Höchstleistungsrechenzentrum HLRS
Universität Stuttgart
Prof. Dr.-Ing. M. Resch
Nobelstraße 19, 70569 Stuttgart

MANAGEMENT VON VERTEILTEN INGENIEURWISSENSCHAFTLICHEN ANWENDUNGEN IN HETEROGENEN GRID UMGEBUNGEN

von der Fakultät Maschinenbau der Universität Stuttgart
zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Abhandlung

vorgelegt von

Dipl.-Ing. Peggy Lindner
aus Zschopau/Deutschland

Hauptberichter: Prof. Dr.-Ing. M. Resch
Mitberichter: Prof. Dr.-Ing. Prof. E. h. Dr.-Ing. E. h. Dr. h.c. mult. E. Westkämper
Tag der Einreichung: 28.Juni 2006
Tag der mündlichen Prüfung: 17. Juli 2007

Alle Rechte vorbehalten.

©2007 by Peggy Lindner

Höchstleistungsrechenzentrum Stuttgart
Nobelstraße 19
D- 70569 Stuttgart

Zusammenfassung

Grid Technologien stellen einen Lösungsansatz für die Verteilung von Anwendungen über mehrere Rechner dar, um Simulationen von wissenschaftlichen Problemen durchführen zu können, die hohe Anforderungen an Rechenressourcen haben. Während diese Art von Anwendungen in den letzten Jahren meistens noch zu Demonstrationszwecken eingesetzt wurde, ist die Grid Technologie heute mehr und mehr ein Werkzeug im täglichen Einsatz.

Dabei ist die Heterogenität der vorhandenen Grid Software Umgebungen das größte Problem mit dem Benutzer umgehen müssen, wenn sie parallele, verteilte Anwendungen effizient im Grid ausführen wollen. Im Rahmen dieser Arbeit wird das Konzept und die Implementierung eines Grid Configuration Managers (GCM) vorgestellt, der die Komplexität der Grid Umgebungen und die damit verbundenen Probleme vor dem Benutzer verbergen soll. Das wichtigste Ziel des GCM ist die Vereinfachung des Managements von Grid Umgebungen für Endanwender und Entwickler. Dafür wurden die für die Ausführung von verteilten, parallelen Anwendungen notwendigen Schritte abstrahiert. Des Weiteren wurde ein Konzept für die Integration verschiedener Grid Software Lösungen entwickelt und implementiert. Zurzeit unterstützt der GCM Globus, UNICORE und ssh basierende Umgebungen.

Der GCM soll Benutzer hauptsächlich während drei Phasen der Ausführung von Anwendungen helfen: bei der Definition einer Grid Konfiguration, beim Starten und bei der Überwachung einer Grid Anwendung. Der GCM bietet außerdem noch eine spezielle Unterstützung für verteilte Anwendungen, die auf Basis der Kommunikationsbibliothek PACX-MPI entwickelt wurden. Dafür werden die benötigten Konfigurationsdateien automatisch erstellt und auf den beteiligten Rechnern konsistent gehalten.

In den Grid Configuration Manager wurde ein auf Leistungsvorhersage basierender Mechanismus zur Auswahl von Rechenressourcen integriert. Ausgehend von einer durch den Benutzer spezifizierten Vorauswahl an Rechnern kann der GCM anhand einer automatischen Abschätzung von Leistungsdaten einer Anwendung vorhersagen, was die effizienteste Umgebung für die Ausführung der Anwendung ist. Für die Leistungsvorhersage wird das Programm Dimemas benutzt. Dimemas kann eine Vorhersage für das Laufzeitverhalten einer Anwendung anhand von Tracing-Daten und Parameter zur Beschreibung der Hardware treffen.

Der Grid Configuration Manager wurde in verschiedenen Szenarien getestet und eingesetzt. Dabei wurde aufgezeigt, dass die Handhabung von verteilten Anwendungen durch die Verwendung des GCM signifikant vereinfacht und die Festlegung der Ausführungsumgebung erleichtert wird.

Abstract

Deploying Grid technologies by distributing an application over several machines is widely emerging for scientific simulations, which have large requirements for computational resources. While in the last couple of years distributed simulations were often executed on a pure demonstration level, currently ongoing work is focusing on taking the steps necessary to use the technology in an every day production environment.

One of the major problems end-users are facing is the diversity in the available Grid-software environments. This work presents the concept and implementation of a Grid Configuration Manager (GCM), a tool developed to hide some of the complexity of Grid environments from the end-user. The central objective of GCM is to ease the handling of computational Grid environments for end-users and application developers, by abstracting the steps necessary for submitting computational jobs and implementing them for various Grid software environments. Up to now, GCM supports Globus, UNICORE and ssh based environments.

The GCM assists users during three phases of the execution of an application: defintion of a Grid configuration, job submission and monitoring. Furthermore, GCM supports the generation and management of distributed multi-site jobs on the basis of the communication library PACX-MPI, by generating the according configuration files and keeping them consistent on all machines participating in the current simulation.

The Grid Configuration Manager integrates a performance based resource brokering mechanism. The basic idea is, that given a pool of machines, GCM can decide based on the prediction of the application performance on various machines or combination of machines, which one to use. The performance prediction step is based on Dimemas. Dimemas can estimate the performance of an application based on an initial tracefile for varying network and machine parameters like network latency or bandwidth.

The Grid Configuration Manager was tested and deployed in various scenarios. It has been shown, that the handling of scientific, computational jobs in heterogeneous Grid environments was significant easier and the determination of the execution environment was facilitated for users by utilization of the GCM.

Für Nora

Danksagung

In erster Linie gebührt mein Dank dem Höchstleistungsrechenzentrum Stuttgart und Professor Michael Resch, der mit einer einmaligen Arbeitsumgebung und kompetenten Mitarbeitern diese Arbeit erst ermöglichte. Bedanken möchte ich mich auch bei Professor Westkämper für die freundliche Übernahme der Zweitbetreuung meiner Arbeit.

Ein großes Dankeschön, auch für die Geduld und Unterstützung im Anwendungsbereich, sowie die kreativen Diskussionen und fachlichen Hinweise geht an Rainer Keller. Sein Einsatz hat wesentlich zum Gelingen dieser Arbeit beigetragen.

Besonderer Dank gilt auch den Systemadministratoren des Höchstleistungsrechenzentrums. Thomas Beisel, Bernd Krischok und die Systembetreuer von NEC, Holger Berger und Danny Sternkopf, waren jederzeit erreichbar. Mein Dank gilt auch Judit Gimenez vom Barcelona Supercomputing Center für die freundliche Anwendungsunterstützung.

Die vorliegende Dissertation wäre nicht zustande gekommen, wenn ich nicht auch breite Unterstützung und Hilfe aus meiner Familie gehabt hätte. Ich möchte mich daher bei meinen Eltern und Schwiegereltern bedanken, die mich in jeder Lebenslage unterstützt und ermutigt haben. Zum Schluss möchte ich mich bei meinem Mann für die Geduld, Aufmunterung und nicht zuletzt die vielen fachlichen Diskussionen bedanken, die einen Grossteil zum Gelingen dieser Arbeit beigetragen haben.

