auf der Diskussion in Kapitel 5.4), Partitionierungen auf Benutzerebene durchzuführen und nicht auf Prozessebene.

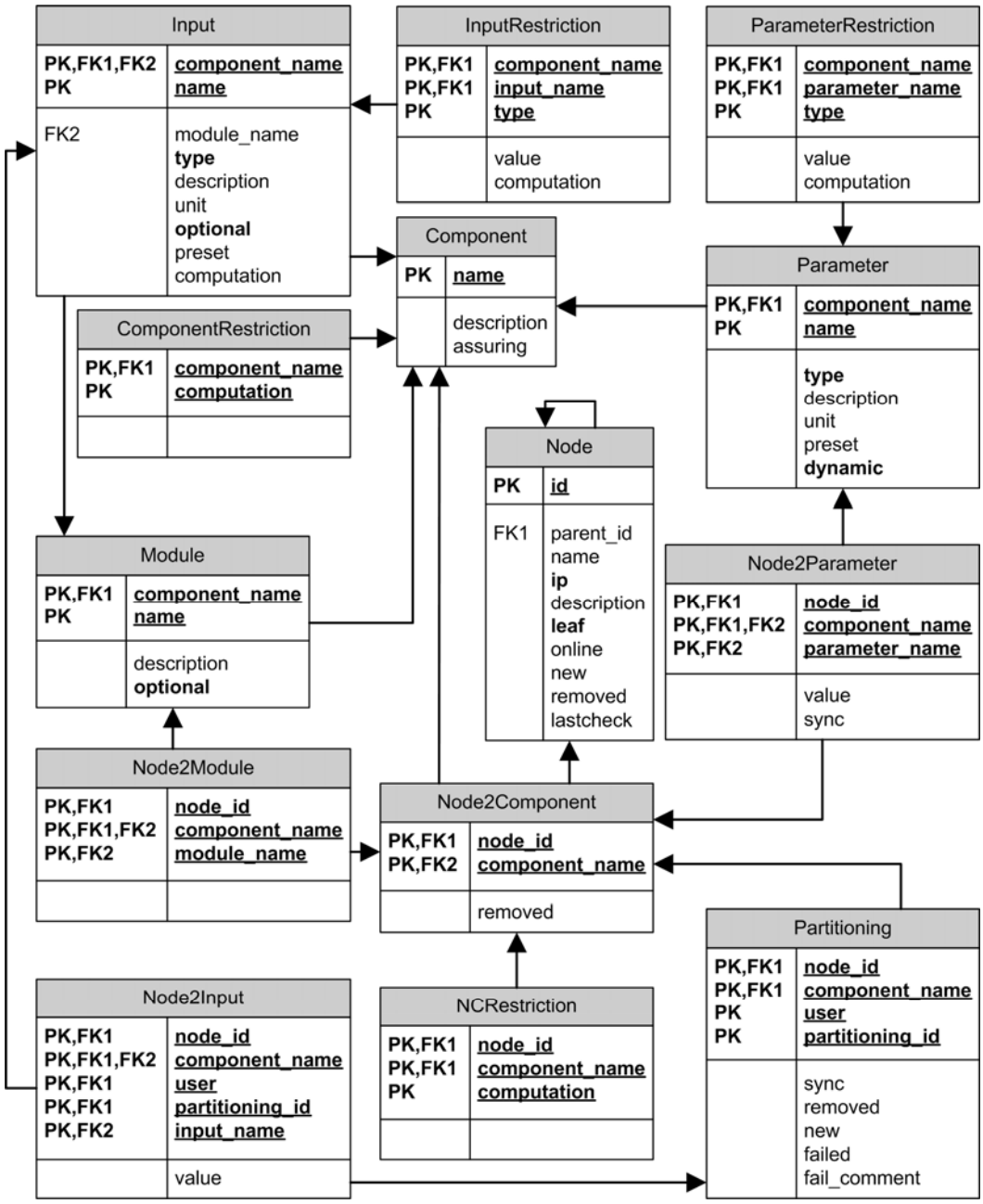


Abbildung 7: Datenbankentwurf

Der Datenbankentwurf zeigt alle für das Framework vorgesehenen Tabellen. Wie bereits erwähnt, ist das Framework ressourcenunabhängig und komponentenbasiert. In der Datenbank sind Komponentendefinitionen, Knoten (Clients), Zuordnung zwischen Knoten und Komponenten sowie Reservierungen gespeichert. Im Folgenden wird auf die einzelnen Tabellen eingegangen. Detaillierte Informationen zu den einzelnen Spalten der Tabellen sind Anhang A zu entnehmen.

Datenbanktabelle	Beschreibung
Component	Speichert Angaben zu Komponenten. Die Spalte `assuring` gibt an, ob mit der Komponente Mindestgarantien gesetzt werden können. Nur wenn dies möglich ist, macht ein Zugriff seitens des Grid auf die Komponente Sinn, andernfalls dient sie ausschließlich zur administrativen Beschränkung.
ComponentRestriction	Diese Tabelle enthält Begrenzungsregeln für Komponenten wie z.B. Anzahl Reservierungen pro Benutzer oder Anzahl Reservierungen insgesamt. Es handelt sich hierbei nicht um administrative Beschränkungen, sondern um technische Beschränkungen.
Module	Jede Komponente kann, muss aber nicht, Module enthalten. Sie werden in dieser Tabelle gespeichert. Sind Module vorhanden, können Partitionierungsparameter einem Modul oder aber auch weiterhin direkt der Komponente zugeordnet sein.
Input	In der Tabelle Input werden die im ER-Diagramm genannten Partitionierungsparameter gespeichert. Sie sind einer Komponente zugeordnet und können einem Modul zugeordnet sein. Basieren sie auf einem Parameter, so steht die Formel in der Spalte `computation`.
InputRestriction	Diese Tabelle enthält Begrenzungsregeln für die Eingabefelder.
Parameter	Enthält alle (sowohl dynamische als auch nicht dynamische) Parameter der Komponente.
ParameterRestriction	Diese Tabelle enthält Begrenzungsregeln für die Parameter.
Node	Das Framework unterstützt eine hierarchische, baumartige Knotenstruktur. In dieser Tabelle sind alle Knoten gespeichert, sowohl Blattknoten als auch Knoten zur Gruppierung von Knoten.
Node2Component	Zuordnung zwischen Knoten und Komponenten.
Node2Module	Zuordnung zwischen Knoten und Modulen.
Node2Parameter	Alle für eine Komponente definierten Parameter müssen pro Knoten gesetzt werden und sind in dieser Tabelle gespeichert.
Partitioning	In dieser Tabelle werden alle Partitionierungen gespeichert. Wie bereits erwähnt, wurde sich für eine benutzerbasierte Identifikation der Ressourcen entschieden, weshalb es die Spalte `user` gibt, in welcher der Benutzername gespeichert wird. Pro (Benutzer, Knoten, Ressource)-Tripel können prinzipiell mehrere Partitionierungen durchgeführt werden, unterschieden werden sie anhand der `partitioning_id`.
Node2Input	Pro Partitionierung müssen alle entsprechenden Partitionierungs-parameter bekannt sein, sie werden in dieser Tabelle gespeichert.
NCRestriction	In dieser Tabelle (NC steht für Node2Component) werden administrative Beschränkungen zu einem Knoten-Komponenten-Paar gespeichert. Diese Tabelle ist vergleichbar mit ComponentRestriction, nur dass die Begrenzungen nicht generell für die Komponente gelten, sondern für ein Knoten-Komponenten-Paar und dass die Begrenzungen nicht technischer Ursache sind, sondern administrativer um Reservierungen seitens des Grids zu beschränken.

Tabelle 6: Beschreibung der Datenbanktabellen

Foreign Keys sorgen für best mögliche Integrität der Daten, nicht abgedeckt werden semantische Begrenzungen. Die Interpretation der in den Tabellen gespeicherten Angaben sowie der `computation` Felder obliegt bei der Broker API (Kapitel 0) sowie beim Database Crawler (Kapitel 0).

5.7 XML RPC Schnittstelle

Für die Kommunikation zwischen Broker Server und den Clients wurde XML RPC[58] ausgewählt, da es ein sehr einfaches und auf XML und HTTP basierendes Protokoll ist. CORBA und DCOM kommen aufgrund ihrer Komplexität nicht in Frage, SOAP wäre als Alternative denkbar, doch ist es ebenfalls komplexer als notwendig. XML RPC bietet genau die benötigte Funktionalität und ist für alle, zur Clientprogrammierung in Frage kommenden, Programmiersprachen verfügbar.

<<interface>>

BrokerClient

+addReservation(Component:string,User:string,Slot:string,ReservationParameter:string):string
+changeReservation(Component:string,User:string,Slot:string, ReservationParameter:string):string
+removeReservation(Component:string,User:string,Slot:string):string
+removeReservations(Component:string,User:string):string
+removeAllReservations(Component:string):string
+setParameter(Component:string,ParameterName:string,Value:string):string
+getparameter(Component:string,ParameterName:string):string
+getDynamicParameter(Component:string,ParameterName:string):string
+getDynamicParameter(Component:string,ParameterName:string,Parameter:string):string
+ping():string

Tabelle 7: Interface BrokerClient

Funktion	Beschreibung
addReservation	Führt eine Reservierung für einen Benutzer durch.
changeReservation	Ändert eine Reservierung eines Benutzers.
removeReservation	Löscht eine Reservierung eines Benutzers.
removeReservations	Löscht alle Reservierungen eines Benutzers.
removeAllReservations	Löscht alle Reservierungen aller Benutzer.
setParameter	Setzt einen Parameter.
getParameter	Liefert einen Parameter zurück.
getDynamicParamter	Liefert einen dynamischen Parameter zurück.
Ping	Liefert den Status des Clients zurück.

Tabelle 8: Beschreibung der Schnittstellenmethoden

Anmerkungen zum BrokerClient Interface:

- Die im Datenbankentwurf eingeführte `reservation_id` wird in den entsprechenden Funktionen mit `slot` bezeichnet.
- Aufgrund der Unabhängigkeit des Frameworks von den einzelnen Ressourcen und somit auch von der Anzahl der Reservierungsparamter werden beim

[58] XML RPC Homepage: http://www.xmlrpc.com/

Setzen und Ändern einer Partitionierung alle Partitionierungsparamter gebündelt übertragen. Hierzu werden alle Parameter nach folgendem Muster konkateniert:

```
ReservationParameter=”Key1=Value1;Key2=Value2;Key3=Value3”
```

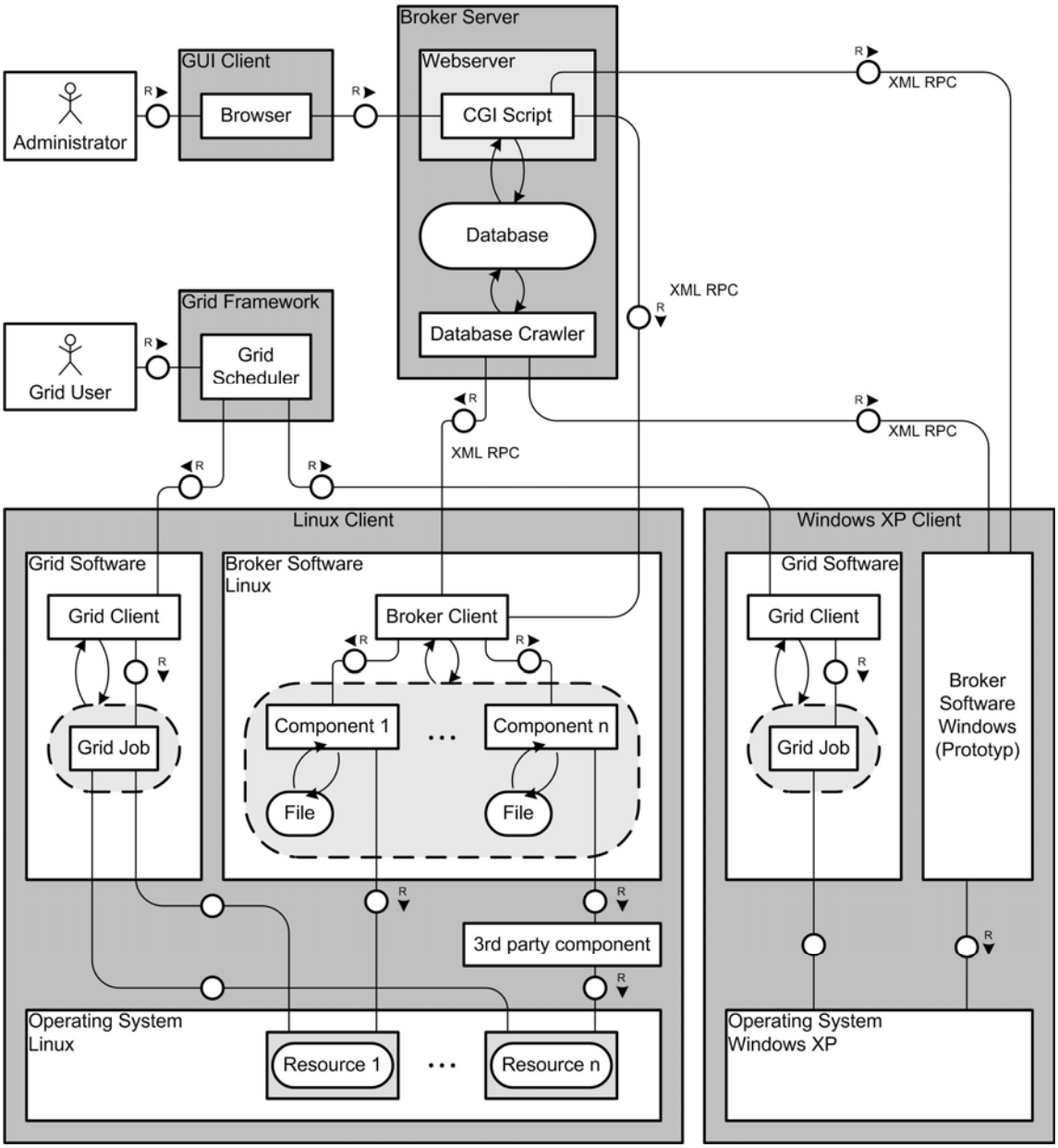
Daraus folgt, dass einzelne Reservierungsparameter weder ‚=’ noch ‚;’ enthalten dürfen.

- Sowohl für Parameter als auch für Partitionierungsparameter erfolgt keine Typbeachtung. Sie werden grundsätzlich als String behandelt, ein Typecasting muss auf Server- und Clientseite erfolgen.
- Bei allen Funktionen mit Ausnahme von `ping()` ist der erste Parameter die Komponente, für die der aktuelle Funktionsaufruf gilt.
- `ping()` ist die einzige Funktion, die vom BrokerClient selbst ausgeführt wird. Sie dient im Wesentlichen zur Überprüfung, ob der Client online ist und der BrokerClient läuft (Rückgabewert: ‚0’).
- Die Funktion `getDynamicParameter` ist polymorph, da sie sowohl für dynamische Parameter mit und ohne Übergabeparameter existiert.
- Der Rückgabewert ist immer vom Typ String und dient (außer bei `getParameter` und `getDynamicParamater`) als Statusinformation. Eine ‚0’ steht für erfolgreiche Ausführung, eine ‚1’ gefolgt von einer, per Semikolon von ihr getrennten, Fehlermeldung steht für eine nicht erfolgreiche Ausführung.

6 Implementierung

Schwerpunkt der Implementierung ist die Überprüfung, ob der in Kapitel 5 aufgestellte Frameworkentwurf tragfähig ist und ob die in Kapitel 4 aufgezeigten Partitionierungsmöglichkeiten mit einem derartigen Framework unter realen Bedingungen praktikabel sind. Kann beides gezeigt werden, so ist es möglich, die Durchführbarkeit der in Kapitel 3.2 aufgestellten Anwendungsszenarien mit dem Framework zu überprüfen. Die Implementierung konzentriert sich auf einen Einsatz für die administrative Beschränkung von Grid-Ressourcen, weniger auf eine direkte Anbindung an ein Grid, da (wie bereits in Kapitel 5.5 aufgezeigt) eine Anbindung nicht ohne Veränderungen an dem vom Grid verwendetem Scheduler möglich ist.

Die folgende Grafik veranschaulicht die Architektur der Implementierung, Änderungen vom Entwurf werden beim Vergleich mit Abbildung 5 (Seite 38) erkennbar. Wie in der Architekturabbildung der Implementierung ist auch hier die Darstellung der Komponenten des Grid-Systems nicht vollständig. Die dargestellten Zugriffe der Grid-Jobs auf das Betriebssystem bzw. auf die vom Betriebssystem verwalteten Ressourcen sind ebenfalls nicht vollständig: Beim Linux-Client sind neben den dargestellten Zugriffen noch andere, auf nicht vom Framework kontrollierte, Ressourcen existent. Die Darstellung des Windows-Client ist insgesamt nur als schematische Darstellung zu verstehen.



Abbildung[59] 8: Architektur der Implementierung

Die Unterschiede im Vergleich zur Entwurfsarchitektur werden in den folgenden Kapiteln aufgezeigt. Generell wird nur auf wesentliche Implementierungsmerkmale eingegangen, Details müssen dem Quellcode entnommen werden.

6.1 Broker Server

Wesentlichster Unterschied zur Architektur des Entwurfs ist das Weglassen der Broker API. Diese wird, da keine RSL Schnittstelle in der Implementierung vorgesehen ist, nicht benötigt. Ihre Funktionalität befindet sich in den jeweiligen CGI Skripten.

[59] Diagrammtyp: FMC Blockdiagramm.

6.1.25 Datenbank

Das Datenbankdesign von Abbildung 7 (Seite 49) wurde mehrheitlich übernommen, es ergaben sich folgende Änderungen und Abweichungen:

- Tabelle Component:
  Wegfall der Spalte `asssuring`, da aufgrund der Fokussierung auf administrative Beschränkungen die damit verbundene Funktionalität nicht benötigt wird.
- Tabelle ParameterRestriction:
  Vollständig weggelassen, da die dahinter stehende Funktionalität nicht implementiert wurde.
- Tabelle ComponentRestriction
  Vollständig weggelassen, da die dahinter stehende Funktionalität nicht implementiert wurde.
- Tabelle NCRestriction
  Vollständig weggelassen, da die dahinter stehende Funktionalität nicht implementiert wurde.
- Tabelle InputRestriction
  Die Tabelle wurde unter dem Namen Restriction angelegt und auch mit Daten gefüllt, die dahinter stehende Funktionalität wurde jedoch nicht implementiert
- Tabelle Partitioning
  Die Tabelle wurde unter dem Namen Reservation angelegt und die Spalte `partitioning_id` wurde `reservation_id` genannt. Aus diesem Grund wird in der folgenden Betrachtung der Implementierung der Begriff Reservierung anstatt Partitionierung verwendet.
- Tabelle Node2Input
  Die Spalte `partitioning_id` wurde in `reservation_id` umbenannt.

Als DBMS wurde MySQL[60] eingesetzt. Zur Realisierung der Foreign Keys und der `AUTO_INCREMENT` Funktion bei `Node.id` wurde der Tabellentyp InnoDB verwendet.