Inhaltsverzeichnis

Abbildungsverzeichnis **xv**

Tabellenverzeichnis **xvii**

Verzeichnis der verwendeten Abkürzungen **xix**

1. Motivation und Einleitung **1**
1.1. Definition des Begriffes "'Computational Grid"' 2
1.1.1. Grid Architektur 3
1.1.2. Anwendungsklassen von Computational Grids 6
1.2. Probleme beim Management verteilter Anwendungen in heterogenen Grid Umgebungen 8
1.3. Ziele dieser Arbeit 11

2. Stand der Technik - Relevante Projekte für Grid Middleware Lösungen **15**
2.1. Vergleichskriterien 16
2.2. SSH 18
2.3. Globus Toolkit (Version 2.0) 19
2.4. UNICORE 23
2.5. TME/ITBL 27
2.6. Avaki/Legion 31
2.7. Symphony 33
2.8. Zusammenfassung und Bewertung der Projekte 35
2.9. Standardisierungsaktivitäten 37
2.9.1. Global Grid Forum (GGF) 37
2.9.2. Organisation for the Advancement of Structured Information Standards (OASIS) 40

3. Das Konzept und die Implementierung eines Grid Configuration Managers **43**
3.1. Ziele des Grid Configuration Managers 44
3.2. Die Kommunikationsbibliothek PACX-MPI 45
3.2.1. Konzept von PACX-MPI 45
3.2.2. Verwendung und Konfiguration von PACX-MPI 46

3.3. Das Leistungsvorhersagewerkzeug Dimemas 48
3.3.1. Konzept von Dimemas 48
3.3.2. Verwendung von Dimemas 54
3.4. Konzept und Implementierung des Grid Configuration Managers 55
3.4.1. Definition einer Grid Konfiguration 57
3.4.2. Unterstützung verschiedener Grid Zugangsmechanismen 59
3.4.3. Leistungsvorhersage mit dem Grid Configuration Manager 61
3.4.3.1. Konfiguration und Starten von Dimemas mit dem GCM 62
3.4.3.2. Automatische Leistungsvorhersage mit Hilfe von Heuristiken 63
3.4.3.3. Der GCM als Entscheidungshilfe in der Software-Entwicklungsphase . . 69
3.4.3.4. Speicherung von Daten in einem Metadaten Repository 70
3.4.4. Starten und Überwachen der Anwendung 72
3.4.5. Komponente für den Dateitransfer 72
3.4.6. Ausführung von verteilten parallelen Anwendungen auf Basis von PACX-MPI 74
3.4.6.1. Erstellen der PACX-Konfigurationsdateien 75
3.4.6.2. Starten von PACX-MPI-Anwendungen 75
3.5. Das UNICORE Metacomputing-Plugin 76
3.5.1. Konzept und Implementierung 76
3.5.2. Verwendung 77

4. Untersuchung und Bewertung von Anwendungsszenarien des Grid Configuration Managers 81
4.1. Einsatz des GCM im DAMIEN Projekt 81
4.2. Evaluierung des GCM unter Verwendung von parallelen Gleichungslösern für ingenieurwissenschaftliche Anwendungen 84
4.2.1. Szenario 1 - Auswahl der Konfiguration mit minimalem Kommunikationsaufwand 86
4.2.2. Szenario 2 - Variation der Kommunikationsparameter zwischen den Rechnern 89
4.2.3. Szenario 3 - Testen verschiedener Implementierungen einer Anwendung 96
4.2.4. Fazit 100

5. Zusammenfassung und Ausblick 103

Literaturverzeichnis 105

A. Verwendung des Grid Configuration Managers 113
A.1. Installation und Starten 113
A.2. Erstellen und Speichern einer Grid Konfiguration 114
A.3. Leistungsvorhersage mit Dimemas 116
A.4. Automatische Leistungsvorhersage 117

A.5. Starten und Überwachen der Anwendung, Dateitransfer 117
A.6. Spezifische Unterstützung von PACX-MPI Anwendungen 118
A.7. Einstellungen und Hilfe 119

Glossar **121**

Abbildungsverzeichnis

1.1. Grid Architektur im Schichtenmodell 4

2.1. Schematische Darstellung der Arbeitsweise von ssh 18
2.2. Globus Sanduhr Modell 20
2.3. Beispielarchitektur eines Globus Grids 22
2.4. Architektur von UNICORE 24
2.5. Schichtenmodell des ITBL Systems 28
2.6. Aufbau von TME 29
2.7. Software Schichten in Avaki 32
2.8. Überblick über das Symphony System 34

3.1. Konzept von PACX-MPI 46
3.2. Beispiel für PACX-MPI Konfigurationsfiles 47
3.3. Dimemas Modell zur Beschreibung der Zielumgebung/architektur [6] . 49
3.4. Dimemas Modell für die Modellierung der Punkt-zu-Punkt Kommunikation [5] 52
3.5. Dimemas Modell zur Beschreibung der Anwendung [7] 54
3.6. Ablaufdiagramm des Grid Gonfiguration Managers 55
3.7. Ausschnitt aus dem Klassendiagramm des Grid Configuration Managers (Oberste Abstraktionsschicht) 57
3.8. Die Klassen der *Hostobjekte* 60
3.9. Methoden der Klasse Dimemas 63
3.10. Prinzipielle Vorgehensweise des GCM bei der automatischen Leistungsvorhersage 64
3.11. Kommunikation bei unterschiedlicher Verteilung von Prozessen bei Verwendung von Blockstrukturen 66
3.12. Abbildung der Prozesse auf ein karthesisches Gitter und deren Einfluss auf die Kommunikation 67
3.13. Beispiel für die Verteilung von Prozessen für den Fall „Blockweise Verteilung“ 68
3.14. Beispiel für die Verteilung von Prozessen für den Fall „Einfache Verteilung“ 69
3.15. Datenstruktur für das Abspeichern von Metadaten im GCM 70
3.16. Beispiel für die Ausgabe von Metadaten 71

3.17. Klassendiagramm der Dateitransferkomponente NIFTI 73
3.18. Aufgaben, die GCM während der Ausführung von PACX-MPI-Jobs unterstützt 74
3.19. Ausführung des Metacomputing Plugins in UNICORE 77
3.20. Das UNICORE Metacomputing Plugin 78

4.1. Der Grid Configuration Manager im DAMIEN Projekt 82

A.1. Der Grid Configuration Manager nach dem Starten 114
A.2. Dialog für die Konfiguration eines Hostobjektes 114
A.3. Dialog für die Konfiguration von Linkobjekten 115
A.4. Das Grid Terminal 118
A.5. Dateitransferkomponente von GCM 118

Tabellenverzeichnis

2.1. Eigenschaften von SSHv2 19
2.2. Eigenschaften von Globus 23
2.3. Eigenschaften von UNICORE 26
2.4. Eigenschaften von TME/ITBL 30
2.5. Eigenschaften von Avaki 33
2.6. Vergleich von ssh, Globus, UNICORE, TME, und Avaki 36
2.7. Arbeitsgruppen im Global Grid Forum (Stand Mai 2006) 40

4.1. Übersicht der verwendeten Tracedateien 85
4.2. Verwendete Parameter für die Dimemas Simulation 86
4.3. Eingangsdaten für die Vorhersagen in Szenario 1, Punkt A 87
4.4. Ergebnisse Szenario 1, Punkt A 88
4.5. Eingangsdaten für die Vorhersagen in Szenario 1, Punkt B 89
4.6. Ergebnisse Szenario 1, Fall B 91
4.7. Eingangsdaten für die Vorhersagen in Szenario 2, Bandbreite variiert . . 92
4.8. Eingangsdaten für die Vorhersagen in Szenario 2, Latenz variiert 93
4.9. Ergebnisse Szenario 2, Variation der Bandbreite zwischen den Rechnern 94
4.10. Ergebnisse Szenario 2, Variation der Latenz zwischen den Rechnern . . . 95
4.11. Vergleich der verwendeten Implementierungen von Gleichungslösern . 97
4.12. Eingangsdaten für die Vorhersagen in Szenario 3 98
4.13. Ergebnisse Szenario 3 - Untersuchung verschiedener Verfahren, 16 Prozesse 99
4.14. Ergebnisse Szenario 3 - Untersuchung verschiedener Verfahren, 32 Prozesse 100

A.1. Menüpunkte des Grid Configuration Managers für Grid Konfigurationen 116
A.2. Befehle des Startup-Server Menüs 119

Verzeichnis der verwendeten Abkürzungen

AJO ..... Abstract Job Object
API ..... Application Programming Interface
ATM ..... Asynchronous Transfer Mode
DAMIEN ..... Distributed Applications and Middleware for Industrial use of European Networks
ftp ..... File Tranfer Protocol
GASS ..... Globus Access to Secondary Storage
GCM ..... Grid Configuration Manager
GGF ..... Global Grid Forum
GIIS ..... Grid Index Information Service
GRAM ..... Grid Resource and Allocation Management
GRIP ..... GRid Interoperability Project
GRIS ..... Grid Resource Information Servcie
GSI ..... Grid Security Infrastructure
GT ..... Globus Toolkit
https ..... Hypertext Transmission Protocol Secure
IDB ..... Incarnation Database
ITBL ..... Information Technology Based Laboratory
LDAP ..... Lightweight Directory Access Protocol
MDS ..... Monitoring and Discovery Service
MPI ..... Message Passing Interface
NJS ..... Network Job Supervisor
OGSA ..... Open Grid Service Architecture
P2P ..... Peer-to-Peer
PKI ..... Public Key Infrastructure
QMR ..... Quasi-Minimal-Residual
QoS ..... Quality of Service
RMI ..... Remote Method Invocation
RPC ..... Remote Procedure Call
RSL ..... Resource Specification Language

scp	Secure Copy
SMP	Symmetric Multi-Processing
ssh	Secure Shell
SSL	Secure Socket Layer
TFQMR	Transpose Free QMR
TME	Task Mapping Editor
TSI	Target System Interface
UNICORE	UNiform Interface to COmputing Resources
UPL	UNICORE Protocol Layer
Usite	UNICORE Site
UUDB	UNICORE User Database
VO	Virtuelle Organisation
Vsite	Virtual Site
WAN	Wide Area Network
XML	eXtentisble Markup Language

1. Motivation und Einleitung

In den letzten Jahren hat eine neue Technologie unter dem Namen „Grid“ Einzug in Wissenschaft und Industrie gehalten, die es Forschern in aller Welt ermöglicht, anspruchsvolle, komplexe Probleme zu lösen und in neuartiger Weise zusammenzuarbeiten. Das Ziel dieser neuen Technologie ist es, eine bessere Auslastung von Ressourcen (Rechner) zu erreichen, die Ausführung zusätzlicher Berechnungen, für die bisher keine Kapazitäten zur Verfügung standen zu ermöglichen und bei Bedarf eine größere Leistungsfähigkeit zur Verfügung zu stellen (z. B. um Peaks auszugleichen). Vereinfacht lässt sich sagen, dass das, was unter „Web“ als Service für die Nutzung von Informationen über das Internet bekannt wurde, als Grid einen Service für die gemeinsame Nutzung von Rechenleistung und Datenspeicherkapazität über das Internet darstellt. Das Grid geht dabei über die einfache Kommunikation zwischen Computern hinaus und zielt darauf ab, das globale Netzwerk von Computern in eine einzige riesige Ressource zu verwandeln [19].

Zurzeit werden Grids noch immer hauptsächlich im Bereich des Hochleistungsrechnens bei verschiedenen wissenschaftlichen Anwendungen benutzt, z. B. bei der Simulation komplexer ingenieurwissenschaftlicher Vorgänge wie Strömung in einer Wasserturbine. Der Anwendungsbereich von Grids verschiebt sich derzeit aber auch von der rein wissenschaftlichen (e-Science) hin zur geschäftsmäßigen Anwendung (e-Business). Anwendungen im Bereich des e-Business sollen von den Vorteilen des Grid Computing profitieren und dabei helfen, eine bessere Zusammenarbeit zwischen geografisch verteilten (Teil-) Unternehmen zu organisieren damit für das Unternehmen wirtschaftliche Vorteile entstehen.

Es handelt sich um ein relativ junges Forschungsgebiet und daher existieren neben der oben genannten Definition für den Begriff „Grid“ zurzeit eine Vielzahl teilweise überlappender und sich ständig erneuernder Begriffsdefinitionen. Zuerst wird daher in der Einleitung versucht, eine Beschreibung zu finden, die den Begriff des „Computational Grids“ erläutert, da diese Ausprägung von Grid den Hintergrund dieser Arbeit bildet. Des Weiteren wird ein Modell vorgestellt, dass die Architektur eines Grids beschreibt und verschiedene Typen von Computational Grids erläutert. Dabei soll kein vollständiger Überblick gegeben werden, sondern lediglich die Grundlage zum Verständnis der folgenden Kapitel gebildet werden. Anschließend werden Probleme spezifiziert, die beim Management von Anwendungen in Grid Umgebungen auftreten und daraus die Ziele dieser Arbeit abgeleitet.

1.1. Definition des Begriffes „Computational Grid“

Grids sind Hochleistungsnetze, die die gesamte Kapazität aller angeschlossenen Rechner nutzen und das Austauschen von Daten und Rechenkapazitäten zwischen einzelnen Computern ermöglichen. Dabei liegen die entscheidenden Herausforderungen im Bereich der Software: Ziel ist eine einheitliche und leicht zu bedienende Zugangsmöglichkeit für alle Benutzer zu schaffen und möglichst einfachen und transparenten Zugriff auf die verteilten Ressourcen zu ermöglichen – vergleichbar dem Stromnetz. Das spiegelt sich in der Definition aus dem Jahr 1998 wieder:

> „A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.“ [31]
> Ein Grid ist eine Hard- und Software-Infrastruktur, die einen zuverlässigen, einheitlichen, ständigen und kostengünstigen Zugang zu Höchstleistungs-Rechnerressourcen zur Verfügung stellt. (Übersetzung des Autors)

Bei dieser Infrastruktur handelt es sich sowohl um Hard- als auch um Software mit der ein zuverlässiger („dependable“) Zugang zu dauerhafter, prognostizierbarer Rechenleistung ermöglicht wird. Die gleiche („consistent“) Leistung soll unabhängig von Zugangsort und -art und zu jeder Zeit („pervasive“) zur Verfügung stehen. Die Akzeptanz und Nutzung eines solchen Systems ist abhängig von der Bezahlbarkeit des Zugangs („inexpensive“).

In weiteren Veröffentlichungen wandelt sich die ursprüngliche Grid Definition in Richtung eines Konzeptes als „koordinierte Ressourcenteilung und Problemlösung in dynamischen, multi-institutionellen virtuellen Organisationen“ [30]. Damit wird die Grid-Definition um gesellschaftliche und politische Aspekte erweitert.

Die wichtigste Aussage aller dieser Definitionen bezieht sich auf „einheitlichen, verteilten Zugang zu Ressourcen“. Grids stellen deshalb eine Weiterentwicklung von verteilten Systemen dar, die als eine Ansammlung von unabhängigen Rechnern, die für ihren Benutzer als ein System angesehen werden können [80], definiert werden. Der Ressourcenbegriff im Zusammenhang mit Grid umfasst allerdings nicht nur Rechner. Unter Ressource innerhalb eines Grids werden verschiedene Hard- und Softwareressourcen verstanden, z. B. Rechenkapazitäten (CPU), Speicherplatz, Daten, Programme und Sensoren.

Grids stellen durch diese Integration beträchtliche Pools an Rechenleistung und Diensten zur Verfügung. Das ist momentan z. B. nur durch traditionelle Höchstleistungsrechner möglich, die gewöhnlich an einem einzelnen Standort stehen. In [24] hat Foster folgende Punkte vorgeschlagen, um festzustellen, ob ein verteiltes System ein Grid ist:

1. Ein Grid koordiniert Ressourcen, die nicht zentral gesteuert werden. Das Grid integriert und koordiniert Ressourcen und Benutzer aus unterschiedlichen Verwaltungsdomänen und behandelt Aufgaben wie Sicherheit, Benutzungsrichtlinien (Policies), Teilnahme, usw.
2. Ein Grid verwendet standardisierte, offene und allgemeine Schnittstellen und Protokolle, die grundlegende Fragen wie Authentifizierung, Autorisierung, Auffinden und Zugriff auf Ressourcen lösen.
3. Ein Grid liefert nicht triviale QoS (Quality of Service). Ein Grid erlaubt eine koordinierte Nutzung seiner Ressourcen, die verschiedene QoS ermöglicht, bzgl. Antwortzeit, Durchsatz, Verfügbarkeit, Sicherheit.