6.1.26 Komponenten

Die folgenden Komponenten wurden entwickelt und ihre Definitionen nach dem gezeigten Komponententypmodell (Abbildung 6) in der Datenbank realisiert:

- Network
  Sowohl zum Begrenzen als auch zum Setzen von Mindestgarantien von Netzwerkbandbreite.
- DSCP
  Zum Setzen eines *Differentiated Services Code Point* im IP Header von ausgehenden Netzwerkpaketen.
- Quota
  Zum Begrenzen von verwendeter Festplattenkapazität.
- FSF
  Zum Einrichten von garantierter Festplattenkapazität.

Eine detaillierte Beschreibung der Komponenten erfolgt in Kapitel 6.2 .

[60] MySQL Homepage: http://www.mysql.de/

6.1.27 Webserverskripte

Die webserverbasierte Administrationsschnittstelle wurde mittels CGI Skripten in der Programmiersprache Perl realisiert. Wie bereits erwähnt, wurde die Implementierung der Broker API verzichtet, stattdessen wurde die benötigte Funktionalität innerhalb der CGI Skripte implementiert.

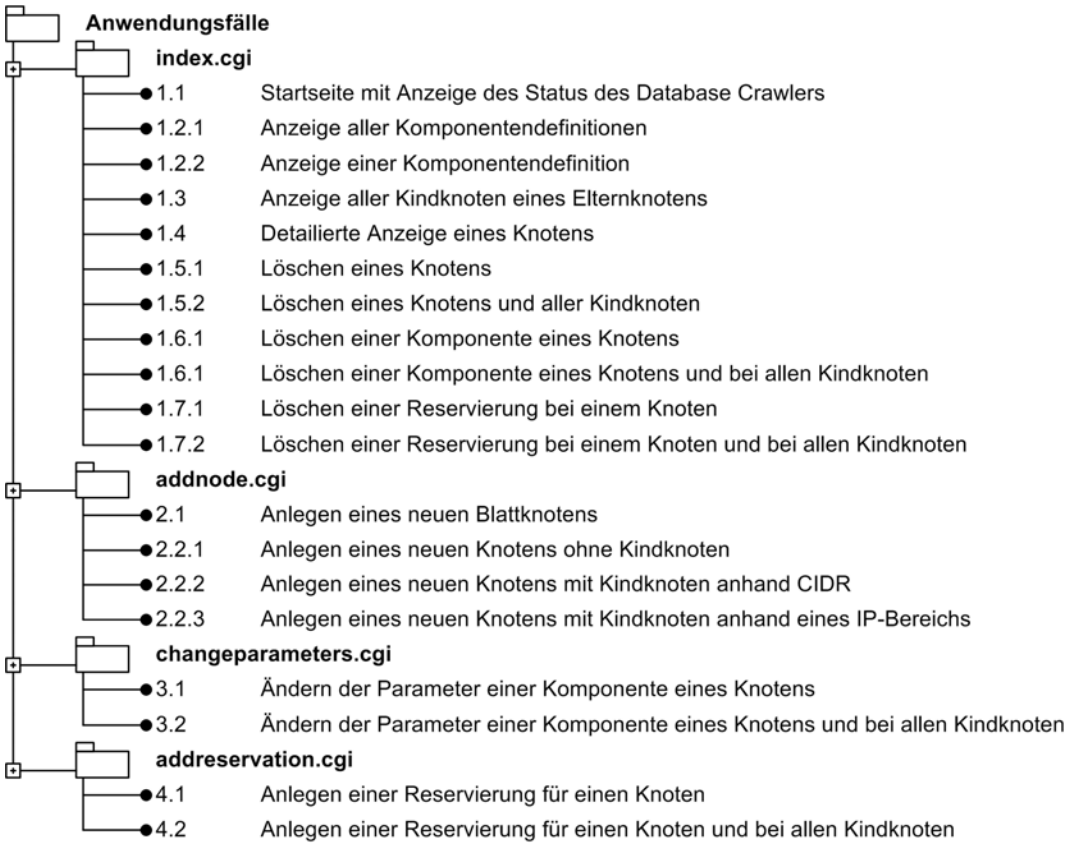


Abbildung 9: CGI Skripte und ihre Anwendungsfälle

Zur Veranschaulichung der Funktionalität sei auf die Bildschirmfotos in Anhang B verwiesen.

Als XML RPC API wurde sowohl beim Server (dies umfasst einige CGI Skripte sowie den Database Crawler) als auch beim Linux Client die Implementierung von Ken MacLeod verwendet[61].

6.1.28 Database Crawler

Der Database Crawler durchläuft in einer Endlosschleife folgende Zustände:

1. Zurückkommende Kindprozesse abfangen.
2. Beenden von zu lange laufenden Kindprozessen[62].
3. Knoten überprüfen ob sie online sind.
4. Nicht synchrone Parameter synchronisieren.

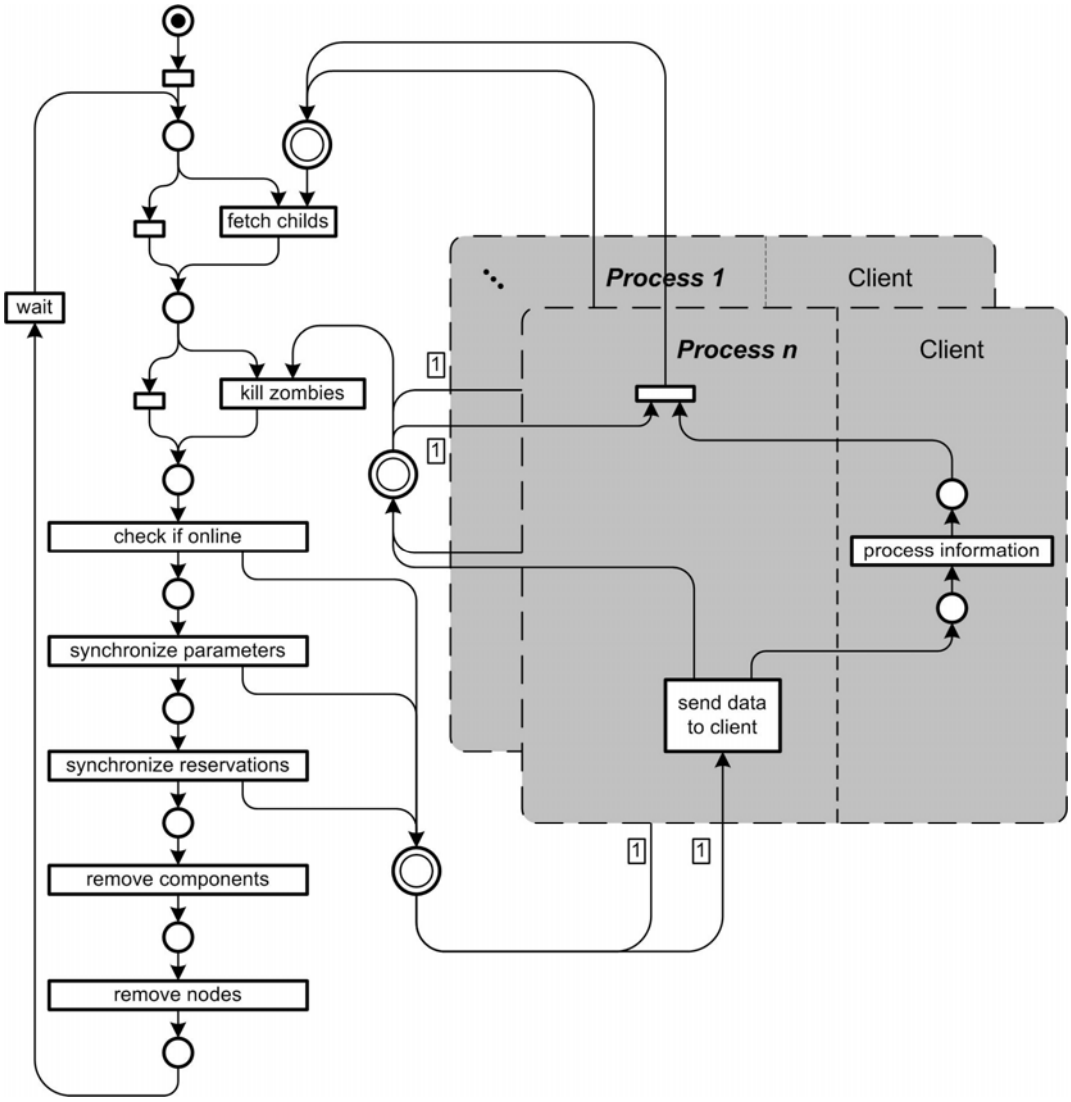
[61] Perl XML RPC Implementierung: http://bitsko.slc.ut.us/~ken/xml-rpc/

[62] Ein automatischer Timeout beim Überprüfen von nicht angeschlossenen (offline) Knoten erfolgt erst nach verhältnismäßig langer Zeit. Prozesse, die auf einen solchen Timeout warten, werden vom Crawler bereits vorher beendet.

5. Nicht synchrone Reservierungen synchronisieren.
6. Als gelöscht markierte Komponenten überprüfen und löschen.
7. Als gelöscht markierte Knoten überprüfen und löschen.
8. Ein paar Sekunden Inaktivität zur Vermeidung von Dauerlast.

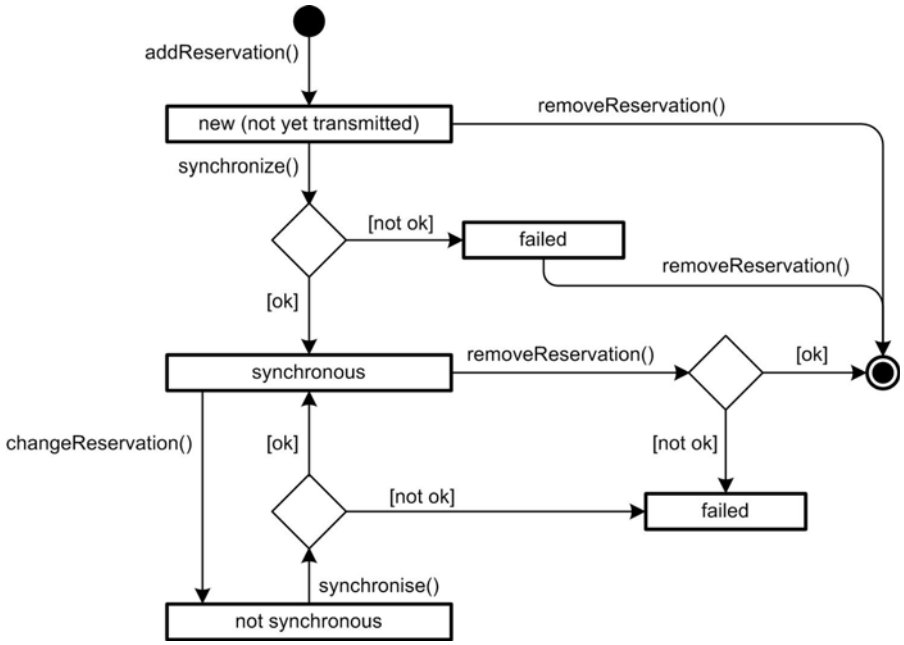
Bei den genannten Aktionen 3, 4 und 5 werden Kindprozesse generiert, die die eigentliche Kommunikation mit dem jeweiligen Client übernehmen. Es werden dabei nicht beliebig viele Prozesse erzeugt, sondern es sind zwei Maximalwerte (eines für Überprüfung des Onlinestatus und eines für Synchronisierungen) im Programm angegeben, die die Maximalanzahl der Kinder definieren. Ist diese Anzahl erreicht, erfolgt kein weiteres Forken, sondern es wird solange gewartet, bis ein Kindprozess zurückgekommen ist oder beendet wurde. Beim Löschen von Komponenten oder Knoten werden vorher alle betroffenen Reservierungen entfernt und Kinder nach dem gleichen Schema behandelt. Hierbei werden diese Schritte zuerst nur in den entsprechenden Feldern in der Datenbank markiert, das eigentliche Löschen von Reservierungen geschieht dann ab dem nächsten Zyklus des Crawlers. Sind Kinder oder ist der Knoten selber offline und können Reservierungen deshalb nicht auf dem Knoten gelöscht werden, so wird die entsprechende Komponente bzw. der entsprechende Knoten solange nicht endgültig gelöscht, bis das Löschen der Reservierungen geschehen ist. Die Funktionalität, dass beim Löschen von Knoten oder Komponenten auch alle Reservierungen gelöscht werden und alle Kinder dementsprechend behandelt werden, sollte von Anwenderseite aus erfolgen[63]. Der Database Crawler hat sie zusätzlich implementiert, um ungewollte Zustände zu vermeiden.

[63] Die Webskripte implementieren dieses Verhalten.



Abbildung[64] 10: Aktivitäten des Database Crawlers

[64] Diagrammtyp: Petrinetz (FMC *dynamic structure diagram*).



Abbildung[65] 11: Reservierungszustände

Das Zustandsdiagramm zeigt die möglichen Zustände einer Reservierung. Der Endzustand (die Löschung der Reservierung aus der Datenbank) wird in zwei Fällen vom Framework nicht erreicht. Schlägt eine Reservierung oder das Löschen einer Reservierung auf einem Client fehl, so erfolgen keine weiteren Versuche dieses durchzuführen. In beiden Fällen muss manuell eingeschritten werden, da keine automatische Entscheidung erfolgen kann.

Die Zustände der einzelnen Objekte werden innerhalb der Datenbank durch folgende booleschen Variablen realisiert:

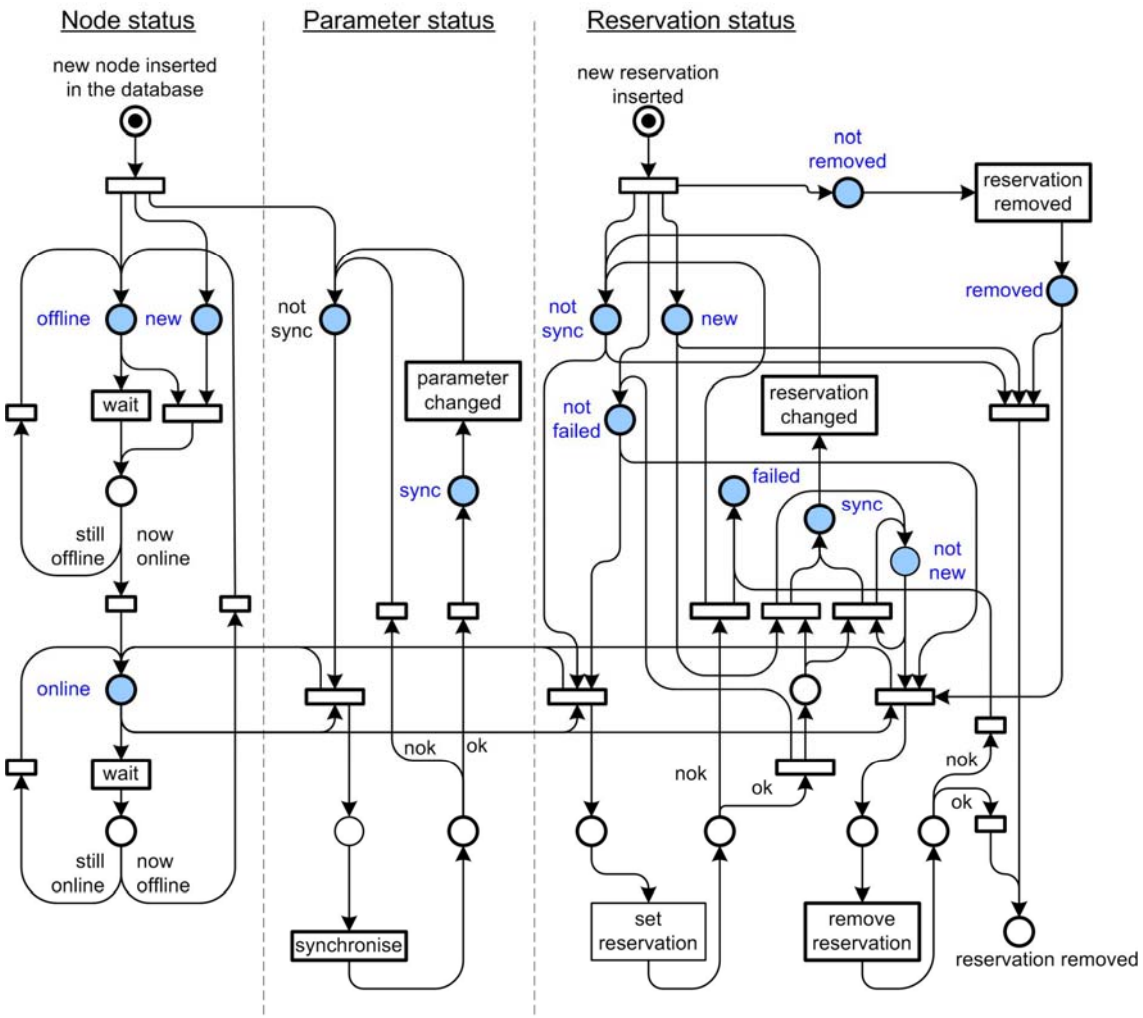
Datenbanktabellenname	Variable
Node	new online removed
Reservation	new removed sync failed
Node2Component	removed
Node2Parameter	sync

Tabelle 9: Boolesche Zustandsvariablen

Details zu den Bedeutungen und Funktionen der Variablen siehe Anhang A.

[65] Diagrammtyp: UML Zustandsdiagramm. Notation der verschiedenen UML Diagrammtypen: [30].

Das folgende Petrinetz zeigt das Systemverhalten, realisiert durch die Aktionen des Benutzers mit Hilfe der Weboberfläche sowie durch den Database Crawler. Die in der obigen Tabelle aufgelisteten booleschen Variablen sind im Petrinetz jeweils doppelt enthalten (blau eingefärbt), sowohl in normaler als auch in negierter Form. Aus Komplexitätsgründen sind im Diagramm nicht die Möglichkeiten des Löschens eines Knotens oder einer Komponente berücksichtigt und somit sind auch nicht die Variablen `Node.removed` und `Node2Component.removed` im Diagramm enthalten. Die Variable `Node.new` ist nicht in negierter Form vorhanden.



Abbildung[66] 12: Zustände von Knoten und Reservierungen

Für Parameter und Reservierungen ist jeweils nur ein Fall dargestellt, dieses ist auf beliebig viele Parameter und Reservierungen zu übertragen. Das vorhergehende Zustandsdiagramm lässt sich auf der rechten Seite im Petrinetz wieder finden.

6.2 Komponenten

In Kapitel 4.4 wurde abschließend beurteilt, welche Partitionierungen unter Linux und Windows durchgeführt werden können. Es sind vier Komponenten entwickelt worden, die Beschreibung erfolgt an dieser Stelle betriebssystemunabhängig, in den

[66] Diagrammtyp: Petrinetz (FMC *dynamic structure diagram*).

Kapiteln zu den Clients wird auf die jeweilige Implementierung eingegangen. Insgesamt jedoch konzentriert sich die Implementierung auf die Entwicklung des Linux Clients, weshalb keine Komponenten für nur unter Windows existierende Partitionierungsmöglichkeiten entwickelt wurden. Hiervon betroffen sind (siehe Tabelle 5, Seite 36) die Möglichkeiten zur Begrenzung der CPU Zeit sowie zur Speicherlimitierung, die beide unter Windows vorhanden sind, in der Implementierung jedoch aufgrund der Fokussierung auf Linux nicht realisiert werden.