Demnach ist die Kerneigenschaft eines Grids seine Fähigkeit, die verteilte Benutzung von Ressourcen innerhalb einer bestimmten Gruppe von Mitgliedern (Anbieter und Nutzer) zu verwalten. Dieser Zusammenschluss aus Nutzern und Ressourcenanbietern, die aus realen Organisationen stammen, wird auch als virtuelle Organisation (VO) bezeichnet. Diese bildet vereinbarte Policies (Regeln für die Mitgliedschaft und die Benutzung der Ressourcen) in den einzelnen Institutionen ab. Damit ist es für die Teilnehmer möglich zu bestimmen, was gemeinsam benutzt werden darf, von wem, wie und unter welchen Bedingungen.

Da die Ressourcen innerhalb eines Grids dynamisch sind, also ständig neue Quellen angeschlossen und entfernt werden können, sind serviceorientierte Systeme praktisch ein Muss für die Struktur. Dabei werden Einheiten in Form von Programmen (plattformunabhängige Software-Komponenten) benutzt, die für einen Benutzer ein Netzwerk nach vorgebbaren Zielen und dazugehörigen Kriterien durchsuchen. Diese neueren Trends in der Grid Entwicklung werden z. B. im OGSA (Open Grid Services Architecture) [23] Konzept zusammengefasst. Ein weiter gehender Ansatz, der bislang noch nicht vollständig realisiert wurde, liegt in der Nutzung von Techniken des Peer-to-Peer Computing (P2P) für das Grid Computing [26]. In einem Peer-to-Peer-Netz sind alle Computer gleichberechtigt und können sowohl Dienste in Anspruch nehmen als auch Dienste zur Verfügung stellen. Die flachen, nicht-hierarchischen P2P-Systeme versprechen eine höhere Skalierbarkeit und eine bessere Ausfallsicherheit durch Redundanz.

1.1.1. Grid Architektur

In der Literatur [57, 62, 20] gibt es viele Ansichten darüber, wie man die Architektur eines Grids beschreiben kann. Eine der allgemein benutzten Beschreibungen wird in [20] beschrieben und reflektiert die Grid Architektur aus der Sicht eines Benutzers. Bild 1.1 zeigt dieses Schichtenmodell der Grid Infrastruktur. Auf der untersten Ebene sehen wir das so genannte Grid Grundgerüst (die physikalischen Ressourcen aus denen ein Grid

besteht) sowie die benötigten lokalen Dienste, um auf diese Ressourcen Zugriff zu erhalten (z. B. Ressourcen Manager). Die nächste Schicht stellt die Sicherheits-Infrastruktur dar, die für die sichere Ausführung von Operationen im Grid essentiell ist. Danach folgen die Kern-Grid-Services und Services für die Benutzer die bereitgestellt werden müssen, damit die Grid Infrastruktur überhaupt sinnvoll verwendet werden kann. An oberster Stelle gibt es die Grid Programmierumgebungen und die Applikationen, welche die Benutzung des Grids vereinfachen sollen. In der Praxis kann man generell nur über die Funktionalitäten der verschiedenen Schichten sprechen, denn in den aktuellen Implementierungen sind die Schichten zum Teil verwischt.

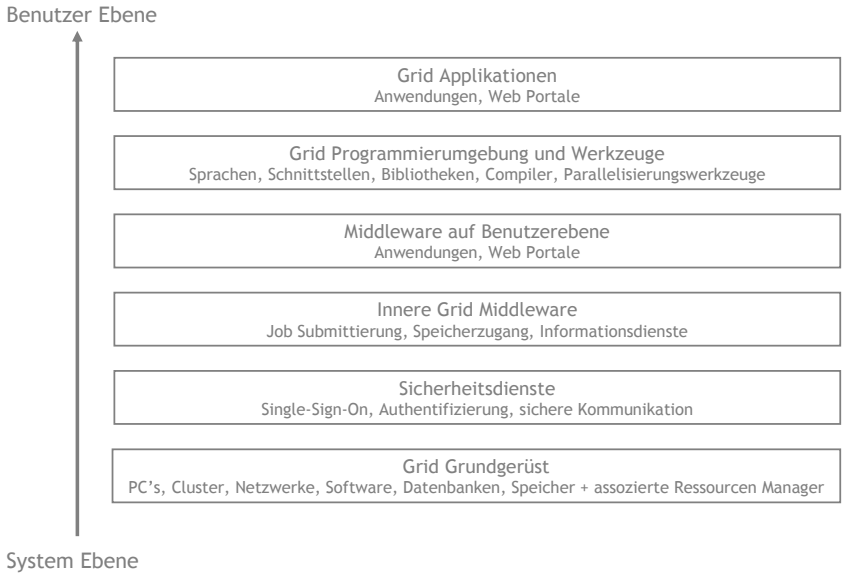

Abbildung 1.1.: Grid Architektur im Schichtenmodell

Im folgenden Abschnitt werden die einzelnen Schichten detailliert erläutert:

Grundgerüst: Dies umfasst die schon existierenden Ressourcen, innerhalb einer Grid Infrastruktur vereinheitlicht werden sollen. Dabei handelt es sich um Rechen- und Datenressourcen sowie andere spezielle Ressourcen wie Monitoring- und Messwerkzeuge. Diese Ressourcen sind normalerweise unter der Kontrolle eines lokalen Managementprogramms wie z. B. LoadLeveler [49], LSF [70], PBS [85] etc. Diese Manager stellen teilweise auch erweiterte Funktionalitäten wie (Vorab-)Reservierung o. ä. zur Verfügung, die wiederum von den höheren Schichten der Grid Infrastruktur verwendet werden können, um z. B. mehrere Ressourcen zeitlich koordiniert zu reservieren, um die Ausführung verteilter paralleler Programme zu ermöglichen .

Sicherheits-Infrastruktur: Diese Schicht befindet sich direkt über den physikalischen Ressourcen und unterhalb der Inneren Grid Middleware. Für eine verteilte Umgebung sind Autorisierung und Authentifizierung sicherheitstechnisch gesehen die zwei wichtigsten Funktionen. Das Problem, die Bereitstellung dieser Sicherheitsservices innerhalb einer Grid Umgebung zu gewährleisten, stellt sich viel komplexer dar, da die in einem Grid zusammengefassten Ressourcen normalerweise von unabhängigen Organisationen kontrolliert werden, die jeweils eigene Sicherheitsanforderungen und -regeln haben. Daraus folgt, dass die Interaktion mit lokalen Sicherheitsstandards, ohne dass diese selbst verändert werden müssen, eine der Grundanforderungen für die Grid Sicherheitsinfrastruktur darstellt. In den zurzeit verwendeten Grid Systemen wird dieses Problem im Allgemeinen dadurch gelöst, dass eine Abbildung der Grid Sicherheitsmechanismen auf jeden lokalen Sicherheitsmechanismus erfolgt, z. B. auf lokale schon vorhandene Benutzernamen. Falls erforderlich, können zusätzlich zu den vorhandenen lokalen Sicherheitsmechanismen innerhalb eines Grids auch für bestimmte Benutzergruppen angepasste Sicherheitsmechanismen zur Anwendung kommen. Eine wichtige und sehr kontrovers diskutierte Technik, ist die Übertragung von Berechtigungen (credential delegetion) die mit Ziel eingesetzt wird, eine dynamische Änderung an laufenden Jobs und der Ausnutzung von Ressourcen zu ermöglichen. Das bedeutet, eine Ressource im Grid muss in der Lage sein, anderen Ressourcen die Erlaubnis zu erteilen an Stelle der Originalressource zu handeln. Diese Methode bildet z. B. die Basis der Sicherheitsinfrastruktur von Globus [28].

Innere Grid Middleware: Die Services dieser Schicht haben die Aufgabe eine Abstraktion der darunter liegenden Ressourcen zu ermöglichen und diese zu verwalten. Deshalb werden hier die folgenden Dienste angeboten: Dienste für das Ressourcenmanagement, Informationsdienste und Dienste für den Datentransfer. Ressourcenmanagementdienste stellen einheitliche und sichere Schnittstellen zwischen den geografisch und administrativ entfernten Bereichen zur Verfügung, z. B. zum Verteilen von Jobs. Auf einer einzelnen Rechenressource sind diese Dienste auch für die Zusammenarbeit mit den lokalen Scheduling-Systemen verantwortlich. Die Informationsdienste sammeln und verbreiten Daten über den Status der Ressource (Rechenkapazität, Netzwerkauslastung, Speicherkapazität etc.). Diese Informationen können dann von einer Vielzahl von anderen Diensten abgerufen werden, um weiterführende Informationen zu erzeugen (z. B. um einen Vergleich verschiedener Ressourcen vornehmen zu können). Die Datentransferdienste ermöglichen den sicheren (authentifiziert und autorisiert) Datenaustausch zwischen administrativ unterschiedlichen Bereichen innerhalb eines Grids.

Middleware auf Benutzerebene: Hier werden Services zur Verfügung gestellt, die über verschiedene Knoten des Grids hin verteilt arbeiten, u. a. Informationsdienste, Discovery Services, Resource Broker und Scheduler. Der Grid Informationsdienst speichert und bereitet Informationen über den Status des gesamten Grids auf (einzelne Ressourcen, virtuelle Organisationen, Netzwerke etc.) Spezielle Discovery Services (Dienste zum

Auffinden) haben die Aufgabe, andere Dienste und Ressourcen anhand spezifischer Benutzerangaben innerhalb des Grid Systems zu finden. Resource Broker und Scheduler ermöglichen die (Vorab)-Reservierung und (zeitlich gesteuerte) Vergabe von Ressourcen, die mit abstrakten Mechanismen auf Benutzerebene angefordert werden können. Das ist z. B. für die Verarbeitung von verteilten parallelen Anwendungen von größter Wichtigkeit. Außerdem befinden sich hier Dienste für den virtualisierten Zugang zu im Grid verteilten Datenspeichern.

Die beiden restlichen Ebenen werden oft unter dem Begriff „Problem Solving Environments“ (PSE) [32] zusammengefasst.

Grid Programmierumgebung und Werkzeuge: Die in dieser Schicht zur Verfügung gestellten Programmierumgebungen und Werkzeuge stellen gewissermaßen eine Zwischenschicht zwischen den darunter angesiedelten systemnahen Grid Diensten und den Grid Applikationen dar - sie ermöglichen die „Grid-Fähigkeit“ von Applikationen oder anders gesagt: sie stellen die Werkzeuge bereit, um Grid Applikationen zu erzeugen. Oftmals geschieht dies durch Anpassung schon vorhandener Programmierumgebungen an die Erfordernisse des Grid.

Grid Applikationen: Wie oben schon erwähnt, findet man als letzte Schicht innerhalb eines Grids die für den Endbenutzer aufbereiten Anwendungen. Diese haben oft grafische Benutzerschnittstellen und erlauben es dem Benutzer sich auf die Lösung seiner spezifischen Probleme zu konzentrieren, ohne sich direkt mit den darunter liegenden Schichten befassen zu müssen.

1.1.2. Anwendungsklassen von Computational Grids

In aktuellen Analysen zum Stand der Grid Forschung [10] wird gesagt, dass die ursprüngliche Idee des Grid Computing eine Erweiterung des parallelen Rechnens von eng gekoppelten Clustern zu geografisch verteilten Systemen vorsah. In der Praxis hat sich aber gezeigt, dass das Grid mehr als eine Plattform für die Integration von lose gekoppelten Anwendungen und für die Vernetzung ungleicher Ressourcen (Speicher, Rechner, Visualisierungsinstrumente) benutzt wird. Das deutet darauf hin, dass sich die generelle Sichtweise des Grid Computing erweitert hat. Grid Technologie wird verwendet, um viele verschiedene Probleme zu lösen oder komplett neue Szenarios zu ermöglichen. Die zurzeit möglichen Ausprägungen von Grid und was mit diesen erreicht werden kann, sind im folgenden Abschnitt zusammengefasst:

- *Verteiltes Höchstleistungsrechnen (Distributed Supercomputing)*
  Damit bezeichnet man das Koppeln von mehreren evt. verteilten Rechnerressourcen. Diese Infrastruktur besteht normalerweise aus HPC Ressourcen. Oft ist eine dedizierte Netzwerkverbindung zwischen den beteiligten Rechenzentren vorhanden. Da die Komponenten dieser Umgebungen von Partnern stammen, die sich

gegenseitig vertrauen, können Sicherheitsaspekte z. T. in den Hintergrund treten. Im Rahmen des GWAAT (Global Wide Area Applications Testbed) Projektes [72] hat das HLRS gemeinsam mit dem Pittsburgh Supercomputing Center (PSC) erfolgreich mehrere Tests mit gekoppelten Höchstleistungsrechnern durchgeführt.

- *Durchsatzrechnen (High-Throughput Computing)*
  Bei dieser Art werden viele aktuell zur Verfügung stehende Rechnerressourcen zusammengefasst, die im Kollektiv an der Lösung von unabhängigen Teilen ein und desselben Problems arbeiten. Bei diesen Ressourcen handelt es sich meist um viele zum Zeitpunkt der Anforderung im Leerlauf befindliche Desktop Computer, weshalb dafür auch der Begriff des „cycle scavenging“ verwendet wird. Ein Beispiel für diese Lösung ist das populäre SETI@home Projekt [65]. High Troughput Computing kommt oft auch bei der Erstellung von Parameterstudien zum Einsatz.

- *Desktop Supercomputing*
  Dieser Begriff bezieht sich auf darauf, wie man den Zugang zu Supercomputern für Benutzer leichter gestalten kann, z. B. für entfernten Zugriff von den Benutzer Workstations aus, ohne dass diese einen tiefen Einblick in die Konfiguration solcher Systeme benötigen, um einen Job ausführen zu können, siehe z. B. das UNICORE Projekt [84]. Diese Klassifikation schließt das Modell des „cycle scavenging“ nicht ein und sollte auch nicht damit verwechselt werden.

- *Datenintensives Rechnen (Data-Intensive Computing)*
  Ist die Vernetzung von Daten mehrerer, verteilter File Systeme um einen großen verteilten Datenspeicher zu bilden. Es ermöglicht z. B. die numerische Simulation oder das Sammeln und Speichern von Resultaten von Experimenten in großem Maßstab. Ein Beispiel dafür ist das europäische DataGrid Projekt [76].

- *Kollaboratives Rechnen (Collaborative Computing)*
  Ermöglicht und verbessert die Interaktion zwischen beteiligten Personen, die an der Lösung einzelner, gekoppelter Probleme arbeiten. Das Wesen von virtuellen Organisationen schließt auch die Koordination von menschlichen Ressourcen mit ein. Kollaborative Umgebungen erlauben den Teilnehmern, die an mehreren Standorten verteilt sind, gemeinsam die Grid weiten Aktivitäten zu beobachten und zu kontrollieren.

- *On-Demand Computing*
  Stellt den Zugriff auf Ressourcen sicher, die kurzfristig benötigt werden. Diese Funktionsweise weist die größten Parallelen zum elektrischen Stromnetz auf, wo zu einem bestimmten Zeitpunkt eine definierte Menge an Leistung für Aufgaben zur Verfügung gestellt werden kann.

1.2. Probleme beim Management verteilter Anwendungen in heterogenen Grid Umgebungen

Die an die Grid Technologie gestellten Erwartungen sind hoch. Ebenso groß sind allerdings die Problemfelder, die aufgrund der Komplexität solcher Hochleistungsnetze entstehen. Neben seiner eigentlichen Aufgabe, der rechnerunterstützten Lösung eines wissenschaftlichen Problems, muss sich der Benutzer von wissenschaftlichen Anwendungen in Grid Umgebungen mit folgenden Problemen auseinander setzen:

1. **Heterogenität**
   Die Verteilung eines Jobs auf verschiedene Rechner kann viele Schwierigkeiten für den Benutzer und die Anwendung verursachen. Die meisten dieser Probleme rühren von der Tatsache her, dass „das Grid“ nicht existiert, sondern aus einer Vielzahl heterogener „Grid-Inseln“ besteht, die unter verschiedenen Aspekten betrachtet werden müssen.