6.2.29 Network

Diese Komponente dient der Partitionierung der Netzwerkbandbreitennutzung. Die übergebene Bandbreite ist dabei sowohl Mindestgarantie, kann aber auch als Beschränkung fungieren. Folgende Parameter und Reservierungsparameter besitzt die Komponente:

Name	Beschreibung
Device	Netzwerkinterface, für das die Reservierungen durchgeführt werden.
DeviceSpeed	Geschwindigkeit des Netzwerkinterfaces in kbit/s.
TC	Aufrufpfad vom Programm zur Bandbreitenpartitionierung, unter Linux iproute2 traffic control. Sofern es in einem über $PATH erreichbaren Verzeichnis liegt, reicht der Programmname (unter Linux „tc“) aus.

Tabelle 10: Parameter der Komponente Network

Derzeit werden diese Parameter nicht automatisch ermittelt, dieses wäre jedoch bei einigen denkbar und sollte bei einer Weiterentwicklung berücksichtigt werden. Für *Idle Time Computing* in Frage kommende Arbeitsplatzrechner verfügen im Allgemeinen nur über einen Netzwerkanschluss mit dem sie an das LAN angeschlossen sind. Deshalb unterstützt die Komponente gegenwärtig nur die Verwendung eines Netzwerkinterfaces.

Name	Beschreibung
SPEED	Netzwerkbandbreite (in kbit/s) der Reservierung.
BOUNDED	Diese boolesche Variable gibt an, ob die Bandbreite auf den über SPEED definierten Wert begrenzt bleiben soll oder ob sie im Falle freier Kapazitäten diese ausnutzen soll.

Tabelle 11: Reservierungsparameter der Komponente Network

Die Funktionalität der Komponente lässt sich nicht in kleinere Einheiten aufteilen, demzufolge sind der Komponente keine Module zugeordnet. Pro Benutzer kann nur eine Reservierung gesetzt werden.

6.2.30 DSCP

Mit der Komponente können keine direkten Partitionierungen durchgeführt werden, sondern die Komponente DSCP dient dazu, IP-Paketen bestimmte *Differentiated Services Code Points* (DSCP) in den *IP Header* zu schreiben, um so eine dedizierte Behandlung durch ein *Differentiated Services* Framework zu ermöglichen. Hierdurch ist es möglich, netzwerkweite Bandbreitenkontrolle zu erlangen. Die Komponente besitzt keine Module, pro Benutzer kann nur eine Reservierung durchgeführt werden und sie besitzt auch keine Parameter. Folgende Reservierungsparameter gibt es:

Name	*Beschreibung*
DSCP	*Differentiated Services Code Point, Wert[67] zwischen 0 und 63.*

Tabelle 12: Reservierungsparameter der Komponente DSCP

6.2.31 Quota

Mittels dieser Komponente kann der Verbrauch an Festplattenressourcen kontrolliert und begrenzt werden. Während in Windows nur die Begrenzung des belegten Speicherplatzes möglich ist, ist unter Linux auch die Dateianzahl pro Benutzer begrenzbar. Um trotzdem eine Komponentendefinition für beide Systeme zu verwenden, besitzt die Komponente zwei Module. Pro Benutzer kann eine Partitionierung pro Mountpoint durchgeführt werden.

Die folgenden Tabellen zeigen die vorhanden Parameter und Reservierungsparameter:

Name	**Parameter**
Mountpoints	Dieser dynamische Parameter gibt eine per Semikolon separierte Liste aller, mit aktiviertem Quotamechanismus versehenen, Mountpoints zurück.
MountpointValues	Dieser dynamische Parameter bekommt einen Mountpoint übergeben und liefert, per Semikolon getrennt, Gesamtkapazität und freie Kapazität (beides in Kilobyte) zurück.

Tabelle 13: Parameter der Komponente Quota

Name	**Modul**	**Beschreibung**
MP		Mountpoint für den die Begrenzung erfolgen soll.
BS	Blocksize	Maximale Festplattenbelegung, hartes (hard) Limit, in Megabytes.
BH	Blocksize	Maximale Festplattenbelegung, weiches (soft) Limit, in Megabytes.
FS	Filesize	Maximale Anzahl von Dateien, hartes (hard) Limit.
FH	Filesize	Maximale Anzahl von Dateien, weiches (soft) Limit.

Tabelle 14: Reservierungsparameter der Komponente Quota

6.2.32 FSF

Während die bisher beschriebenen Komponenten relativ betriebssystemneutral entwickelt wurden, ist der Komponente FSF (*Fixed Swap File*) die Nähe zu Linux deutlich anzumerken. Mit ihr ist es möglich, Festplattenplatz mit dem in Kapitel 0 beschriebenen Verfahren zu reservieren. Pro Benutzer sind dabei theoretisch beliebig viele Reservierungen denkbar. Folgende Parameter und Reservierungsparameter besitzt die Komponente:

[67] Für den DSCP Wert stehen im IP Header 6 Bits des TOS Feldes zur Verfügung.

Name	*Beschreibung*
MountpointBaseDir	*Verzeichnis in dem die Swapdateien[68] gemountet werden. Dieses Verzeichnis liegt selber direkt im Home-Verzeichnis des Benutzers. Dieses bietet den Vorteil, dass es im Home-Verzeichnis, welches häufig per NFS gemountet wird, einen lokalen Bereich gibt, sodass dort Grid-Jobs temporäre Daten erzeugen können ohne auf ein Netzlaufwerk zuzugreifen.*
SwapFileDirectory	*Verzeichnis in dem die Swapdateien erzeugt werden. Die Dateien werden unter Kennung des Benutzers angelegt. Ist parallel die Komponente Quota aktiviert, muss dieses bedacht werden.*
MountpointValues	*Dieser dynamische Parameter bekommt per Semikolon getrennt einen Benutzer sowie eine* `reservation_id` *übergeben und liefert, ebenfalls per Semikolon getrennt, Gesamtkapazität und freie Kapazität (beides in Kilobyte) des dazugehörenden Mountpoints (im MountpointBaseDir) zurück.*
SwapFileDirectoryValues	*Dieser dynamische Parameter liefert, per Semikolon getrennt, Gesamtkapazität und freie Kapazität (beides in Kilobyte) des SwapFileDirectorys zurück.*

Tabelle 15: Parameter der Komponente FSF

Name	**Beschreibung**
FS	Größe der anzulegenden Datei / des anzulegenden Mountpoints in Kilobytes.

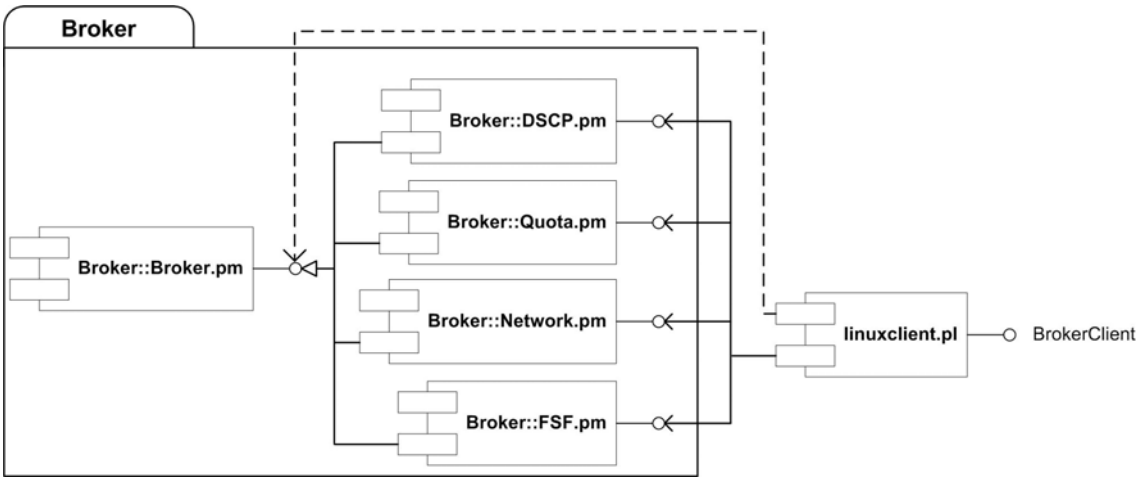
Tabelle 16: Reservierungsparameter der Komponente FSF

6.3 Linux Client

Für Linux wurde ein vollständiger Client gemäß der im Entwurf entwickelten Frameworkarchitektur implementiert. Die in Kapitel 4.2 aufgezeigten Möglichkeiten zur Ressourcenpartitionierung unter Linux verlangen häufige Interaktion mit Kommandozeilentools des Betriebssystems, aus diesem Grund erfolgte die Implementierung in der Programmiersprache Perl.

Die folgenden Abbildungen beschreiben die einzelnen Bestandteile des Linux Clients und ihre Abhängigkeiten untereinander.

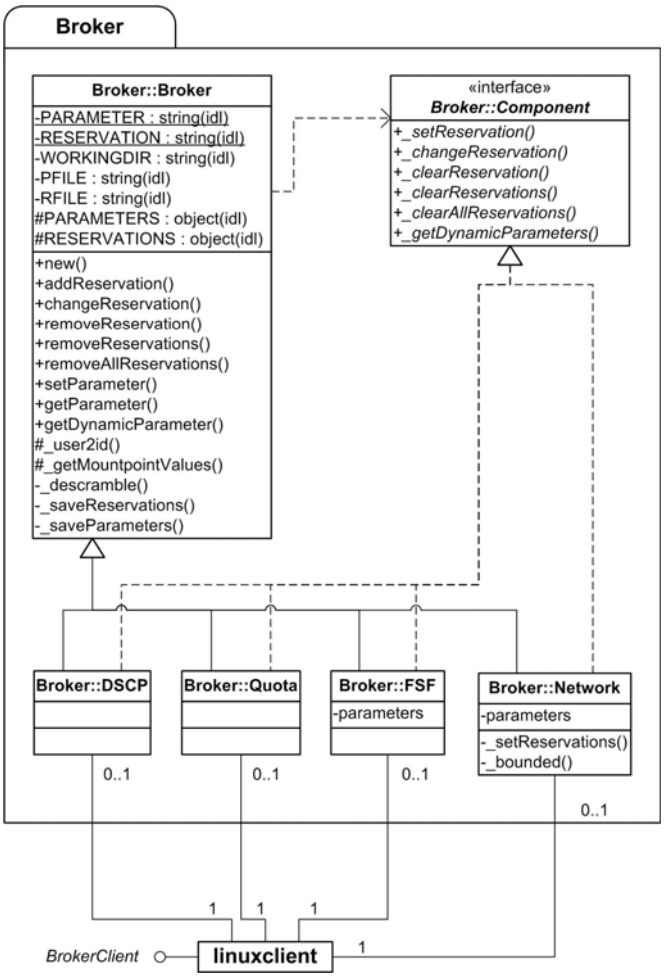
[68] Als Swapdateien werden hier diejenigen Dateien bezeichnet, die formatiert und über ein *Loop Device* eingebunden werden.



Abbildung[69] 13: Komponentenmodell des Linux Clients

In der Abbildung sind die vier entwickelten Komponenten zu erkennen, jede durch eine Klasse realisiert. Alle Komponenten sind von der Klasse `Broker` abgeleitet, die die standardisierte Schnittstelle, auf die der Broker Client zugreift, liefert. Hieraus ergibt sich der Abhängigkeitspfeil vom Broker Client (`linuxclient.pl`) zur Klasse `Broker.pm`. Alle Komponenten sowie ihre Superklasse gehören zum Paket Broker.

[69] Diagrammtyp: UML Komponentendiagramm.



Abbildung[70] 14: Klassendiagramm des Linux Clients

Die Schnittstelle (public Methoden von `Broker::Broker`) zwischen Broker Client (in der Abbildung: `linuxclient`) und den Komponenten orientiert sich bei der Namensgebung an der XML RPC Schnittstelle (beschrieben in Kapitel 5.7). Alle Methodennamen können 1:1 zugeordnet werden und mit Ausnahme des Konstruktors[71] besitzen die Komponenten keine weiteren öffentlichen (public) Methoden. In der Abbildung ist zu erkennen, dass die Komponenten nicht nur von `Broker::Broker` abgeleitet sind, sondern auch das `Broker::Component` Interface implementieren. Die implementierten Methoden werden direkt aus den in `Broker::Broker` definierten Methoden aufgerufen, deshalb besteht eine Abhängigkeit zwischen `Broker::Broker` und `Broker::Component`.

6.3.33 Broker Client

Der Quellcode vom Linux Broker Client (linuxclient.pl) ist in Anhang C zu finden. Wird der Client gestartet, werden alle im Unterverzeichnis `Broker` vorhandenen Komponenten instanziert, um nicht-persistente Reservierungen zu erneuern. Anschließend wird der XML RPC Server[72] gestartet und der Client wartet auf ankommende Nachrichten. Ein Hash definiert dabei die Zuordnung zwischen

[70] Diagrammtyp: UML Klassendiagramm.
[71] Der Konstruktor ist die `new()` Methode. Zu Objektorientierung in Perl siehe [04], Kapitel 13.
[72] Im Framework handelt es sich um einen Client, auf XML RPC Ebene jedoch um einen Server, welcher Anfragen von außen (vom RPC Client, hier der Broker Server) beantwortet.

einkommenden *remote procedure calls* und eigenen Subroutinen. Den Broker Client zeichnet aus, dass er modular ist und zur Laufzeit neue Komponenten nachgeladen werden können. Dieses wird durch das dynamische Laden von Modulen mittels des Perl Befehls `require` innerhalb der Subroutinen `_metaMethod` und `_metaMethod2` ermöglicht. Diese Methoden bekommen einen Komponentennamen und einen Methodennamen übergeben, instanziieren bei Bedarf die Komponenten (wenn diese erst nach Start des Clients in das Verzeichnis kopiert wurden) und rufen die entsprechende Methode auf. Details sind dem Quellcode zu entnehmen.

Der Broker Client muss unter der Kennung root gestartet werden, da die von den Komponenten durchgeführten Aktionen nur mit root-Rechten möglich sind.

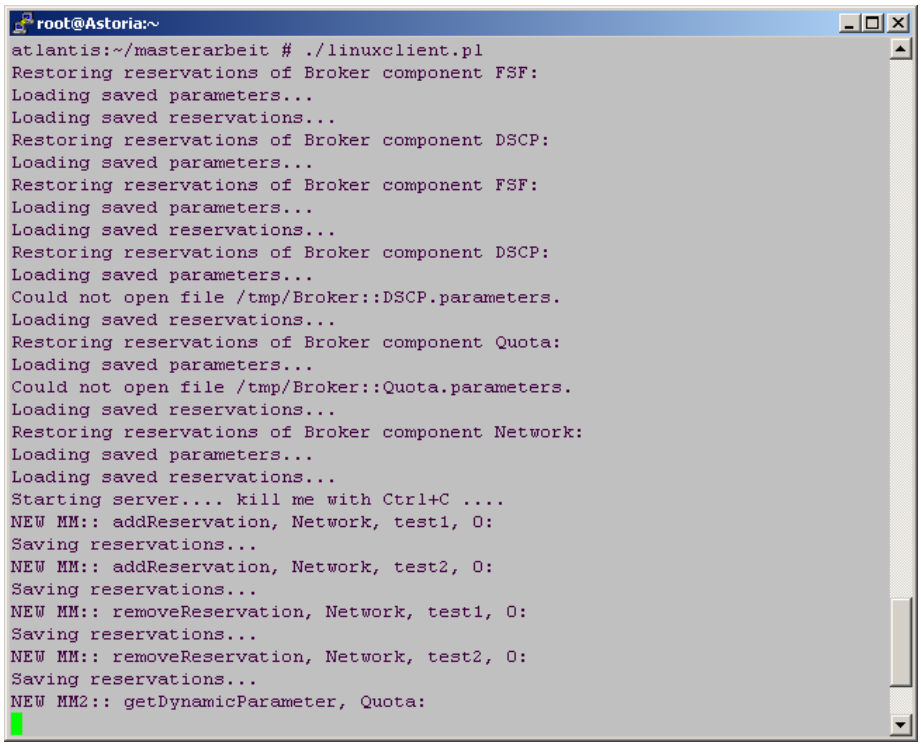


Abbildung 15: Programmlauf mit Debugausgabe des Linux Broker Clients

6.3.34 Komponenten

Für Linux wurden alle vier in Kapitel 6.2 aufgelisteten Komponenten entwickelt. Die Implementierung verwendet dabei die in Kapitel 4.2 untersuchten Mechanismen. Im Folgenden wird nur noch auf spezifische Eigenschaften eingegangen und ansonsten auf die entsprechenden Kapitel verwiesen. Details sind dem Quellcode zu entnehmen, dieser ist aus Platzgründen nicht im Anhang enthalten, sondern dem beiliegenden Datenträger zu entnehmen.

Network

Zur Benutzererkennung der Datenströme wird iptables verwendet, zur Bandbreitenbegrenzung iproute2. Beides wurde in Kapitel 0 aufgezeigt. Es wurden zwei verschiedene *Queuing Disciplines* ausgetestet, die Diskussion hierzu erfolgt in Kapitel 7.3 .

DSCP

Realisiert mittels iptables, wie in Kapitel 0 aufgezeigt.

Quota

Realisiert mittels Quota, wie in Kapitel 0 aufgezeigt.

FSF

Realisiert wie in Kapitel 0 beschrieben. Die Anzahl der *Loop Devices* ist im Allgemeinen auf 16 festgelegt, dadurch ist auch die Anzahl der Partitionierungen auf 16 begrenzt.

6.4 Windows Client

Der Windows Client wurde als reine Machbarkeitsstudie entwickelt. Er besitzt weder eine vergleichbare Architektur wie der Linux Client noch ähnlich umfangreiche Funktionalität und ist nicht komponentenbasiert. Die Funktionalität der Quota Komponente wurde in den Broker Client integriert. Als XML RPC Server wurde Apache XML RPC[73] verwendet. Beim Start bekommt dieser XML RPC Server eine Klasse (in der Implementierung: `Methodhandler`) übergeben, deren Methodensignaturen er mittels Introspection analysiert und so eine automatische Zuordnung zwischen eingehenden RPC Methodenaufrufen und Javamethoden erreicht.

Wie in Kapitel 4.3.18 beschrieben, können Quotas unter Windows XP über das Kommandozeilentool `fsutil` konfiguriert werden, es existiert jedoch keine WMI Schnittstelle in dieser Windowsversion. Interaktion mit Kommandozeilentools bedeutet in diesem Fall auch das Auswerten und Analysieren der zurückgegebenen Werte. Die Implementierung erfolgte deshalb mittels Java, da hier auf reguläre Ausdrücke zurückgegriffen werden kann[74]. Es handelt sich bei Quotas um eine persistente Partitionierung, die auch nach einem Systemneustart bestehen bleibt, aus diesem Grund speichert der Windows Client selber keine Partitionierungsinformationen.

Der Windows Client muss wie der Linux Client unter einer Kennung mit administrativen Rechten gestartet werden. Implementierungsdetails sind dem kommentierten Quellcode auf beiliegendem Datenträger zu entnehmen.

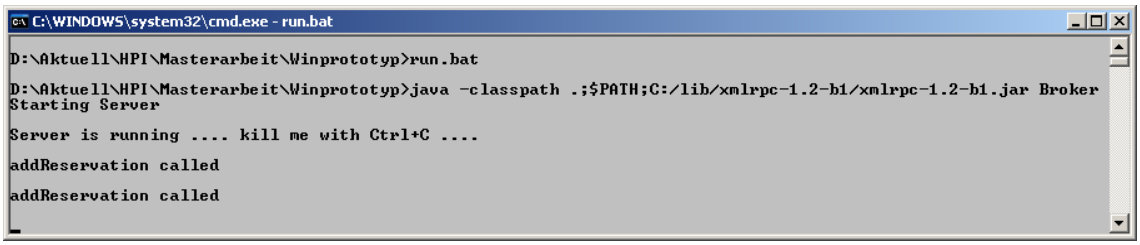


Abbildung 16: Programmlauf mit Debugausgabe des Windows Clients

[73] Apache Foundation, Java XML RPC Implementierung: http://ws.apache.org/xmlrpc/

[74] Reguläre Ausdrücke sind auch in Perl möglich. Die Implementierung wurde aus zwei Gründen in Java durchgeführt: Zum einen ist die Verwendung von Perl unter Windows aufgrund der starken Nähe der Programmiersprache zu Linux / Unix im Allgemeinen nicht zu empfehlen, zum anderen ist die Verbreitung von Java auf Desktopbetriebssystemen mit Windows höher als die Verbreitung von Perl und es kann bei Verwendung von Java somit von einem insgesamt geringeren Installationsaufwand ausgegangen werden.

6.5 Abhängigkeiten

Eine Anforderung an das Framework ist die Verwendung betriebssystemnaher Lösungen, um das Verwenden von umfangreicher und komplexer Software von dritter Seite zu vermeiden. Folgende Abhängigkeiten ergeben sich für die jeweiligen Softwarekomponenten:

Abhängigkeiten des Servers:

- Datenbanksystem MySQL
- Perl inklusive der Perlmodule:

Name	**Beschreibung**
Getopt::Std	Handling von Übergabeparametern beim Programmaufruf
POSIX	Prozess-Handling
Data::Dumper	Debugausgabe von komplexen Datenstrukturen
CGI	CGI Handling
HTML::Template	HTML Templates um HTML im Perlcode zu vermeiden
Frontier::Client	XML RPC Client
DBI	Datenbankzugriff
dbeasy.pl	Modul zum Datenbankhandling, geschrieben von Henning Meyer

  Die von den bekanntesten Linux-Distributionen mitgelieferten Perlinstallationen beinhalten bereits die Mehrzahl der benötigten Module. Bei der Installation von SuSE 9.1. muss lediglich der XML RPC Client hinzugefügt werden.

Abhängigkeiten des Linux Clients (Broker Client sowie Komponenten):

- Perl inklusive der Perlmodule:

Name	***Beschreibung***
Data::Dumper	*Debugausgabe von komplexen Datenstrukturen*
Frontier::Daemon	*XML RPC Server*
Frontier::RPC2	*Benötigt zum Typecasting in XML RPC Typen*[75]

- Quota
  Aktiviertes Quota, muss in den Kernel kompiliert werden, bei aktuellen Distributionen ist dieses der Fall.
- Quotatool
  Zum Setzen von Festplattenquotas wird das Quotatool benötigt, nicht alle Distributionen liefern es mit.
- Iproute2
  Benötigt für Traffic Shaping, muss in den Kernel kompiliert werden, bei aktuellen Distributionen der Fall. Je nach Distribution ist die *Queueing Discipline* HTB nicht vorhanden und muss nachträglich hinzugefügt werden[76].

Abhängigkeiten des Windows Clients:

- Java 1.4.1 (reguläre Ausdrücke)
- Java Apache XML RPC Server

[75] Der verwendete XML RPC Server beinhaltet eine automatische Typerkennung, die bei Zeichenketten, deren Inhalt auch als Integer interpretiert werden kann, Integer wählt.

[76] HTB Homepage: http://luxik.cdi.cz/~devik/qos/htb/

7 Evaluierung

In diesem Kapitel erfolgt die Analyse des implementierten Frameworks. Schwerpunkt der Betrachtung sind zum einen die Zuverlässigkeit und Performanz des Frameworks als Ganzes sowie die Zuverlässigkeit der Partitionierungsmechanismen. Aufgrund der in Kapitel 5.5 geschilderten Problematik, wurde in der Implementierung keine direkte Verbindung zwischen Grid-System und dem Framework berücksichtigt, da hierzu notwendige Veränderungen im Grid- bzw. Cluster-Scheduler durchgeführt werden müssen. Die derzeitige Implementierung beschränkt sich folglich auf die Durchführung der ersten Anwendungsfallgruppe, dem Durchführen von administrativen Beschränkungen. Beruhend auf der Diskussion in Kapitel 5.4 wurde sich für eine kennungsbasierte Ressourcenpartitionierung entschieden. Es wurde weder für eine Linux-Komponente noch für den Windows-Client ein Programm zum Erkennen von Grid-Prozessen benötigt[77], da alle verwendeten Mechanismen Partitionierung auf Basis der Benutzerkennung von sich aus unterstützen. Geht man davon aus oder zeigt, dass die kennungsbasierten Partitionierungsmöglichkeiten funktionieren, so kann die volle Funktionsfähigkeit des Frameworks im Kontext von *Grid Computing* gezeigt werden, ohne dass bei der Evaluierung ein Grid-System verwendet werden muss. Dieses Vorgehen ist möglich, da aufgrund der fehlenden direkten Anbindung des Frameworks an ein Grid die einzige Verbindung über die verwendeten Benutzerkennungen besteht.

7.1 Durchgeführte Test- und Anwendungsfälle

Die Testphase bei der Frameworkentwicklung unterteilte sich in drei Abschnitte. Zuerst erfolgte ein Test der entwickelten Clientsoftware, anschließend wurde die Serversoftware getestet. Als Drittes wurde das Verhalten des Frameworks als Ganzes getestet und dabei die in Kapitel 0 und 0 aufgestellten Anwendungsfälle überprüft.

7.1.35 Test der Clientsoftware

Aufgrund der standardisierten XML RPC Schnittstelle ist die Funktionsweise der Clients auch ohne die Serverimplementierung sehr gut überprüfbar. Folgende Testfälle wurden durchgeführt:

Testfall	***Status***
Starten des Linux-Clients mit automatischer Initialisierung aller vorhandenen Komponenten.	*4 x OK*
Setzen einer Partiotionierung (Eingehenden XML RPC Aufruf annehmen, Weiterleiten des Aufrufs an die Komponente, Setzen der Partitionierung, Testen ob die Partitionierung durchgeführt wurde), durchgeführt für jede der vier Komponenten	*4 x OK*
Ändern einer Partitionierung (…), durchgeführt für jede der vier Komponenten	*4 x OK*
Löschen einer Partitionierung (…), durchgeführt für jede der vier Komponenten	*4 x OK*
Löschen aller Partitionierungen eines Benutzers (…), durchgeführt für jede der vier Komponenten	*4 x OK*
Löschen aller Partitionierungen aller Benutzer (…), durchgeführt für jede der vier	*4 x OK*

[77] Die Notwendigkeit eines derartigen Mechanismus für die Verwendung von prozessbasierten Partitionierungsmöglichkeiten (wie CPU Schedulern) wurde im vorletzten Absatz in Kapitel 5.4 aufgezeigt.

Komponenten	
Testen ob der verwendete Partitionierungsmechanismus vollständig funktioniert, durchgeführt für jede der vier Komponenten	*3 x OK*[78]
Setzen eines Konfigurationsparameters, durchgeführt für alle Komponenten, die mindestens einen Konfigurationsparameter besitzen / verwenden	*Alle OK*
Abfragen der dynamischen Parameter, durchgeführt für alle dynamischen Parameter in allen Komponenten	*Alle OK*
Testen auf Fehleranfälligkeit, durchgeführt in diversen Testläufen für den gesamten Client und speziell für jede der vier Komponenten	*Teilweise OK*[79]
Test der `ping()` Funktion des Clients	*OK*
Dynamisches Nachladen einer noch nicht beim Starten des Clients vorhandenen Komponente	*OK*

Tabelle 17: Testfälle des Linux-Clients

Testfall	**Status**
Starten des Windows Client	OK
Setzen einer Partitionierung (Eingehenden XML RPC Aufruf annehmen, Setzen der Partitionierung, Testen ob die Partitionierung durchgeführt wurde)	OK
Ändern einer Partitionierung (…)	OK
Löschen einer Partitionierung (…)	OK
Löschen aller Partitionierungen eines Benutzers (…)	OK
Löschen aller Partitionierungen aller Benutzer (…)	OK
Abfragen der dynamischen Parameter	Alle OK
Test der `ping()` Funktion des Clients	OK
Testen auf Fehleranfälligkeit, durchgeführt in diversen Testläufen für den gesamten Client	Teilweise OK

Tabelle 18: Testfälle des Windows-Clients

Mit Ausnahme der in Tabellen verzeichneten Probleme, wurde kein Fehlverhalten aufgedeckt. Das Durchführen der XML RPC verwendenden Testfälle erfolgte mit dem Skript `prototypS01.pl` (enthalten auf dem beiliegenden Datenträger).

7.1.36 Test der Serversoftware

Es erfolgte ein Test aller in Abbildung 9 (Seite 56) aufgelisteten Anwendungsfälle sowie die Überprüfung der Funktionalität des Database Crawlers. Die Überprüfung erfolgte im Zusammenspiel mit mehreren Clients und konnte die in Kapitel 6.1.28 vorgestellte Funktionalität verifizieren[80].

7.1.37 Durchführbare Anwendungsfälle

Anhand der bereits durchgeführten Testfälle konnte gezeigt werden, dass die Funktionalität des Frameworks als Ganzes gegeben ist. Im Folgenden werden die in Kapitel 0 und 0 aufgezeigten Anwendungsszenarien aufgegriffen und es wird überprüft, in wie weit sie mit dem vorliegenden Framework realisiert werden können.

	Anwendungsfall	***Realisierbarkeit***
1	*Beschränkung der verwendeten Festplattenbelegung um sicherzustellen,*	*Durchführbar (unter Windows XP und Linux)*

[78] Der verwendete Mechanismus zum Partitionieren der Bandbreite arbeitet nicht immer wie gewünscht. Eine ausführliche Evaluierung der *Traffic Shaping* Fähigkeiten von iproute2 erfolgt in Kapitel 7.3 .

[79] Auf die Implementierung einer vollständigen Fehlerbehandlung wurde verzichtet. Das gleiche gilt für den Windows-Client.

[80] Aufgrund der hohen Anzahl von Testfällen wird auf eine Auflistung aller durchgeführten Testfälle für die Serverimplementierung verzichtet.

	dass genügend Kapazität für Programmdaten anderer Anwendungen sowie für zukünftige Installationen vorhanden ist. Ebenfalls dient dieses dem Schutz vor einem vollständigen Belegen der Festplatte, was unter Umständen zu einer Beeinträchtigung der Systemstabilität führen kann.	
2	*Beschränkung des verwendeten Arbeitsspeichers um die Lauffähigkeit des Systems und anderer Anwendungen sicherzustellen.*	*Nicht durchführbar. Aufgrund der fehlenden Partitionierungsmöglichkeit unter Linux wurde keine Komponente zur Speicherpartitionierung entwickelt.*
3	*Beschränkung der durch einen Job auf einer Anlage verwendeten CPU Zeit um anhaltende Dauerlast zu vermeiden und um außer Kontrolle geratene Endlosprozesse zu verhindern.*	*Nicht durchführbar. Aufgrund der fehlenden Partitionierungsmöglichkeit unter Linux wurde keine Komponente zur CPU Zeit-Partitionierung entwickelt.*
4	*Beschränkung der lokal verwendeten Bandbreite beim Senden von Daten um Netzwerkverkehr anderer Anwendungen, besonders von Anwendungen eines lokalen Benutzers, nur gering zu beeinflussen.*	*Durchführbar (unter Linux mit den in Kapitel 7.3 analysierten Einschränkungen).*
5	*Beschränkung des netzweiten Datenverkehrs um zu verhindern, dass der von Jobs des Grid-Computings erzeugte Netzwerkverkehr keine Netzelemente wie NFS Server oder Netzwerkknoten (z.B. Switches, Router) blockiert.*	*Zur Realisierung einer netzweiten Bandbreitenpartitionierung wird ein netzweiter Mechanismus benötigt. Die Implementierung unterstützt unter Linux das Setzen von DSCPs in IP-Headern und unterstützt somit* Differentiated Services*.*

Tabelle 19: Anwendungsfälle zur administrativen Beschränkung

	Anwendungsfall	***Realisierbarkeit***
1	*Forderung nach einer bestimmten verfügbaren lokalen Festplattenkapazität für temporäre zur Laufzeit des Jobs erzeugten Daten.*	*Durchführbar (unter Linux mit den in Kapitel 0 aufgezeigten Einschränkungen)*
2	*Forderung nach einer bestimmten Menge von verfügbarem Arbeitsspeicher.*	*Nicht durchführbar.*
3	*Forderung nach verfügbarer Netzwerkbandbreite um bestimmte Datenmengen in begrenzter Zeit zu transferieren.*	*Durchführbar (unter Linux mit den in Kapitel 7.3 analysierten Einschränkungen).*
4	*Allgemeine Forderung nach garantierten Minimalanteilen an allen Ressourcen. Diese Forderung gründet darauf, dass ein Job nicht vollständig zum Erliegen kommen soll, selbst wenn seitens lokaler Aktivitäten eine vollständige Ressourcenauslastung erreicht wird.*	*Nur für die Ressourcen Festplattenkapazität und Netzwerkbandbreite durchführbar, aufgrund fehlender Partitionierungsmöglichkeiten ist derzeit keine Garantie von Prozessorkapazität möglich.*

Tabelle 20: Anwendungsfälle zur Garantie von Mindestressourcen

Aufgrund der nicht realisierten direkten Anbindung zwischen Framework und Grid-System ist die zweite Anwendungsfallgruppe nur durch manuelles Partitionieren über

die Weboberfläche möglich und kann nicht durch das Grid-System automatisch erfolgen.

7.2 Evaluierung der Frameworkimplementierung

Das in dieser Arbeit entworfene und teilweise in der Implementierung realisierte Framework hat sich beim Testbetrieb als sehr zuverlässig und stabil erwiesen. Mittels der in Kapitel 6.1.28 aufgezeigten Zustandsvariablen und den dazugehörigen Zustandsübergängen konnte eine optimale Synchronität zwischen dem Broker Server und den Clients erreicht werden. Der Anwender bzw. das Anwenderinterface ist unabhängig von dem jeweiligen Online/Offline Zustand der Clients. Dadurch können in Verbindung mit den Zustandsvariablen auch komplexere Operationen (z.B. Löschen eines Knotens inklusive aller Kindknoten) von Benutzerseite aus durchgeführt werden, obwohl nicht alle beteiligten Knoten zu dem Zeitpunkt vom Server erreicht werden können. Ungewollte Zustände wie das Vorhandensein von Reservierungen, obwohl der Knoten bereits in der Datenbank gelöscht wurde, sind nicht möglich und im Testbetrieb auch nicht aufgetreten. Hierbei ist zu beachten, dass dies nur dann gewährleistet ist, wenn auf den Clients kein direkter Zugriff auf die Partitionierungsmechanismen erfolgt und so keine Inkonsistenzen auf Clientseite erzeugt werden.

7.2.38 Performanz der Serverimplementierung

Die Performanz des Frameworks hat sich mehrheitlich als unkritisch herausgestellt, jedoch gibt es einen Punkt, welcher die Performanz bei ungünstigen Verhältnissen erheblich beeinträchtigen kann: Zur Überprüfung des Online/Offline-Zustandes eines Clients sowie zur Synchronisierung wird jeweils ein neuer Prozess pro Knoten erzeugt, welcher dann für die Kommunikation mit dem Client zuständig ist. Aufgrund der Ausführzeit der *remote procedure calls* ist es notwendig, mehrere Verbindungen parallel aufzubauen, da sonst die Anzahl der Aktionen des Servers pro Zeiteinheit stark begrenzt wären. Die verwendete XML RPC Lösung lässt dieses jedoch nicht zu, da jeder von ihr abgesetzte RPC-Aufruf blockierend ist. Aus diesem Grund wurde eine Lösung mit mehreren, parallelen Prozessen verwendet, wodurch sich, abhängig von verwendeter Hardware sowie der Anzahl der parallelen Prozessen, eine mitunter sehr hohe Auslastung des Servers ergibt, die dazu führen kann, dass Anfragen seitens des Benutzers nicht mehr zeitgerecht bearbeitet werden können. Dieses Verhalten ist jedoch stark hardwareabhängig. Bei der Implementierung und Evaluierung wurde ein 200MHz Pentium MMX mit 128 MB RAM Arbeitsspeicher als Server verwendet und es ist nicht ungewöhnlich, dass bei dieser Konstellation mit laufendem Apache und MySQL relativ schnell eine Vollauslastung erreicht wird. Auch wenn sich die Anzahl der Prozesse, bis ein kritischer Zustand erreicht wird, mit aktueller Hardware vervielfachen lässt, so wäre eine Lösung ohne das Erzeugen neuer Prozesse performanter. Es ist jedoch keine XML RPC API für Perl bekannt, mit der asynchrone RPC Aufrufe möglich sind. Mit Ausnahme des eben beschriebenen Verhaltens erwies sich das Framework als hinreichend performant bezüglich der in Kapitel 3.2 aufgestellten Anforderungen. Die Zeit zwischen Abschicken der Partitionierung seitens des Benutzers und erfolgter Parititionierung auf dem Client ist im Wesentlichen abhängig von der Aktivität des Database Crawlers sowie von der Latenzzeit des Netzes und der Bearbeitungszeit beim Client. Liegt auf dem Server keine Überlast vor, erfolgt die Partitionierung innerhalb weniger Sekunden, was für alle Anwendungsszenarien mehr als

ausreichend ist. Insgesamt darf bei der Betrachtung der Performanz nicht vergessen werden, dass es sich um eine prototypische Implementierung handelt, deren Schwerpunkt nicht auf Performanz, Sicherheit und Fehlerbehandlung liegt.

7.2.39 Zuverlässigkeit der Partitionierungen

Die Zuverlässigkeit der durchgeführten Partitionierungen hängt sowohl von den Komponenten als auch von den verwendeten Mechanismen ab. Wie bereits in Kapitel 7.1 gezeigt, laufen alle implementierten Komponenten unter Linux- sowie der Windows-Client stabil und es werden weder Fehler noch Inkonsistenzen bei der Partitionierung erzeugt. Auch die folgenden Neustartsituationen wurden von allen Komponenten fehlerlos bewältigt:

- Neustart des Broker Clients:

  Theoretisch müssen auch transiente Partitionierungen nicht erneuert werden, jedoch kann die Komponente Network unter Linux dem System nicht entnehmen, welche *traffic shaping* Regeln gerade aktiv sind. Diese Komponente führt deshalb bei jedem Start ein Löschen aller *traffic shaping* Regeln durch, gefolgt von einem anschließenden Setzen der Partitionierungen.

- Neustart des Rechners und anschließend des Broker Clients:

  Hierbei müssen transiente Partitionierungen wiederhergestellt werden, dieses betrifft unter Linux die Komponenten DSCP (iptables-Einträge setzen), Network (iptables-Einträge setzen und *traffic shaping* Regeln setzen) sowie FSF (erneutes Mounten der Swapdatei).