   Das fängt mit unterschiedlichen Datenrepräsentationen auf verschiedenen Rechnern an, was eine Umwandlung der Daten in den verwendeten Bibliotheken oder direkt in der Anwendung selbst erfordert. Verschiedene Prozessorgeschwindigkeiten und Unterschiede in den vorhandenen Speichergrößen machen es erforderlich, dass Anwendungen über eine intelligente Anfangslastverteilung und/ oder dynamische Lastbalancierung verfügen müssen [74].

   Ein anderer Aspekt der Heterogenität wird oft außer Acht gelassen. Benutzer müssen sich noch immer damit auseinander setzen, dass Rechner verschiedenartigste Zugangsmechanismen haben und die gleiche Anwendung jeweils unterschiedlich gestartet werden muss. Er muss sich darum kümmern, welches System mit welchem Betriebssystem wie zu bedienen ist. Er muss sich mehrfach authentifizieren und dabei unterschiedliche Loginnamen und Passwörter verwenden. Während einige Rechenzentren ssh, rsh oder ähnliche Protokolle verwenden, werden zunehmend auch Grid Umgebungen wie Globus [27], Legion [46] oder UNICORE [84] als Zugangsportal für die Rechner angeboten. Es besteht somit eine große Wahrscheinlichkeit, dass sich der Benutzer mit einem Konglomerat dieser Zugangsmöglichkeiten befassen muss, wenn er seine Anwendung auf die verschiedenen Zielrechner verteilen will. Das ist genau der Aspekt, wenn im folgenden von heterogenen Grid Umgebungen gesprochen wird.

   Neben den o. g. Problemen, beeinflusst dieser Aspekt auch die Startprozeduren von MPI-Bibliotheken, einem häufig für parallele Anwendungen verwendeten Programmiermodell. Entwickler von Bibliotheken tendieren dazu, ihre Kommunikationsbibliotheken und die Startprozeduren nur für bestimmte Grid Umgebungen zu zuschneiden. So ist z. B. MPICH-G2 Teil der Globus Umgebung und benutzt Globus Mechanismen für Authentifizierung, Autorisierung und Kommunikation.

Das bedeutet aber, dass es fast unmöglich ist, einen MPICH-G2 Job auf einem Rechner laufen zu lassen, auf der die Globus Software nicht installiert ist.

2. **Grids besitzen keine zentrale Kontrollinstanz**
   Das Verteilen einer Anwendung auf einer Menge heterogener, sich dynamisch ändernder Ressourcen, erfordert die Koordination der verwendeten Ressourcen und stellt somit ein komplexes Problem dar. Zurzeit gibt es keine allgemeinen Strategien und Systeme für Scheduling innerhalb von Grids die alle Bedürfnisse befriedigen können [83]. Die zur Verfügung stehenden Scheduling Systeme sind oft auf die spezifischen Charakteristika der Zielressourcen oder spezielle Anwendungsszenarien zugeschnitten.

   Die Komplexität der Anwendungen und die Anforderungen der Benutzer auf der einen Seite und die Heterogenität der Systeme auf der anderen erlauben es auch nicht, eine effiziente manuelle Durchführung der Scheduling Prozesse aufzusetzen. Die Aufgabe, das Scheduling für eine Anwendung durchzuführen, schließt nämlich nicht nur die Suche nach passenden Ressourcen entsprechend den Benutzervorgaben ein. Das Scheduling System muss zusätzlich auch die Koordination von verschiedenen unterschiedlich ausgelasteten Ressourcen vornehmen, die zu einem bestimmten Zeitpunkt zur Verfügung stehen sollen, um die Anwendung optimal durchführen zu können. Dafür existiert momentan keine allgemeine, generische Lösung. Vielleicht wird es nie eine geben, denn die einzelnen Anwendungsszenarien für sich gesehen machen es erforderlich, dass sehr spezielle Scheduler zur Verfügung stehen, damit sie effizient durchgeführt werden können. Man kann allerdings diverse Gemeinsamkeiten in existierenden Grid Schedulern feststellen, was zu der Annahme führt, dass eine generische Architektur vorstellbar ist, die nicht nur die Implementierung von Grid Schedulern vereinfacht, sondern auch die Interaktion zwischen diesen möglich macht.

   Ein weiteres Problem, das durch die fehlende zentrale Steuerung entsteht, ist die Datenhaltung. Diese liegt weitestgehend in der Verantwortung des Benutzers. Er muss Daten manuell zwischen den Rechnern transportieren, muss wissen, wo welche Mengen von Daten für welchen Zeitraum abgelegt werden dürfen. Er muss angeben, wo sich die Eingabedaten für seine Anwendung befinden und wohin Ausgabedaten geschrieben werden sollen. Die zurzeit angebotenen Lösungen für solche Datengrids sind meist nur auf eine bestimmte Anwendergruppe oder Anwendung zugeschnitten [76] und können nicht auf andere Projekte übertragen werden.

3. **Grids sind dynamische Systeme**
   Zusätzlich zu den vorher genannten Problemen beim Scheduling muss in Betracht gezogen werden, dass sich der Zustand von Ressourcen und der Bedarf innerhalb eines Grids dynamisch ändern können und dadurch eine gewisse Unvorhersagbar-

keit von Ereignissen entsteht. Durch Ausfall von Rechnern, Wartungsarbeiten etc. stehen z. B. Ressourcen nicht mehr zur Verfügung, auf der anderen Seite werden aber auch neue Rechner und Anwendungen von Anbietern hinzugefügt. Dafür wurden verschiedene Resource Brokering Systeme und Grid Scheduler entwickelt [17, 81], die allerdings jeweils nur einen Teil der Probleme für sehr spezifische Anwendungen lösen können.

Ein anderer Weg für die Lösung dieses Problems wird durch so genannte fehlertolerante Anwendungen angeboten. Das sind Anwendungen, die aufgrund ihrer Implementierung Ausfälle von Ressourcen ohne Absturz überleben können. Es gibt dabei verschiedene Ansatzpunkte. Zum einen existieren Anwendungen, die durch bestimmte Programmiertechniken und Algorithmen fehlertolerant sind [1]. Zum anderen gibt es fehlertolerante Implementierungen des MPI-Standards, z. B. MPICH-V [12] oder LAM/MPI [16]. Diese basieren in der Regel auf check-point/roll-back Technologien oder verwenden Replika Techniken. Ein großer Nachteil aller dieser Anwendungen liegt im durch die Fehlertoleranz verursachten Mehraufwand, der zum Teil erhebliche Einbußen bei der Effizienz der Anwendungen erzeugt.

4. **Sicherheits- und Vertraulichkeitsaspekte**
   Sicherheit ist ein wesentliches Problem in Grid Umgebungen [82], denn dieser Aspekt beeinflusst sowohl die Akzeptanz der Benutzer als auch der Anbieter. So möchten Benutzer sicherstellen, dass die Daten, die sie in einer Grid Umgebung bearbeiten vertraulich behandelt werden. Ressourcenanbieter wollen verhindern, dass unerlaubte Zugriffe auf die Ressourcen erfolgen. Die Grundlagen für die Sicherheit im Grid sollten deshalb auf Standards basieren, welche bereits erprobt sind, sprich schon häufig verwendet werden. Nur so können Sicherheitslücken weitestgehend ausgeschlossen werden.

   Die Grid Sicherheitstechnologien müssen verschiedene Aufgaben erfüllen. Benutzer müssen befähigt sein, sich einmalig einzuloggen und danach Zugriff auf alle ihre entsprechenden Ressourcen zu haben (Single-Sign-On). Dem Benutzer muss es möglich sein, seine Rechte an ein Programm weiterzugeben, welches auf seine entsprechenden Ressourcen zugreifen kann. Es muss eine Integration von verschiedenen lokalen Sicherheitslösungen angeboten werden, damit jeder Ressourcenprovider die Möglichkeit hat, weiterhin eigene Sicherheitslösungen zu verwenden. Die Gridsoftware muss mit ihnen interagieren können. Das Sicherheitssystem muss Benutzerbezogen sein. Dies bedeutet, dass sich die jeweiligen Ressourcenanbieter nicht kennen müssen und keine Konfigurations- und Administrationsinformationen austauschen müssen. Des Weiteren sollte die Grid Sicherheitslösung ein flexibles Auswählen der Sicherheitsstufe ermöglichen und mit verbindungslosen sowie verbindungsorientierten Protokollen arbeiten können.