Die von den Komponenten verwendeten Mechanismen funktionieren mit partieller Ausnahme von iproute2 (eine Evaluierung erfolgt in Kapitel 7.3) ebenfalls korrekt und weisen keine Fehler auf. Abschließend kann festgestellt werden, dass alle Komponenten das gewünschte Verhalten aufweisen und die mit ihnen gesetzten Partitionierungen die gestellten Anforderungen erfüllen.

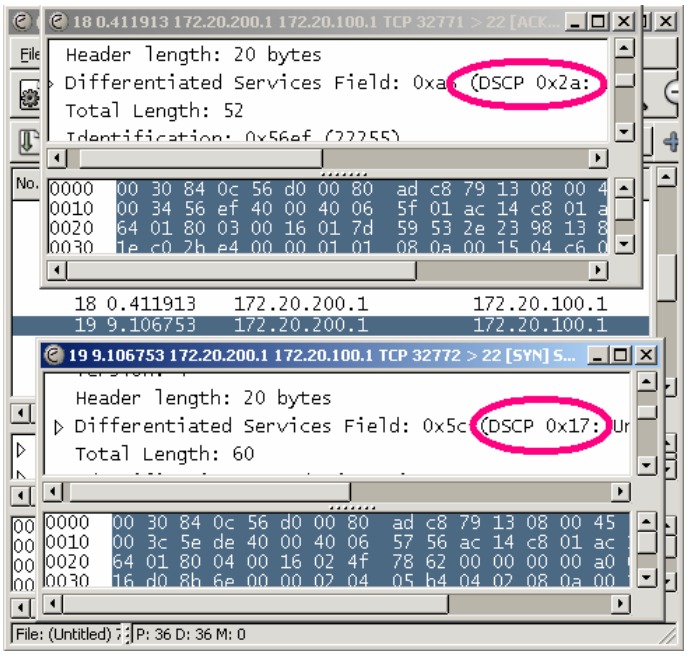


Abbildung 17: Verschiedene DSCPs in IP Headern

In der Abbildung sind Details zu zwei direkt hintereinander von 172.20.200.1 nach 172.20.100.1 (erkennbar in der Mitte des Bildes / im Hintergrundfenster) gesendeten IP Paketen zu erkennen. Beide weisen verschiedene DSCPs im IP-Header auf, da sie von zwei verschiedenen Benutzern gesendet worden sind, denen über das Framework ein bestimmter DSCP zugewiesen worden war.

7.3 Evaluierung von Traffic Shaping mit iproute2

In diesem Kapitel erfolgt eine Überprüfung der Qualität der mit iproute2 erreichbaren Bandbreitenpartitionierungen. Die folgende Grafik[81] zeigt zwei Datenströme ohne Partitionierung und es ist erkennbar, dass die Bandbreite gleichmäßig aufgeteilt wird. Bei dieser wie auch bei den noch folgenden Grafiken sind neben den jeweils aufgezeigten Besonderheiten in allen Graphen kleinere Oberschwingungen zu erkennen. Hierfür gibt es mehrere Gründe. Zum einen entstehen sie durch das Zusammenspiel der beteiligten Hard- und Software auf Sender- und Empfängerseite, durch die sich keine dauerhaft 100%ig konstanten Datenraten ergeben, zum anderen führt das Messen[82] mittels IPAC-NG[83] selbst zu einer leichten Beeinflussung der Messwerte.

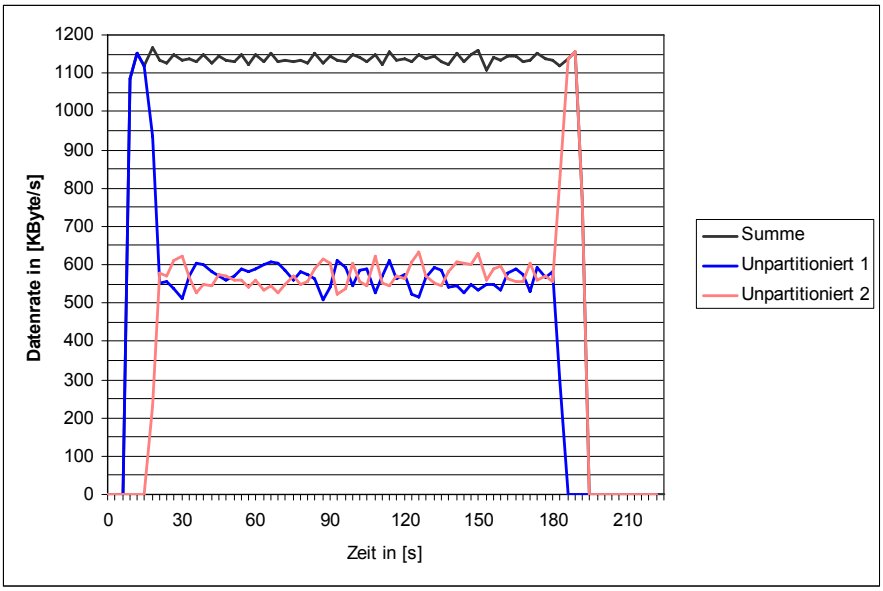


Abbildung 18: Zwei unpartitionierte, parallele Datenströme

In Linux werden alle Pakete vor dem Senden in einer Queue zwischengespeichert. Diese Queue wird vom Kernel nach einem Queueing-Mechanismus geleert und die entnommenen Pakete werden über das Netzwerkinterface gesendet. Um Bandbreitenmanagement unter Linux zu betreiben, implementiert iproute2 mehrere dieser Mechanismen, im Folgenden *Queueing Disciplines* genannt. Die

[81] Diese und alle weiteren Messungen wurden auf einem Pentium III mit 500 MHz, 384 MB RAM unter SuSE Linux 9.1, Kernel 2.6.4-52 mit einer ISA Netzwerkkarte mit 10MBit/s durchgeführt.

[82] Die Messungen wurden im Dreisekundenabstand durchgeführt. Hierbei hat sich gezeigt, dass die Laufzeit von `fetchipac` (dem IPAC-NG Kommandozeilentool zum Messen der Daten) zwischen wenigen Millisekunden bis hin zu 500ms schwankt. Trotz Miteinberechung dieser Schwankung ist eine hierdurch bedingte Verfälschung der Messewerte nicht auszuschließen.

[83] Source Forge, IPAC-NG Homepage: http://ipac-ng.sourceforge.net/

unterschiedliche Behandlung verschiedener Datenströme ist mit *Classful Queueing Disciplines* möglich, hierbei können jeder Klasse gesonderte Regeln zugewiesen werden. Zur vollständigen Bandbreitenpartitionierung gehört sowohl das Begrenzen der Datenrate (das so genannte Shaping), als auch das Garantieren von Bandbreite. In Kombination von beidem ergibt sich, dass die Datenrate auf einen bestimmten Wert fixiert werden muss. Dieses wird von den beiden folgenden *Queuing Disciplines* realisiert:

- CBQ (*Class Based Queuing*, Technische Details: [22] Kapitel 9.5.4)
- HTB (*Hierarchical Token Bucket*, Technische Details: [22] Kapitel 9.5.5)

Im Folgenden werden beide *Queueing Disciplines* anhand ausgewählter Szenarien miteinander verglichen. Es wird hierbei nicht auf die internen Mechanismen eingegangen (siehe dazu die angegebenen Quellen), sondern allein auf die Qualität der Partitionierungen[84]. Die in den Szenarien dargestellten Datenströme wurden manuell gestartet, deswegen herrscht keine Deckungsgleichheit der Anfangszeiten, dieses ist jedoch auch nicht von Belang.

7.3.40 Szenario 1

Ziel des Szenarios ist die Überprüfung, inwieweit die beiden *Queueing Disciplines* einfache Partitionierungen durchführen. Hierzu wurden mit der Frameworkimplementierung zwei Partitionierungen (für zwei Benutzer) gesetzt, eine à 750Kbyte/s und eine à 112,5 Kbyte/s. Beide Datenströme sind dabei strikt auf diesen Wert begrenzt, können also keine weitere Kapazität nutzen. Parallel zu den zwei begrenzten Datenströmen wurde ein nicht partitionierter Datenstrom gestartet.

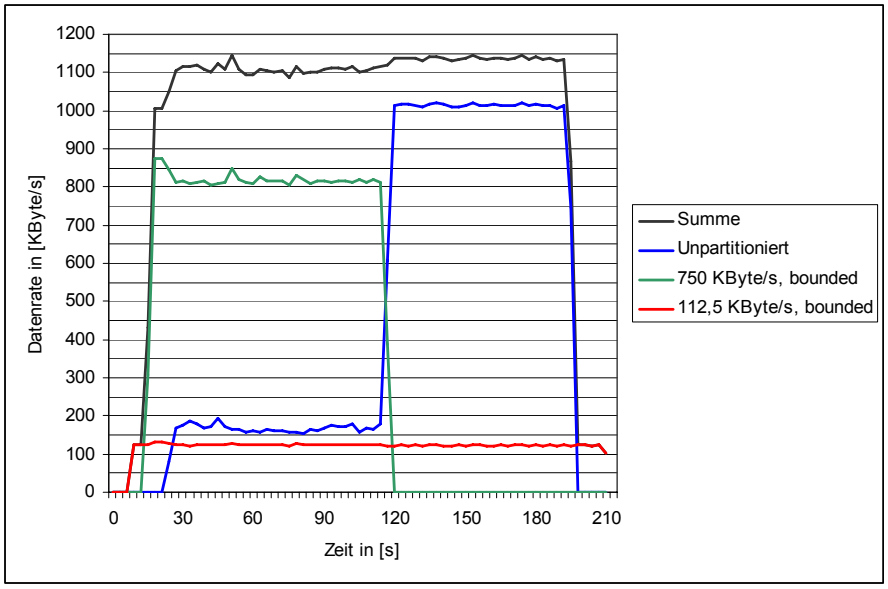


Abbildung 19: Szenario 1 mit CBQ

[84] An dieser Stelle sollte erwähnt werden, dass die in den folgenden Diagrammen vorkommenden unpartitionierten Datenströme indirekt auch partitioniert sind: Bei *Classfull Queueing Disciplines* (dazu gehören CBQ und HTB) werden Datenpakete zu Klassen mit einer Datenrate zugeordnet. Pakete die von keinem speziellen Filter erfasst werden, werden einer Defaultklasse zugeordnet und auch diese hat eine Datenrate. Innerhalb der Komponente Network wird dieser Klasse die Geschwindigkeit des Netzwerkinterfaces abzüglich der Summe der Datenraten aller Partitionierungen zugeordnet.

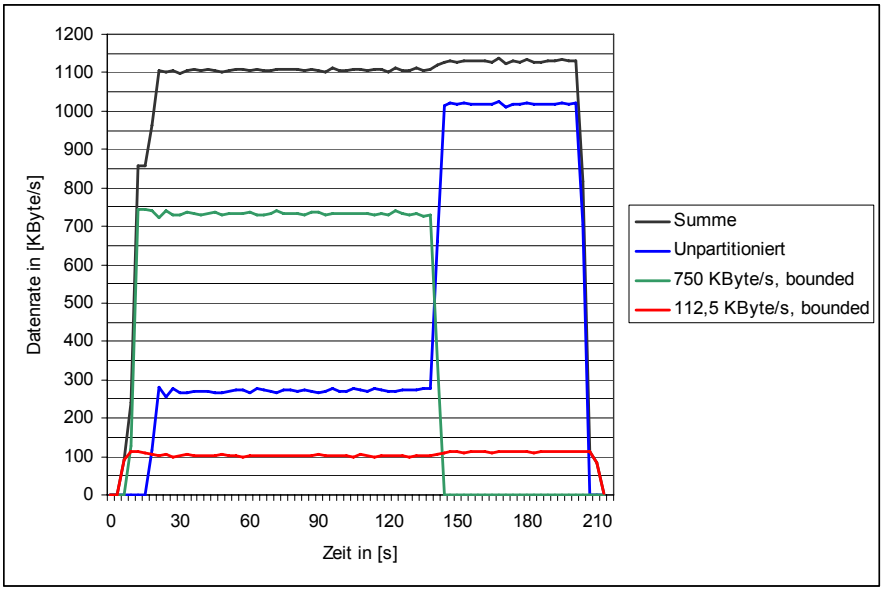


Abbildung 20: Szenario 1 mit HTB

Folgende Beobachtungen lassen sich treffen:

- Begrenzte Datenströme bei CBQ haben einen bis zu 15% höheren Durchsatz als verlangt (am 750Kbyte/s-Graphen erkennbar), dadurch ergibt sich ein geringerer Durchsatz des nicht partitionierten Datenstroms. Dieses Verhalten ist mit bekannten Timing-Problemen von CBQ zu erklären, hierauf wird in Kapitel 7.3.43 eingegangen.
- Bei HTB haben begrenzte Datenströme einen leicht niedrigeren Durchsatz als verlangt, jedoch beträgt die Abweichung weniger als 5%, sodass sie vernachlässigbar erscheint.
- Beide Graphen haben einen höheren Durchsatz wenn weniger parallele Datenströme vorliegen (zu erkennen am Anstieg des Gesamtgraphen ab dem Zeitpunkt wo der 750Kbyte/s-Graph auf Null geht). Je niedriger die Anzahl der parallelen Datenströme ist, desto weniger Overhead entsteht beim Queueing und der Kernel ist in der Lage eine (wenn auch nur geringfügig) höhere Gesamtdatenrate zu erzeugen.
- Der Durchsatz des 112,5Kbyte/s-Graphen bleibt bei CBQ über die Zeit konstant. Es gibt keine Veränderung ab dem Zeitpunkt wo der 750Kbyte/s-Graph auf Null geht. Bei HTB dagegen erhöht sich der Durchsatz leicht ab diesem Zeitpunkt. Anscheinend erfolgt bei HTB der erhöhte Overhead zu Lasten aller Datenströme, während bei CBQ nur der unpartitionierte Datenstrom beeinträchtigt wird.
- CBQ hat einen insgesamt leicht höheren Gesamtdurchsatz, vermutlich durch den unterschiedlich großen Overhead beider *Queueing Disciplines* begründet.
- Die Datenraten der einzelnen Datenströme sind bei HTB einer geringeren Schwankungsbreite unterworfen. Vermutlich sind auch hier die noch in Kapitel 7.3.43 aufgezeigten Timing-Probleme schuld.

Beide *Queueing Disciplines* erfüllen die an sie gestellten Anforderungen mit leichten Abweichungen.

7.3.41 Szenario 2

Dieses Szenario ist eine anspruchsvollere Variante des ersten. Es werden vier strikt begrenzte Datenströme à 100Kbyte/s, 200 Kbyte/s, 300 Kbyte/s und 400 Kbyte/s gesendet, zuzüglich existiert wie im ersten Szenario noch ein nicht partitionierter Datenstrom.

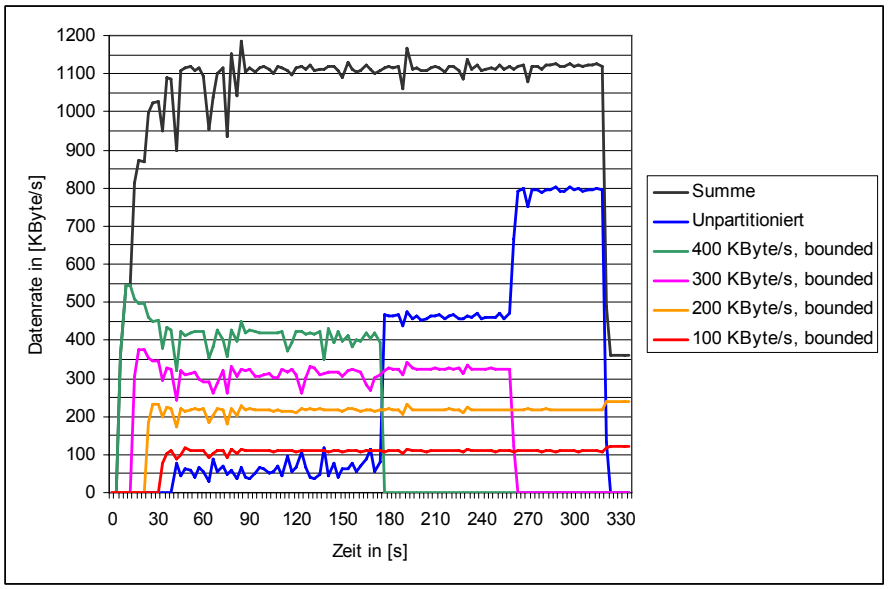


Abbildung 21: Szenario 2 mit CBQ

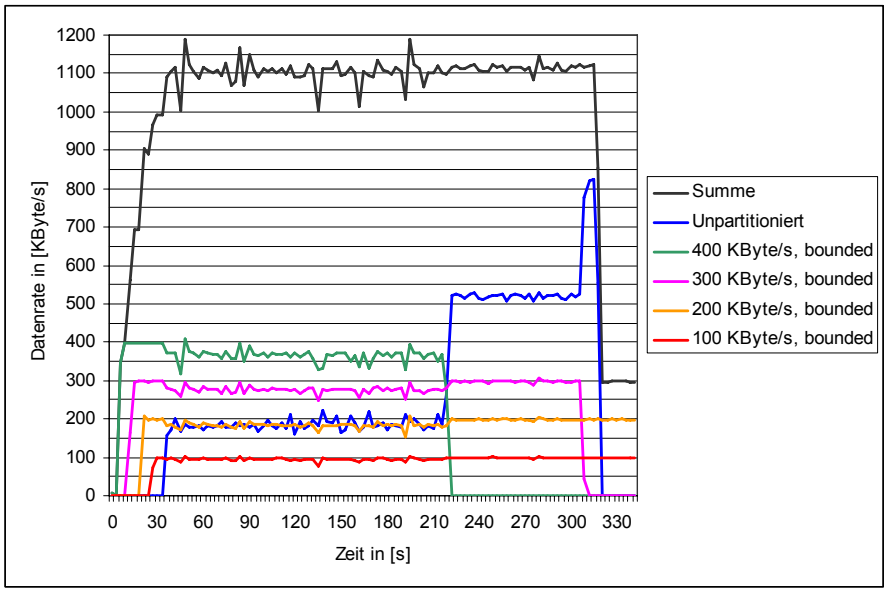


Abbildung 22: Szenario 2 mit HTB

Folgende Beobachtungen lassen sich treffen:

- Wie im ersten Szenario ist zu erkennen, dass bei CBQ die partitionierten Datenströme eine höhere Datenrate als angestrebt erhalten und bei HTB das Gegenteil der Fall ist. Im Vergleich zu Szenario 1 fällt auf, dass die

Abweichungen bei HTB jetzt deutlich höher sind (beim 400Kbyte/s-Graph sind es bis zu 10%). Der mit der Anzahl der parallelen Verbindungen steigende Overhead scheint auf HTB höhere Auswirkungen zu haben und diese *Queueing Discipline* stärker zu beeinflussen als bei CBQ. Aufgrund der Abweichungen von den angestrebten Datenraten ergibt sich in Summe eine deutlich spürbare Auswirkung auf den nicht partitionierten Datenstrom, dieser hat bei HTB die dreifache Datenrate als bei CBQ (erkennbar im Zeitraum 50s bis 160s).

- Bei CBQ haben die Datenströme zu Beginn eine deutlich höhere Datenrate, welche dann auf den gewünschten Wert (± der bereits erwähnten Abweichungen) abfällt. Die Gründe hierfür sind unklar, vermutlich liegt es an Timing-Problemen (Kapitel 7.3.43). CBQ verwendet den Zeitabstand zwischen Hardwarerückmeldungen zur Berechnung der Datenrate und es ist zu vermuten, dass sich dieses System zwischen Hard- und Software erst stabilisieren muss, bevor die angestrebte Datenrate erreicht wird.
- Sowohl bei CBQ als auch bei HTB lässt sich beobachten, dass die Oszillation der einzelnen Datenströme proportional mit der Anzahl der parallelen Datenströme steigt, d.h. je mehr Datenströme vorliegen, desto stärker schwanken die einzelnen Datenraten. Zum einen liegt dieses am steigenden Overhead beim Queueing, zum anderen ist das schon erwähnte Problem im Messverfahren dafür mitverantwortlich. Eine steigende Anzahl von parallelen Datenströmen führt zu einem höheren Schwankungsbereich in der Ausführzeit von `fetchipac`. Schlagen alle Graphen im Diagramm in eine Richtung aus, so ist dieses auf Ungenauigkeiten im Messverfahren zurückzuführen.
- Der Gesamtdurchsatz ist bei beiden *Queueing Disciplines* ungefähr gleich.

Wie in Szenario 1 erfüllen beide *Queueing Disciplines* die an sie gestellten Anforderungen mit leichten Abweichungen. Der Einfluss auf die Datenrate des nicht partitionierten Datenstroms ist inzwischen sehr deutlich zu erkennen.

7.3.42 Szenario 3

Die bisherigen Szenarios haben sich mit begrenzten Partitionierungen beschäftigt, in denen die Datenströme auch bei freien Kapazitäten auf die vorgegebene Datenrate limitiert sind. Um eine maximale Auslastung des lokalen Netzwerkanschlusses zu erreichen, kann freie Kapazität von den ansonsten begrenzten Datenströmen verwendet werden. In der Komponente Network ist diese Möglichkeit vorgesehen, siehe hierzu Tabelle 10 (Seite 61). Der boolesche Parameter `BOUNDED` gibt an, ob der jeweilige Datenstrom strikt begrenzt sein soll oder ob er freie Kapazität verwenden kann. In beiden *Queuing Disciplines* gibt es einen Mechanismus um dieses zu ermöglichen: Während bei CBQ nur angegeben werden kann, ob die Datenrate strikt begrenzt sein soll oder nicht, kann bei HTB neben der Datenrate ein zusätzlicher Maximalwert angegeben werden. Zur Beibehaltung der Kompatibilität beider Komponentenversionen wird bei HTB der Maximalwert im Bedarfsfall[85] auf die Geschwindigkeit der Netzwerkkarte gesetzt[86]. Hierdurch ist

[85] Der Maximalwert wird nur gesetzt, wenn es keine strikte Begrenzung geben soll. Bei strikter Begrenzung wird kein Maximalwert angegeben.

erreicht, dass die beiden Komponentenversionen kompatibel sind und ohne Veränderungen in der Komponentendefinition ausgetauscht werden können.

Bei den jetzt folgenden Messungen wurde ein häufig vorkommendes Szenario in *Grid Computing* abgebildet: Ein Grid-Job schickt einen großen und lang anhaltenden Datenstrom, während parallel dazu mehrere kleinere Datenströme von anderen Prozessen gesendet werden. Um den Einfluss des Grid-Computings auf die anderen Datenströme möglichst gering zu halten, um ihm aber auch gleichzeitig eine Mindestgarantie zu gewähren, wird dieser Datenstrom auf eine bestimmte Bandbreite begrenzt, wobei ihm aber auch die Möglichkeit gegeben wird, ungenutzte Kapazitäten zu verwenden.

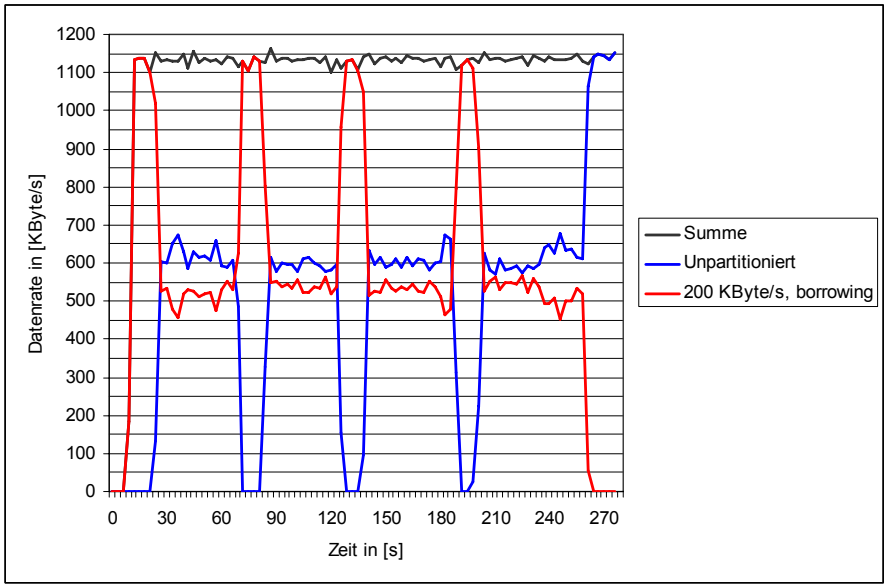


Abbildung 23: Szenario 3 mit CBQ[87]

[86] Die in Fußnote 84 beschriebene Defaultklasse für nicht partitionierte Datenströme ist grundsätzlich nicht strikt begrenzt, d.h. sie besitzt eine Datenrate (die ihr garantiert ist), jedoch besteht für die ihr zugeordneten Datenströme immer die Möglichkeit, ungenutzte Kapazitäten zu verwenden.

[87] Das Gegenteil von bounded wird in der Literatur als borrowing bezeichnet auch wenn der Begriff zu Irritationen führen kann, da die ausgeliehene Bandbreite nicht zurückgegeben wird.

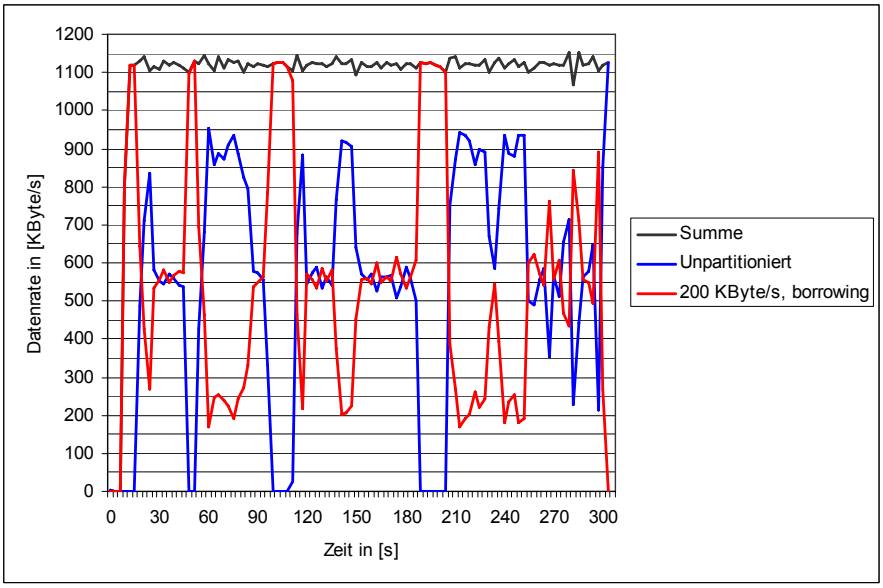


Abbildung 24: Szenario 3 mit HTB

Folgende Beobachtungen lassen sich treffen:

- CBQ realisiert ansatzweise das gewünschte Verhalten. Werden keine weiteren Daten parallel zum partitionierten Datenstrom gesendet, so nutzt dieser, wie gewünscht, die volle Kapazität aus. Allerdings wird bei einsetzender Übertragung eines nicht-partitionierten Datenstroms die Datenrate des partitionierten Datenstroms nicht auf die angestrebte Datenrate gesenkt, sondern die Datenrate liegt mehr als 150% über dieser. Der Mechanismus scheint insgesamt zu funktionieren, denn im Vergleich zu Abbildung 18 (Seite 74) ist deutlich zu erkennen, dass es sich nicht, wie man vermuten könnte, um zwei gleichwertige Datenströme handelt, allerdings arbeitet der Mechanismus sehr ungenau. Dieses Verhalten konnte mit unterschiedlichen Datenraten reproduziert werden.

- HTB weist in diesem Szenario ein sehr vom Idealverlauf abweichendes Verhalten auf. Unmittelbar nach Beginn des nicht-partitionierten Datenstroms sowie seltener auch zwischendurch wird das gewünschte Verhalten für einen kurzen Zeitraum erreicht (siehe unter Anderem bei 60s bis 80s, 140s und 210s bis 230s). Jedoch scheinen anschließend (bei 30s bis 40s und 160s bis 180s) beide Datenströme gleichberechtigt behandelt zu werden, da sie in diesen Phasen ungefähr die gleiche Datenrate aufweisen. Gegen Ende der Messung zeigt sich ein sehr instabiles Systemverhalten. Es kommt zu starken Oszillationen und zu einer zum Teil deutlich höheren Datenrate des partitionierten Datenstroms im Vergleich zur Datenrate des nicht-partitionierten Datenstroms. Dieses Verhalten konnte bei weiteren Messungen bestätigt werden. Insgesamt wirkt das Verhalten von HTB bei Verwendung dieser Funktionalität sehr instabil.

Die möglichen Ursachen der Abweichungen vom Idealverhalten werden im folgenden Kapitel aufgezeigt.

7.3.43 Fazit

In den beiden ersten Szenarios zeigen beide *Queueing Disciplines* keine größeren Mängel auf und das gewünschte Verhalten konnte nachgewiesen werden. Die Datenrate wird mit maximal ± 15% annähernd richtig partitioniert, dieses Verhalten ist zumindest bei CBQ bekannt und hat technische Gründe, siehe hierzu [22], Kapitel 9.5.4:

> *As said before, CBQ is the most complex qdisc available, the most hyped, the least understood, and probably the trickiest one to get right. This is not because the authors are evil or incompetent, far from it, it's just that the CBQ algorithm isn't all that precise and doesn't really match the way Linux works.*
>
> *Besides being classful, CBQ is also a shaper and it is in that aspect that it really doesn't work very well. It should work like this. If you try to shape a 10mbit/s connection to 1mbit/s, the link should be idle 90% of the time. If it isn't, we need to throttle so that it is idle 90% of the time.*
>
> *This is pretty hard to measure, so CBQ instead derives the idle time from the number of microseconds that elapse between requests from the hardware layer for more data. Combined, this can be used to approximate how full or empty the link is.*
>
> *This is rather tortuous and doesn't always arrive at proper results. For example, what if the actual link speed of an interface that is not really able to transmit the full 100mbit/s of data, perhaps because of a badly implemented driver? A PCMCIA network card will also never achieve 100mbit/s because of the ways the bus is designed – again, how do we calculate the idle time?*

Szenario 3 konnte von keiner der beiden *Queueing Disciplines* befriedigend bewältigt werden. Bei CBQ werden die eben zitierten technischen Gründe dafür verantwortlich sein, die Gründe für das Nichtfunktionieren bei HTB sind unbekannt. Zu vermuten ist ein Bug in HTB. Es existiert ein Kernelpatch für den 2.4er Kernel, jedoch wird im Internet berichtet[88], dass auch der 2.6er Kernel (≤ 2.6.8.1) gepatcht werden muss. Auch ein reger Emailaustausch mit Martin Devera (dem Entwickler von HTB) konnte nicht zur Lösung des Problems führen.

HTB ist erst seit kurzem in iproute2 integriert und nicht alle Distributionen beinhalten diese *Queueing Discipline* in einer aktuellen Version. Folglich ist ein Einsatz im Kontext von *Grid Computing* zweifelhaft, da keine manuelle Anpassung der Kernels auf mehreren Dutzend Maschinen erfolgen sollte. Beide Varianten der Komponente Network sind auf dem beiliegenden Datenträger vorhanden, sodass je nach Bedarf und Möglichkeit die entsprechende *Queueing Discipline* verwendet werden kann.

[88] Gelesen auf http://www.mail-archive.com/lartc@mailman.ds9a.nl/msg10370.html

8 Zusammenfassung und Ausblick

In diesem Kapitel erfolgen eine abschließende Beurteilung der geleisteten Untersuchungen sowie ein Ausblick auf mögliche Weiterentwicklungen.

8.1 Fazit

Ressourcenpartitionierung auf Arbeitsplatzrechnern ist im Kontext von *Grid Computing* ein wichtiger Aspekt, um ein faires paralleles Arbeiten von Arbeitsplatznutzern und Grid-Jobs zu ermöglichen. Es konnte gezeigt werden, dass Standardbetriebssysteme im Arbeitsplatzumfeld generell die Möglichkeit bieten, Ressourcen zu partitionieren und zum Teil die dafür notwendigen Mechanismen bereits mitliefern. Zur Ressourcenpartitionierung in einem heterogenen Netzwerk bedarf es eines betriebssystemunabhängigen Frameworks, das die benötigten Partitionierungen auf den entsprechenden Rechnern durchführt. Ein in Bezug auf seine Einsatzmöglichkeiten vergleichbares Framework, wie das hier vorgestellte, ist bisher noch nicht entwickelt worden. Das hier vorliegende Framework überzeugt durch seine Unabhängigkeit gegenüber Clientbetriebssystemen sowie gegenüber einzelnen Ressourcen und bietet durch seine Komponentenarchitektur eine sehr flexible Möglichkeit, Ressourcen unterschiedlichster Art zu partitionieren. Zwei Anwendungsfallgruppen werden vom Framework bedient: Administrative Ressourcenbegrenzung sowie Ressourcenreservierung durch den Grid-Scheduler. Beim letzteren bedarf es der Anpassung des verwendeten Grid-Schedulers und es zeigt sich eine noch bestehende Unklarheit, woher Informationen über den Ressourcenbedarf kommen und wer sie dem Framework mitteilt. Der Schwerpunkt bei der Implementierung wurde folglich auf die erste Anwendungsfallgruppe gelegt.

Die Evaluierung des Frameworks hat gezeigt, dass die von den Komponenten benutzten Mechanismen funktionieren und verwendet werden können. Probleme ergeben sich lediglich bei iproute2, nicht alle gewünschten Szenarien konnten erfolgreich getestet werden. Die Weiterentwicklung von HTB und die Integration aktueller Kernel-Patches in gängige Distributionen lassen jedoch hoffen, dass auch dieses in naher Zukunft möglich sein wird.

8.2 Weiterentwicklung

Im Rahmen dieser Arbeit konnten aufgrund der hohen Anzahl der angeschnittenen Themengebiete nicht alle Aspekte des Frameworks abschließend behandelt werden. Sowohl bezüglich der Implementierung als auch auf der Entwurfsebene existieren mehrere Tätigkeitsfelder, welche bei einer Weiterverfolgung des Themas bedacht werden sollten.

8.2.44 Weiterentwicklungen auf Entwurfsebene

- Entwurf bzw. Weiterentwicklung eines Grid- oder Cluster-Schedulers um eine Anbindung des Frameworks zu ermöglichen.
- Entwicklung einer Sprache, um Restriktionen innerhalb der Datenbank zu ermöglichen. Diese werden benötigt, um Begrenzungen von Komponenten,

Parametern und Eingabeparametern zu realisieren (siehe Datenbankmodell in Kapitel 5.5). Erste, in dieser Arbeit nicht erwähnte, Überlegungen wurden bereits angestellt, zu sehen in Abbildung 26.

- Weiterentwicklung des Komponentenmodells, um komplexere Hierarchien zu ermöglichen. Beim vorliegenden Entwurf wird von einer einfachen Baumstruktur ausgegangen und Knoten können nur einen Elternknoten haben. Im Allgemeinen wird sich in dieser Hierarchie die Netzwerkstruktur widerspiegeln, doch ist es für Reservierungen sinnvoll, virtuelle bzw. temporäre Knotengruppen zu bilden, die für die Dauer einer Reservierung über einen gemeinsamen Elternknoten angesprochen werden können. Dadurch wäre es möglich, Rechner aus verschiedenen Netzen unter einem Knoten zusammenzufassen und über diesen Partitionierungen, welche dann automatisch vererbt werden, durchzuführen um so zu verhindern, jeden Knoten einzeln partitionieren zu müssen. Mit dieser Möglichkeit ergeben sich jedoch neue Probleme, da sich Partitionierungen unterschiedlicher Eltern widersprechen können. Hierfür bedarf es einer transparenten Lösung, bei der auf Elternebene signalisiert wird, ob die jeweilige Partitionierung in mindestens einem Kindknoten nicht durchführbar ist.
- Entwurf eines Sicherheitskonzeptes für die Server ↔ Client Schnittstelle. Als Protokoll wird XML RPC verwendet, welches von sich aus keine Möglichkeiten zur Authentifizierung mitbringt. Dieses ist jedoch notwendig, da ansonsten unerlaubte Zugriffe auf die Clients und somit Manipulationen der Partitionierungen möglich sind. Auch besteht die Gefahr, dass über die unter administrativen Rechten laufenden Clients Zugang auf den Rechner erlangt werden kann. Eine mögliche Lösung wäre das Verwenden von SSH, um darüber die XML RPC Pakete zu tunneln.
- Entwurf eines Sicherheitskonzeptes für die Anwender ↔ Server Schnittstelle. Der derzeitige Entwurf sieht kein Sicherheitskonzept vor, doch bedarf es einer Authentifizierung der an die Broker API gestellten Partitionierungsanfragen. Im einfachsten Fall kann von zwei Anwendern ausgegangen werden:
    1. Administratorkennung. Diese besitzt alle Rechte und kann Partitionierungsbegrenzungen setzen sowie Partitionierungen in Komponenten vornehmen, die nur Begrenzungsmöglichkeiten[89] anbieten.
    2. Grid-Kennung. Diese besitzt lediglich die Möglichkeit Reservierungen vorzunehmen.
- Entwurf eines Konzeptes zur Ressourcenauslastungsanzeige. Derzeit ist es nicht möglich, parallel zu gesetzten Partitionierungen den tatsächlichen Verbrauch bzw. die Auslastung der Ressource im Allgemeinen und der Partitionierung im Speziellen zu beobachten. Wahlweise können hierfür Mechanismen des Frameworks[90] oder ein Monitoringframework dritter Seite

[89] Identifiziert über das Feld `assuring` in der Tabelle Component, siehe Abbildung 7.

[90] Diese wurden teilweise bereits implementiert, so sind in der Komponente FSF dynamische Parameter vorhanden, die die Belegung von Mountpoints zurückgeben.

dafür verwendet werden. Zur Ressourcenanzeige wurden unter Verwendung des RRD Tools mehrere Frameworks entwickelt[91].