5. **Latenzzeiten und Bandbreiten bei der Übertragung**
   Die Unterschiede in den Kommunikationsmustern für die Kommunikation zwischen den Prozessen, die sich potentiell auf unterschiedlichen Rechnern befinden, erfordern spezielle Programmiertechniken um die hohen Latenzen und die niedrigen Bandbreiten zwischen geografisch weit verteilten Rechnern handhaben zu können. Die letztgenannten Aspekte wurden in den letzten Jahren von verschiedenen Projekten behandelt, deren Ziel es war, MPI-Bibliotheken für Grid Umgebungen zu entwickeln z. B. MPICH-G2 [54], Stampi [51], Grid-MPI [45] und PACX-MPI [39].

6. **Kosten**
   Für den Aufbau und die Administration einer Grid Infrastruktur ist es wichtig, dass die Teilnehmer die Kosten, die durch die Nutzung von Hard- und Software, Wartung etc. anfallen auf faire Weise untereinander aufteilen. Es existiert allerdings bisher keine Lösung für die Kostenberechnung, die die gegenseitige Bereitstellung und Nutzung von Grid Ressourcen widerspiegelt. Eine Art Computational Economy ist erforderlich, die Mechanismen bereitstellt um die Grid Ressourcen anhand von Bedarf und Verbrauch zu regulieren. Auf der einen Seite müssen Anreize geschaffen werden, damit die Besitzer von Ressourcen diese bereitstellen wollen. Diese könnten z. B. die Ausnutzung ihrer Ressourcen maximieren, indem sie durch günstige Zugangspreise attraktiv für Benutzer werden. Auf der anderen Seite sollen die Benutzer (Ressourcen Verbraucher) die Möglichkeit haben, den Anbieter auszuwählen der am besten ihre Erfordernisse befriedigt. Solange allerdings der Zugang z. B. zu Hochleistungsrechnern durch staatliche und politische Gremien verwaltet wird, wird es schwer sein, einvernehmliche Lösungen zu finden, diese Ressourcen auf einem wettbewerbsorientierten und transparenten Markt anzubieten.

   Auch die technischen Lösungen für dieses Problem befinden sich zurzeit noch in der Entwicklungsphase. Das Kostenproblem muss nämlich zusätzlich zu den unter Punkt 2 genannten Funktionalitäten in die Implementierung von Grid Schedulern einfließen. Es gibt nur wenige auf bestimmte spezifische Grid Umgebungen zugeschnittene Lösungen, z. B. wird in [18] ein Kostenmodell für den Nimrod/G Resource Broker beschrieben.

1.3. Ziele dieser Arbeit

Alle oben genannten Probleme zu lösen, würde den Rahmen dieser Arbeit übersteigen, da sich z. B. im Bereich des Schedulings noch keine konkrete Lösung abzeichnet. Stattdessen wird hier ein Weg vorgeschlagen, Benutzer von wissenschaftlichen Anwendungen bei der Handhabung der im Grid implizit vorhandenen Heterogenität zu unterstützen

und Lösungsvorschläge für die effiziente Nutzung von Rechnerkapazitäten im Grid zu unterbreiten.

Deshalb steht an erster Stelle der Entwurf eines Konzeptes für eine benutzerfreundliche **Umgebung für das Management von parallelen Anwendungen in heterogenen Umgebungen**. Diese Umgebung soll als eine Art Vermittler zwischen Anwendungen und vorhandenen Ressourcen agieren und den Benutzer bei der Durchführung seiner Aufgaben unterstützen. Speziell sollen Anwender in der Lage sein, ihre schon vorhandenen Programme innerhalb einer heterogenen Grid Umgebung ausführen zu können. Des Weiteren sollen sie bei Entscheidungen unterstützt werden, wie ihre Anwendungen in optimaler Weise auf die verschiedenen Rechnerressourcen verteilt werden. Das Ziel des Konzeptes soll sein, die Effektivität bei der Durchführung von Anwendungen auf den im Grid angebotenen Ressourcen zu erhöhen. Dadurch können nicht nur Laufzeiten für einzelne Anwendungen und damit die Kosten für Rechenzeit gesenkt werden, sondern auch der gesamte Durchsatz in Grid Umgebungen erhöht werden.

Danach wird die **Umsetzung und Implementierung** des o. g. Konzeptes als eine graphische Benutzeroberfläche vorgestellt. Die Benutzer werden bei verschiedenen Schritten der Durchführung von komplexen wissenschaftlichen Anwendungen unterstützt: Konfiguration innerhalb einer heterogenen Grid Umgebung einschließlich automatischem Verteilen von Konfigurationsdateien, Starten und Überwachung der Anwendung, Dateitransfer von Ein-/Ausgabedateien, sowie Sammeln von Daten für eine automatische Vorhersage der Performance.

In einem nächsten Schritt soll die **Einsatzmöglichkeit der Umgebung für unterschiedliche wissenschaftliche Anwendungen** untersucht werden. Dabei wird im Besonderen auf zwei Szenarios eingegangen: Zum einen soll gezeigt werden, wie den Benutzern geholfen wird, die bestmögliche Ressourcenkonfiguration für ihre Anwendung zu finden. Die Aufteilung der Prozesse von parallelisierten Anwendungen auf die verteilten Ressourcen spielt eine große Rolle für das Laufzeitverhalten und damit der Leistung der Anwendung innerhalb des Grids. Das System soll in der Lage sein, eine automatische Vorhersage über die Leistung/Ausführungszeit der Anwendung innerhalb bestimmter Rechnerkonfigurationen zu liefern und die Ergebnisse in Form von globalen Metadaten abzuspeichern. Damit wird der Benutzer bei der Entscheidung unterstützt, welche Verteilung der Prozesse er für seine Anwendung wählen soll, um die bestmögliche Leistung zu erhalten. Das Auffinden von Ressourcen, die in verschiedene Grid Umgebungen integriert sind, kann dabei abhängig von den Möglichkeiten des jeweiligen Grid Systems automatisiert erfolgen oder muss vom Benutzer selbst vorgenommen werden.

In einem zweiten Szenario soll gezeigt werden, wie Anwendungsentwickler beim Testen von verschiedenen Implementierungen ihrer Anwendungen unterstützt werden. Für bestimmte Algorithmen/Funktionen existieren zum Teil verschiedene Implementierungen, um den unterschiedlichen Anforderungen gerecht zu werden, die auf Grund

verschiedener Laufzeitumgebungen und -konfigurationen in verteilten Umgebungen entstehen. Die Umgebung soll deshalb in der Lage sein, Vorhersagen über das Laufzeitverhalten der verschiedenen Implementierungen zu machen, die in einer festgelegten Konfiguration ausgeführt werden. Damit kann der Benutzer diejenige Implementierung auswählen, die für die jeweilige Konfiguration aufgrund des Laufzeitverhaltens optimal ist.

2. Stand der Technik - Relevante Projekte für Grid Middleware Lösungen

„Grid Computing is a Grid of Technologies“.
Miron Livny (Condor)

Bei heutigen Grid Middleware Lösungen handelt es sich um eine Menge von Technologien und Werkzeugen die es dem Benutzer erlauben, einfachen Zugriff auf Grid Ressourcen und Applikationen zu erlangen. Die darin enthaltenen Grid Computing Environments (kurz GCE) beschreiben grob die Benutzersicht eines Grid Systems [32], wobei es eine unscharfe Grenze zwischen den Grid Computing Environments und dem so genannten Kern des Grids gibt. Letzteres beinhaltet den Zugriff auf die Ressourcen, das Management der Abläufe zwischen denselben, die Sicherheit und andere Kernkompetenzen (siehe Kapitel 1.1.1). Die GCEs ähneln oft einem Webportal oder einer normalen Shell. Natürlich handelt es sich in diesen Fällen um meist viel mächtigere und vielseitigere Tools wie Grid-Portale und Grid-Shells.