- Konkretisierung der Überlegungen inwieweit das Framework in den Jobplatzierungsmechanismus mit eingebunden werden kann, hierzu gehören auch die folgenden Punkte:
    - Analyse der Nutzbarkeit von Topologieinformationen. Es wäre denkbar, im Framework auf netzwerktopologische Informationen einzugehen und diese in der Datenbank zu speichern, um bei der Platzierung von Jobs mit hoher Netzwerklast diese in topologischer Nähe zu ihren Kommunikationspartnern zu platzieren.
    - Verknüpfung des Frameworks mit einem Monitoringframework, um aktuelle Ressourcenauslastung bei der Wahl möglicher Knoten zu berücksichtigen.
    - Analyse ob eine Verknüpfung des Frameworks mit *Network Weather Service* neue Einsatzmöglichkeiten ergeben würde.
- Verfolgung der Entwicklung des *Grid Resource Allocation Agreement Protocols* (Kapitel 3.3.7), um nach Veröffentlichung der Spezifikation den Sinn einer möglichen Implementierung des Protokolls in der Broker API zu analysieren.

8.2.45 Weiterentwicklungen auf Implementierungsebene

- Implementierung einer stabilen und umfassenden Fehlerbehandlung innerhalb der Komponenten.
- Implementierung eines performanteren Database Crawlers, um unnötige Lasterzeugung zu vermeiden. Hierzu sollte entweder ein nicht-blockierender XML RPC Client verwendet oder auf Thread-Programmierung umgestiegen werden.
- Implementierung eines Windows Client gemäß der Entwurfsarchitektur.
- Implementierung der Broker API, um Anbindung an Globus über RSL zu ermöglichen.
- Weiterentwicklung und Anpassung des *Scheduling Servers* von Peter Tröger (Kapitel 0), um CPU Partitionierung zu ermöglichen.
- Weitergehende Analyse von HTB, um die Ursachen des in Kapitel 7.3.42 , Szenario 3, aufgetretenen Verhaltens herauszufinden.

[91] RRD Monitoring Frameworks: http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/rrdworld/index.html

Anhang A: Details zum Datenbankentwurf

Primary Keys und *Foreign Keys* sind dem Datenbankdiagramm in Kapitel 5.5 zu entnehmen. In der Tabelle bedeutet „→“ soviel wie „Referenz auf“ und symbolisiert einen *Foreign Key* oder einen Teil eines *Foreign Keys*.

Tabelle `Component`		
`name`	`varchar(40)`	Name der Komponente
`description`	`varchar(200)`	Beschreibung der Komponente
`assuring`	`bool`	True: Komponente kann Reservierungen durchführen False: Komponente kann keine Reservierungen durchführen
Tabelle `ComponentRestriction`		
`component_name`	`varchar(40)`	→ `Component.name`
`computation`	`varchar(200)`	Berechnungsformel für technisch bedingte komponentenweite Beschränkungen, wie z.B. Anzahl der Reservierungen pro Benutzer oder Anzahl Reservierungen insgesamt.
Tabelle `Module`		
`component_name`	`varchar(40)`	→ `Component.name`
`name`	`varchar(40)`	Name des Moduls
`description`	`varchar(200)`	Beschreibung des Moduls
`optional`	`bool`	True: Modul ist optional False: Modul ist nicht optional, immer aktiv wenn Komponente verwendet wird
Tabelle `Input`		
`component_name`	`varchar(40)`	→ `Component.name` → `Module.component_name`
`name`	`varchar(40)`	Name der Eingabe
`module_name`	`varchar(40)`	→ `Module.name`, nur gesetzt wenn der Partitionierungsparameter einem Modul zugeordnet ist.
`Type`	`varchar(40)`	Typ der Eingabe, wie INPUT oder CHECKBOX
`description`	`varchar(200)`	Beschreibung
`unit`	`varchar(40)`	Einheit (optional)
`optional`	`bool`	True: Eingabe ist optional, für Partitionierung nicht notwendig False: Eingabe ist nicht optional, sondern muss bei jeder Partitionierung vorhanden sein
`preset`	`varchar(40)`	Defaultwert (optional)

computation	varchar(200)	Beruht der Eingabewert auf einem Parameter, so definiert dieses Feld die Berechnung, beispielsweise kann ein dynamischer Parameter eine per Semikolon separierte Liste zurück geben, dann würde dies hier in der frameworkeigenen Syntax stehen.
Tabelle InputRestriction		
component_name	varchar(40)	→ Input.component_name
input_name	varchar(40)	→ Input.name
type	varchar(40)	Art der Beschränkung, z.B. FLOOR für Minimalwert, CEILING für Maximalwert oder TYPE für Typdefinition
value	varchar(40)	Wert der Begrenzung (z.B. 0, 34, INT)
computation	varchar(200)	Handelt es sich um keine statische Begrenzung, sondern um eine dynamische, so bleibt das value Feld leer und stattdessen erfolgt in diesem Feld die Berechnungsformel für die Begrenzung.
Tabelle Paramter		
component_name	varchar(40)	→ Component.name
name	varchar(40)	Name des Parameters
type	varchar(40)	Typ des Parameters, wie INPUT oder CHECKBOX
description	varchar(200)	Beschreibung
unit	varchar(40)	Einheit (optional)
preset	varchar(40)	Defaultwert (optional)
dynamic	bool	True: Parameter ist dynamisch. Dies bedeutet, dass er kein Eingabeparameter ist, sondern zur Laufzeit Systeminformationen liefert. Sowohl preset als auch type sind irrelevant. False: Es handelt sich um einen statischen Parameter zur Administration der Komponente.
Tabelle ParamterRestriction		
component_name	varchar(40)	→ parameter.component_name
parameter_name	varchar(40)	→ parameter.name
type	varchar(40)	Art der Beschränkung, z.B. FLOOR für Minimalwert, CEILING für Maximalwert oder TYPE für Typdefinition
value	varchar(40)	Wert der Begrenzung (z.B. 0, 34, INT)
computation	varchar(200)	Handelt es sich um keine statische Begrenzung, sondern um eine dynamische, so bleibt das value Feld leer und stattdessen erfolgt in diesem Feld die Berechnungsformel für die Begrenzung.

Tabelle Node		
id	int unsigned	Knoten ID
parent_id	int unsigned	→ Node.id, nur gesetzt wenn Knoten einen Elternknoten hat
name	varchar(40)	Name des Knotens
ip	varchar(40)	IP Adresse des Knotens, wenn Blattknoten, dann muss sie eine echte IP sein, ansonsten kann sie einen Bereich definieren.
Description	varchar(200)	Beschreibung
leaf	bool	True: Der Knoten ist ein Blattknoten und steht für einen adressierbaren Rechner im Netz. False: Elternknoten für andere Knoten. Dieser Wert muss nur gesetzt werden, wenn es sich um einen Blattknoten handelt.
Online	bool	True: Der Knoten ist bei der letztmaligen Überprüfung online gewesen. False: Knoten ist offline.
New	bool	True: Der Knoten ist neu in die Datenbank eingetragen worden, eine erstmalige Überprüfung ob er online ist, hat noch nicht statt gefunden. False: /
removed	bool	True: Knoten soll entfernt werden. False: /
lastcheck	bigint unsigned	Zeitstempel der letztmaligen Überprüfung ob der Knoten online ist. Sekunden seit 01.01.1970, 00:00 Uhr.
Tabelle Node2Component		
node_id	int unsigned	→ Node.id
component_name	varchar(40)	→ Component.name
removed	bool	True: Komponente soll von dem Knoten entfernt werden. False: /
Tabelle Node2Module		
node_id	int unsigned	→ Node2Component.node_id
component_name	varchar(40)	→ Node2Component.component_name → Module.component_name
module_name	varchar(40)	→ Module.name
Tabelle Node2Parameter		
node_id	int unsigned	→ Node2Component.node_id
component_name	varchar(40)	→ Node2Component.component_name → Parameter.component_name
parameter_name	varchar(40)	→ Parameter. Name
value	varchar(200)	Wert des Parameters
sync	bool	True: Wert wurde an die Komponente übermittelt und ist synchron. False: Wert waretet auf Synchronisierung.
Tabelle Partitioning		
node_id	int unsigned	→ Node2Component.node_id
component_name	varchar(40)	→ Node2Component.component_name
user	varchar(40)	Name des Benutzers, für den die Reservierung durchgeführt werden soll.
Partitioning_id	smallint unsigned	Reservierungsnummer für den angegebenen Benutzer auf dem gegebenen Knoten und bei der gewählten Komponente.
Sync	bool	True: Reservierung wurde durchgeführt und ist aktiv.

Removed	bool	False: Reservierung wurde nicht durchgeführt oder ist fehlgeschlagen. True: Reservierung soll entfernt werden.
new	bool	False: / True: Reservierung ist neu und es wurde noch kein Versuch unternommen sie zu setzen.
failed	bool	False: / True: Reservierung ist fehlgeschlagen.
failed_comment	varchar(200)	False: / Schlägt eine Reservierung fehl, wird in diesem Feld der übertragene Fehlergrund gespeichert.
Tabelle ReservationRestriction		
node_id	int unsigned	→ Node2Component.node_id
component_name	varchar(40)	→ Node2Component.component_name
computation	varchar(200)	Berechnungsformel für komponentenweite Reservierungsbeschränkung.
Tabelle Node2Input		
node_id	int unsigned	→ Reservation.node_id
component_name	varchar(40)	→ Reservation.component_name → Input.component_name
user	varchar(40)	→ Reservation.user
partitioning_id	varchar(200)	→ Reservation. Partitioning_id
input_name	smallint unsigned	→ Input.Name
value	varchar(200)	Wert

Tabelle 21: Datenbankdetails

Anhang B: Bildschirmfotos der Weboberfläche

An dieser Stelle folgen zur Veranschaulichung der Funktionalität der Webserverskripte kommentierte Bildschirmfotos.

Abbildung 25: Bildschirmfoto – Startseite

In Abbildung 25 ist die Startseite des Frameworks zu sehen. Neben den Verweisen auf die Knotenliste sowie zur Detailansicht der jeweiligen Komponenten, wird angezeigt, ob der Database Crawler, hier „Database scanner“ genannt, läuft und wenn ja, welche PID er hat.

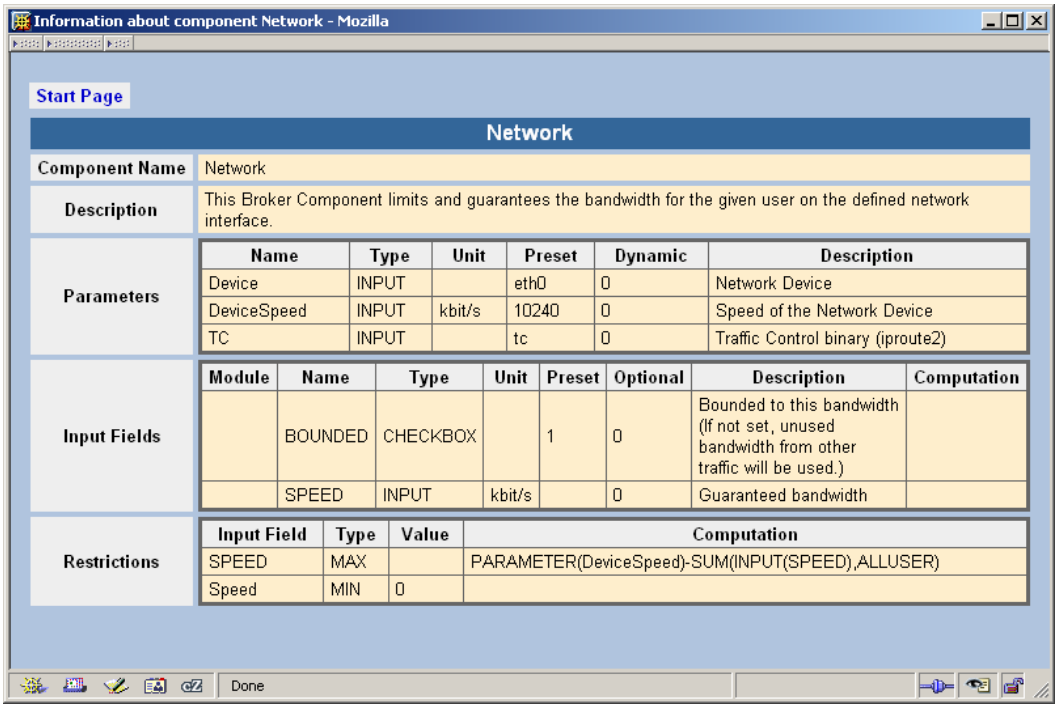


Abbildung 26: Bildschirmfoto – Komponentendefinition

Abbildung 26 zeigt eine Komponentendefinition und somit alle Informationen die zu der Komponente in der Datenbank vorliegen. Auch wenn in der Implementierung noch keine Berücksichtigung der Begrenzungen erfolgt (siehe Kapitel 6), kann man hier, bei der Restriction zu SPEED, erste Überlegungen zu einer möglichen Syntax erkennen.

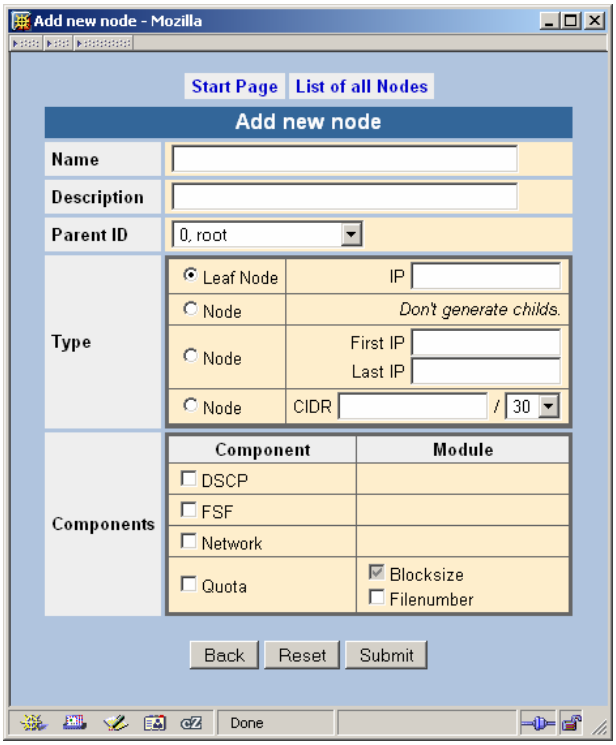


Abbildung 27: Bildschirmfoto – Neuen Knoten anlegen

Abbildung 27 zeigt die Eingabemaske zum Anlegen eines neuen Knotens. Knoten können an beliebige Knoten (die nicht vom Typ leaf sind) angehangen werden und können ebenfalls Kindknoten besitzen. Neben den Komponenten können die Module, sofern sie optional sind, ausgewählt werden.

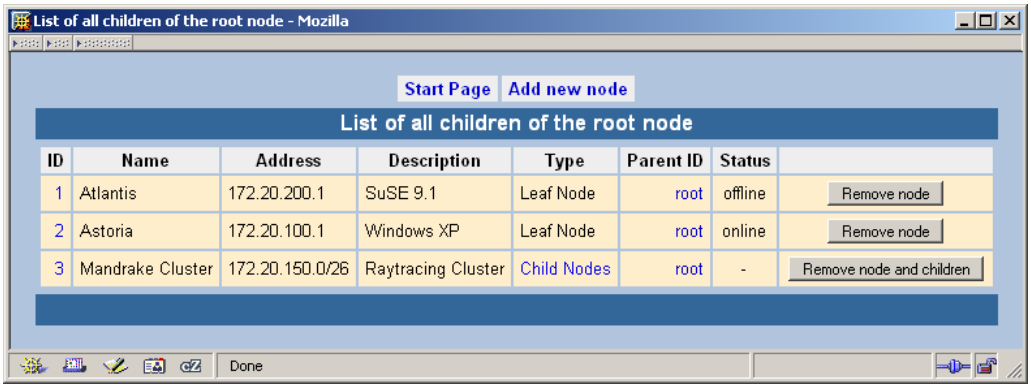


Abbildung 28: Bildschirmfoto – Knotenliste des Wurzelknotens

Abbildung 28 zeigt eine Knotenliste aller Knoten ohne definierten Elternknoten[92]. Zu sehen sind drei Blattknoten sowie ein Knoten mit weiteren Kindern. Von den Blattknoten ist einer mit dem Status online in der Datenbank verzeichnet.

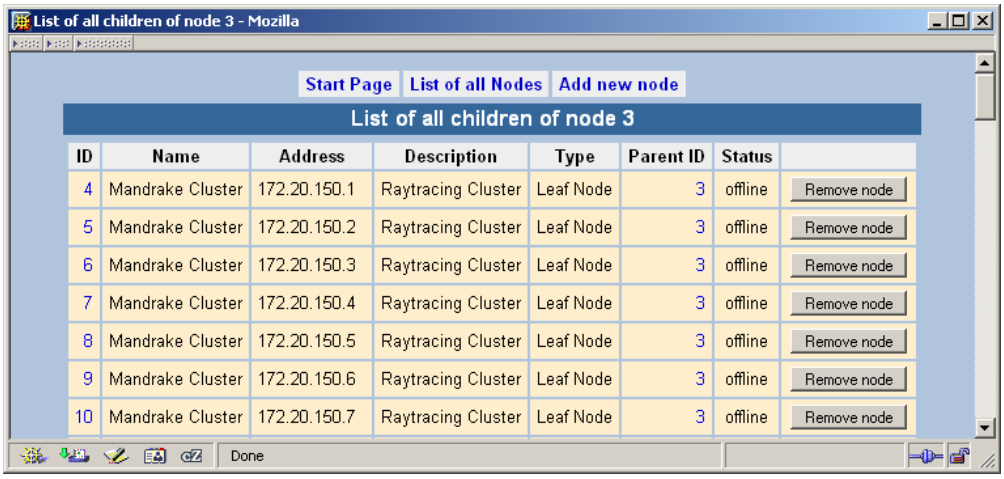


Abbildung 29: Bildschirmfoto – Knotenliste

Abbildung 29 zeigt einen Ausschnitt aus der Liste aller Kindknoten des dritten Knotens in der vorherigen Abbildung. Alle Kindknoten wurden beim Anlegen des Elternknotens automatisch erzeugt, die IP-Adressen wurden dabei aus der CIDR[93]-Angabe generiert. Hierbei haben sie die gleiche Komponentenkonfiguration wie die des Elternknotens erhalten.

[92] Diese Knoten werden als Kinder des Wurzelknotens dargestellt, der jedoch nicht in der Datenbank existiert, sondern von den Webskripten als Wurzelknoten für alle elternlose Knoten eingeführt wird.

[93] *Classless Inter-Domain Routing* beschreibt ein Verfahren zur Nutzung des 32 Bit IP Adressraums, bei dem die Unterteilung der IP Adresse in Netzwerk- und Hostteil mittels einer Netzwerkmaske erfolgt.

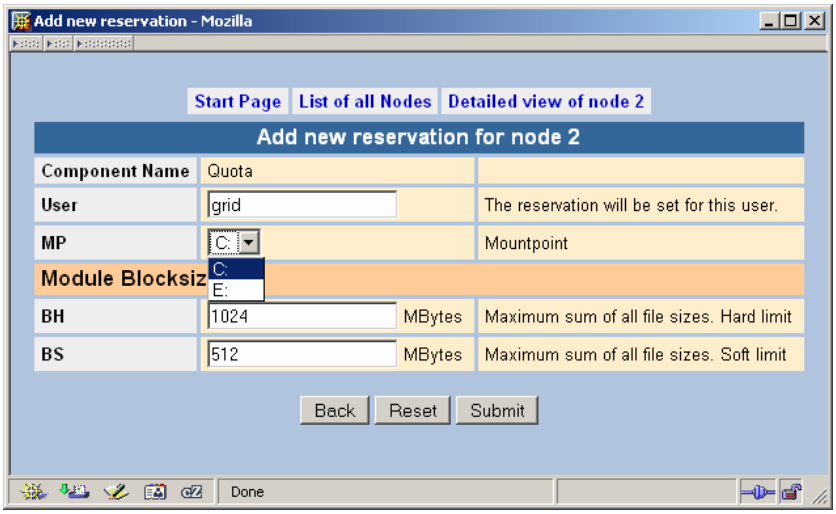


Abbildung 30: Bildschirmfoto – Reservierung anlegen

In Abbildung 30 ist das Anlegen einer Reservierung der Komponente Quota zu erkennen. Sofern es sich, wie in diesem Fall gegeben, um einen Blattknoten handelt und er online ist, wird versucht, die verfügbaren, mit Qutoas versehenen, Mountpoints automatisch zu ermitteln und als Auswahlliste zur Verfügung zu stellen.

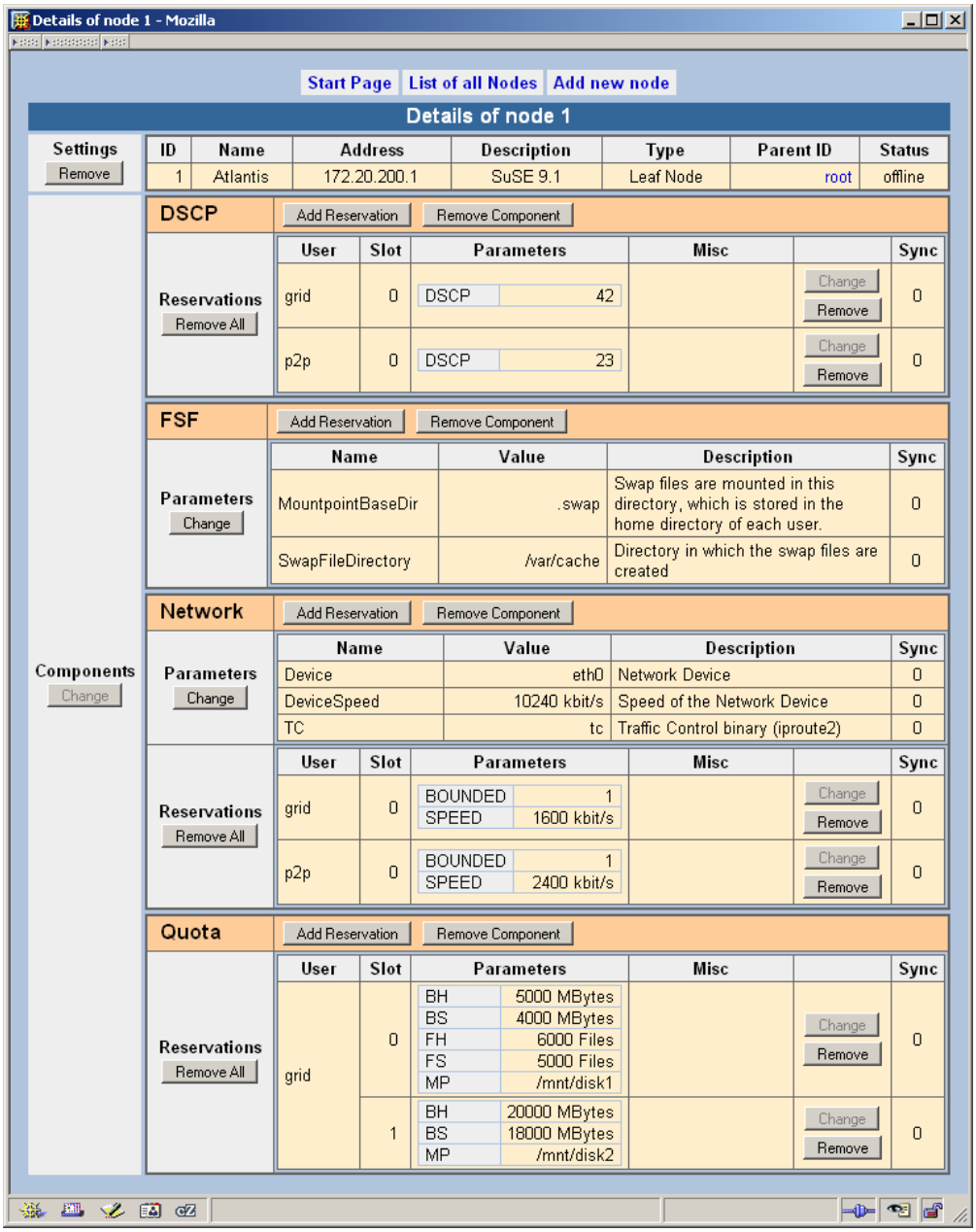


Abbildung 31: Bildschirmfoto – Detailansicht

Abbildung 31 zeigt die Detailansicht eines Knotens. Zu erkennen sind alle Komponenten mit allen Parametern und allen Reservierungen. An der Spalte Sync ist zu erkennen, ob die jeweiligen Werte mit dem Knoten synchronisiert wurden oder nicht. Nicht implementiert wurde die Funktionalität zum Ändern einer Reservierung sowie zum Ändern der Komponentenkonfiguration eines Knotens, die entsprechenden Buttons sind inaktiv.

Anhang C: Quellcode Linux Broker Client

```
#!/usr/bin/perl



# (c) by Matthias Lendholt                                                  #
# Hasso Plattner Institut                                                   #
```

```
# Universitaet Potsdam                                                    #
#                                                                         #
# Version of this file: 0.1.0                                             #
# Last modified: 2004-10-27                                               #
#                                                                         #
# This file is part of the master thesis of Matthias Lendholt.            #
#                                                                         #

# include public perl libs
use strict;
use Frontier::Daemon;
use Frontier::RPC2;
my $XMLRPCcoder = Frontier::RPC2->new;

# include the Broker module
use Broker::Broker;
my %broker; # hash for all used brokers (loaded at runtime!)

# directory in which the Broker Modules store information about reservations
# (needed for reinstall reservations after restart)
my $workingdir="/tmp";

# restore reservations via creating a new instance of each component
opendir(DIR,"Broker") or print "Can not open default directory for Broker components.\n";
while (defined(my $file=readdir(DIR))){
  if ($file=~/^(.*)\.pm$/ && $1 ne 'Broker'){
    my $type="Broker::".$1;
    next unless (eval "require $type");
    print "Restoring reservations of Broker component $1:\n";
    $broker{$type}=$type->new();
  }
}
closedir(DIR);

my $methods={
  'addReservation' =>\&addReservation,
  'changeReservation'=>\&changeReservation,
  'removeReservation'=>\&removeReservation,
  'removeReservations' =>\&removeReservations,
  'removeAllReservations'=>\&removeAllReservations,
  'setParameter'   =>\&setParameter,
  'getParameter'   =>\&getParameter,
  'getDynamicParameter'=>\&getDynamicParameter,
  'ping'     =>\&ping
  };
print "Starting server…. Kill me with Ctrl+C ….\n";
Frontier::Daemon->new(LocalPort => 80, methods => $methods)or die "Couldn't start HTTP server:
$!";
exit;

sub addReservation{return $XMLRPCcoder->string(_metaMethod("addReservation",@_));}
sub changeReservation{ return $XMLRPCcoder->string(_metaMethod("changeReservation",@_));}
sub removeReservation{ return $XMLRPCcoder->string(_metaMethod("removeReservation",@_));}
sub removeReservations{
return $XMLRPCcoder->string(_metaMethod("removeReservations",@_));}
sub removeAllReservations{
  return $XMLRPCcoder->string(_metaMethod("removeAllReservations",@_));}
sub setParameter{return $XMLRPCcoder->string(_metaMethod2("setParameter",@_));}
sub getParameter{return $XMLRPCcoder->string(_metaMethod2("getParameter",@_));}
sub getDynamicParameter{
return $XMLRPCcoder->string(_metaMethod2("getDynamicParameter",@_));
}
sub ping{return $XMLRPCcoder->string(1);}

sub _metaMethod{
  (my $method, my $type, my $user, my $reservation, my $parameters)=@_;
  print "NEW MM:: $method, $type, $user, $reservation:\n";
  return Broker::Broker::_user2uid($user) if ((defined
$user)&&(Broker::Broker::_user2uid($user)<0));
  return "1; no module type given" if (!defined $type || $type eq "" || $type eq "Broker");
  $type="Broker::".$type;
  unless (defined $broker{$type}){
    return "1; could not load module;".$@ unless (eval "require $type");
    $broker{$type}=$type->new();
  }
  if (defined $parameters){
    return $broker{$type}->$method($user,$reservation,$parameters);
  }elsif (defined $reservation){
    return $broker{$type}->$method($user,$reservation);
  }elsif (defined $user){
    return $broker{$type}->$method($user);
  }else{
```

```
    return $broker{$type}->$method();
  }
}

sub _metaMethod2{
  my $method=shift;
  my $type=shift;
  print “NEW MM2:: $method, $type:\n”;
  return “1; no module type given” if (!defined $type || $type eq “” || $type eq “Broker”);
  $type=”Broker::”.$type;
  unless (defined $broker{$type}){
    return “1; could not load module;”.$@ unless (eval “require $type”);
    $broker{$type}=$type->new();
  }
  return $broker{$type}->$method(@_);
}
```

Anhang D: Inhalt des beiliegenden Datenträgers

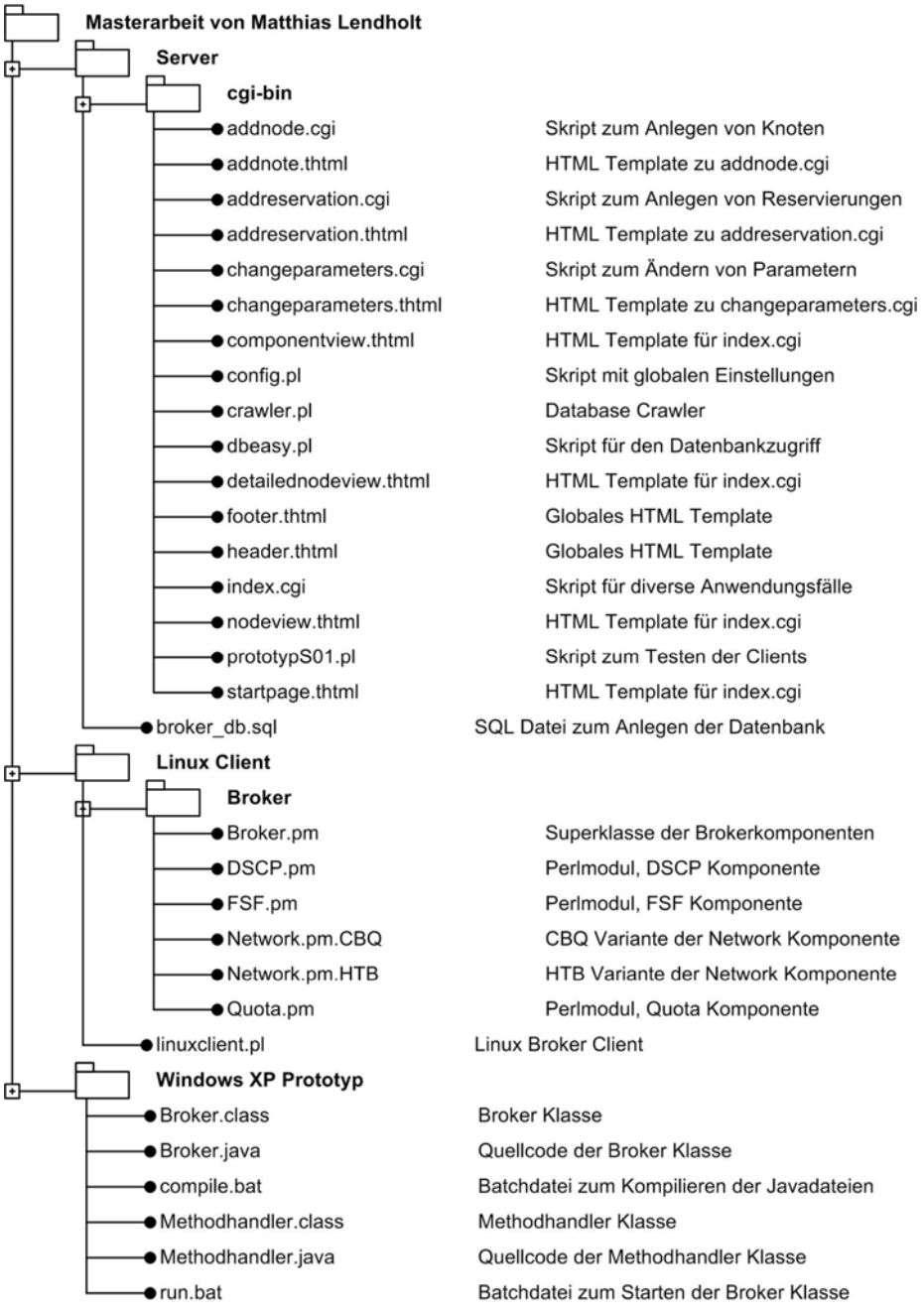


Abbildung 32: Inhalt des beiliegenden Datenträgers

Quellenverzeichnis

[01] *Brooke, Fellows, MacLaren: "Resource Brokering: The EUROGRID/GRID Approach", University of Manchester, 2004, Manchester (UK)*

[02] *Buyya, Baker, Laforenza: "Grids and Grid Technologies for Wide-Area Distributed Computing", International Journal of Software: Practice and Experience (SPE), vol. 32, no. 15, pp. 1437-1466, Wiley Press, December 2002, USA*

[03] *Carpenter, Nichols: "Differentiated Services", IETF, September 2002, USA*

[04] *Christiansen, Torkington: "Perl Kochbuch", O'Reilly, March 2000, Köln (Germany), ISBN 3897211408*

[05] *Czajkowski et al.: "Web Services Agreement Specification (WS-Agreement)", Global Grid Forum, Mai 2004, USA*

[06] *Czajkowski, Dan, Rofrano, Tuecke, Xu, "WS-Agreement: Agreement-based Grid Service Management (OGSI Agreement)" Global Grid Forum, June 2003, USA*

[07] *Czajkowski, Foster, Kesselman, Sander, Tuecke: "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems", 8th Workshop on Job Scheduling Strategies for Parallel Processing, July 2002, Edinburgh (UK)*

[08] *Condor Team: "Condor Version 6.7.2 Manual", University of Wisconsin-Madison, October 2004, Wisconsin (USA)*

[09] *Drees: "Traffic Shaping mit Linux", in Linux Enterprise 7/8.04, Software & Support Verlag GmbH, July 2004, Frankfurt a.M. (Deutschland)*

[10] *Foster: "What is the Grid? A Three Point Checklist", Argonne National Laboratory & University of Chicago, July 2002, Chicago (USA)*

[11] *Foster, Fidler, Roy, Sander, Winkler: "End-to-End Quality of Service for High-End Applications", Elsevier Computer Communications Journal, vol. 27, no. 14, pp. 1375-1388, 2004, USA*

[12] *Foster, Kesselman: "The Grid", Morgan Kaufmann Publishers, November 2003, USA, ISBN 1558609334*

[13] *Foster, Kesselman, Lee, Lindell, Nahrstedt, Roy: "A distributed resource management architecture that supports advance reservations and co-allocation", International Workshop on Quality of Service, 1999, USA*

[14] *Foster, Kesselman, Nick, Tuecke: "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Globus Project, June 2002, USA*

[15] *Foster, Kesselman, Tuecke: "The anatomy of the grid: Enabling scalable virtual organizations", in International Journal of Supercomputer Applications, 2001, USA*

[16] *Foster, Roy, Sander: "A Guide to GARA", Argonne National Laboratory, May 2000, Illinois (USA)*

[17] *Foster, Roy, Sander: "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation", in Proceedings of the 8th International Workshop on Quality of Service (IWQOS), pp. 181-188, June 2000, Pittsburgh (USA)*

[18] *Foster, Roy, Sander: "Administrators Guide to GARA", Argonne National Laboratory, March 2000, Illinois (USA)*

[19] *Foster, Roy, Sander: "Programmers Guide to GARA", Argonne National Laboratory, March 2000, Illinois (USA)*

[20] *GGF DRMAA WG: "Distributed Resource Management Application API Specification 1.0", Global Grid Forum, June 2004, USA*

[21] *Globus Alliance, GRAM Documentation, GRAM Architecture: http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/developer/architecture.html*

[22] Hubert: “Linux Advanced Routing & Traffic Control HOWTO”, October 2003, http://lartc.org/

[23] Mauro, Schmidt: “Essential SNMP”, O’Reilly, July 2001, USA, ISBN 0596000200

[24] Microsoft: “Best Practices for Managing Applications with Process Control”, Microsoft Windows 2000 Datacenter Server Documentation, 2001, USA

[25] Microsoft, Windows XP Home Edition Product Documentation: Proccon: http://www.microsoft.com/windowsxp/home/using/productdoc/en/default.asp?url=/WINDOWSXP/home/using/productdoc/en/processcontrol.asp

[26] Microsoft Developer Network, Platform SDK, Windows Management Instrumentation: Disk Quota Provider: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/disk_quota_provider.asp

[27] Microsoft Developer Network, Win32 and COM Development: Job Objects: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/job_objects.asp

[28] Microsoft Developer Network, Win32 and COM Development: JOBOBJECT_BASIC_LIMIT_INFORMATION: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/jobobject_basic_limit_information_str.asp

[29] Morgan: “The Linux-PAM System Administrators Guide”, June 2002, http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html

[30] Oestereich: “Die UML 2.0 Kurzreferenz für die Praxis“, Oldenbourg Verlag, March 2004, Deutschland, ISBN 3486276042

[31] Polze, Tröger: “Grid-Computing - Technische Berichte“, Universität Potsdam, March 2004, Potsdam (Germany)

[32] Roy: “End-to-End Quality Of Service For High-End Applications”, University of Chicago, August 2001, Chicago (USA)

[33] Roy, Sander: “GARA: A Uniform Quality Of Service Architecture”, in Jarek Nabrzyski: “Grid Resource Management”, Kluwer Academic Publishers, November 2003, USA, ISBN 1402075758

[34] Sander: “Design and Evalutaion of a Bandwidth Broker that Provides Network Quality of Service for Grid Applications”, RWTH Aachen, November 2002, Aachen (Germany)

[35] Sander, Foster, Roy, Winkler: “A Differentiated Services Implementation for High-Performance TCP Flows”, Elsevier Computer Networks, no. 34 pp. 915-929, 2000, USA

[36] Silva: “The Master Managed Job Factory Service (MMJFS), A custom GRAM client for the Globus Toolkit 3.x”, IBM, Mai 2004, USA

[37] Tordsson: “Resource Brokering for Grid Environments”, Umeå University, June 2004, Umeå (Sweden)

[38] Tröger: “CPU partitioning on Windows NT and SUN Solaris”, Humboldt Universität Berlin, June 2002, Berlin (Germany)

[39] Tuecke, et. al, "Open Grid Services Infrastructure (OGSI) Version 1.0", Global Grid Forum OGSI Working Group, June 2003, USA

[40] Wroclawski: “Integrated Services”, IETF, September 2000, USA

[41] Zhang, Braden: “Resource Reservation Protocol“, IETF, March 2001, USA

ISBN 3-937786-72-4
ISSN 1613-5652