Die Grid Middleware Lösungen können auf viele verschiedene Weisen klassifiziert werden. Meistens wird nach den verwendeten Technologien unterschieden bzw. unterteilt. Diese Technologieunterschiede zwischen den einzelnen Projekten spielen natürlich eine große Rolle, aber wichtiger sind die Unterschiede zwischen den benötigten Ressourcen oder den verwendeten Programmiermodellen speziell im Hinblick auf die Anwendbarkeit durch Benutzer. Genauso wie spezielle Features beinhaltet jede Grid Middleware ein besonderes Computing Modell (Ver-/Bearbeitungsweise) für das Grid. Dieses verwendete Modell spiegelt sich in der Architektur der Grid Middleware und in der Sicht, in der sich diese dem Anwender darstellt, wider.

Alle heute verwendeten Grid Systeme setzten voraus, dass sie auf eine oder mehrere entfernte Ressourcen - Quelle(n) Zugriff haben und deshalb versuchen sie, die Verbindung zu diesen Ressourcen möglichst bequem und einfach zu gestalten. Dies hat zur Folge, dass praktisch alle heutigen Grid Lösungen der gleichen „Gundprogrammstruktur“ folgen. Im einfachsten Fall stellt diese Grundstruktur einen laufenden Job [1] dar, der initialisiert werden muss und zu welchem Zugriff gewährt wird (sowohl zum Status des Jobs als auch zum Ergebnis das er berechnet). Komplizierte Anwendungsfälle wie

[1] Ein Job ist ein Programm, dass ein Benutzer auf einem Rechner innerhalb des Grids ausführen möchte.

z. B. zeitaufwendige Simulationen und Visualisierungen benötigen dabei zusätzlich die Möglichkeit, Daten aufeinander abgestimmt zu sammeln und zu verarbeiten.

Im Folgenden soll ein Überblick über verschiedene existierende Middleware Projekte gegeben werden, die momentan im Bereich des Grid Computing weltweit zum Einsatz kommen. Es wird dabei auf die am häufigsten genutzten eingegangen bzw. auf solche, die eine spezifische Unterstützung für MPI-Anwendungen bieten, da diese den Hintergrund dieser Arbeit bilden. Hier eine Liste der betrachteten Projekte:

- Globus
- UNICORE
- TME/ITBL
- Avaki/Legion

Zunächst wird aber im zweiten Unterkapitel dargestellt wie zurzeit auch noch mit herkömmlichen Mitteln verteilte Anwendungen in Grid ähnlichen Strukturen ausgeführt werden, die mittels ssh als Zugangsmechanismus aufgebaut wurden. Ein weiteres Kapitel befasst sich mit aktuellen Entwicklungen auf dem Gebiet der Multi-Grid Lösungen. Am Ende des Kapitels erfolgt noch ein kurzer Überblick über die momentanen Standardisierungsaktivitäten im Grid Bereich.

2.1. Vergleichskriterien

Für den Vergleich der verschiedenen Middleware Lösungen wurde die oben erwähnte Grundprogrammstrukur zugrunde gelegt. Des Weiteren wird auf Eigenschaften eingegangen, die im Zusammenhang mit dem Management von verteilten Anwendungen eine Rolle spielen. Damit ergeben sich als wichtigste Vergleichskriterien:

1. *Auffinden und Verwalten von Ressourcen:* Die Grid Middleware ist üblicherweise eng mit der vorhandenen Ressourcenverwaltung wie z. B. Batch-Systemen verknüpft, um die back-end Ressourcen zu verwalten bzw. zu handhaben. Innerhalb des GCE werden Benutzerschnittstellen angeboten, um Ressourcen zu suchen oder die angebotenen Ressourcen zu verwalten. Diese Schnittstellen können einfach oder komplex sein und sind oft hierarchisch konstruiert. Zum Beispiel kann eine so genannte GCEShell Zugriff auf die Grundfunktionen der Grid Umgebung bieten.
2. *Authentifizierung und Autorisierung:* Eine sehr wichtige und überall benötigte Eigenschaft ist der Umgang mit Sicherheitsaspekten. Benutzer müssen Zugang zu Rechnern, Anwendungen und Daten erhalten. Auf der anderen Seite müssen sich die Anbieter von Ressourcen gegen unerlaubten Zugriff schützen und eine Verschlüsselung von Datenströmen ermöglichen. Die Grid Umgebungen haben

dafür jeweils eigene Lösungen entwickelt, die sich in der Regel mit vorhandenen Sicherheitsmechanismen integrieren lassen. Die Benutzer müssen sich allerdings noch immer mit verschiedenen Zugangsmechanismen befassen und eine Vielzahl von Passwörtern und Zertifikaten verwalten.

3. *Job Submission:* Um eine verteilte Anwendung auf den Zielsystemen zu starten, muss in der Regel bekannt sein, wo sich die Ausführungsdateien und Eingabedaten befinden (Aufgabenverteilung). Diese Angaben (Namen und Pfade) können für jedes Zielsystem unterschiedlich sein. Außerdem muss der Benutzer spezifizieren, welche Ressourcen (z. B. Anzahl und Typ der Prozessoren) er verwenden möchte. Bei MPI-Anwendungen muss zusätzlich noch angegeben werden, wie viele Prozesse auf jedem Rechner gestartet werden sollen. Des Weiteren ist es notwendig, anzugeben, mit welchem MPI-Startkommando die Anwendung jeweils gestartet wird. Dabei haben Benutzer oft das Problem, dass die auf den Rechnern installierten MPI-Bibliotheken unterschiedliche Kommandos verwenden (z. B. mpirun, mpiexec, poe). Innerhalb einer Grid Umgebung wird in der Regel versucht, dafür generische Lösungen anzubieten. So kann z. B. eine grafische GCE bei der Ressourcenbeschreibung helfen und eine Vorauswahl der installierten Bibliotheken anbieten.

4. *Job Überwachung (Monitoring):* Während der Ausführung einer Anwendung kann es notwendig sein, Änderungen vorzunehmen oder Fehler zu korrigieren. Deshalb sollten Benutzer in der Lage sein, Rückmeldungen ihrer Anwendung oder des Systems, auf der der Job läuft, zu erhalten und auf diese zu reagieren. Die Grid Umgebungen sind meistens in der Lage, diese Meldungen an die Benutzer weiterzuleiten. Allerdings kann es bei der Verarbeitung von komplexen, parallelen Anwendungen kompliziert sein, diese Ausgaben zu verarbeiten, da z. B. viele verschiedene Ausgabefenster geöffnet werden müssen, um alle Daten zu sehen.

5. *Dateiverwaltung und -transfer:* Eine Schlüsselkomponente für komplexe Anwendungen in Grid Umgebungen ist die Bereitstellung von Funktionalitäten für das Datenmanagement. Dateitransfers sind erforderlich, um Eingabedaten und Ausführungsdateien zur Verfügung zu stellen und Ausgabedaten an bestimmten, festgelegten Speicherorten abzulegen, um eine vernünftige Auswertung und Weiterverarbeitung der Ergebnisse vornehmen zu können. Dateitransfermechanismen werden auch für die Verteilung von Konfigurations- und Steuerungsdateien benötigt, die von vielen Tools und Bibliotheken gebraucht werden. Benutzer haben in Grid Umgebungen mit zwei wichtigen Aspekten des Dateimanagements zu tun: Welches Protokoll für den Dateitransfer wird unterstützt (z. B. ftp oder scp) und welche Mechanismen müssen verwendet werden, um den Transfer zu initiieren.

2.2. SSH

Secure Shell (ssh) [9] wurde ursprünglich als ein sicherer Ersatz für die (Quasi)standard-UNIX-Werkzeuge rlogin, rsh und telnet entwickelt. Es gibt diverse Implementierungen des ssh Standards [68, 53, 75] und mittlerweile ist ssh der meist benutzte Zugangsmechanismus für Ressourcen im Bereich des Höchstleistungsrechnens. Es ermöglicht die Ausführung von Prozessen auf entfernten Rechnern mittels sicherer, verschlüsselter Kommunikation zwischen zwei als nicht vertrauenswürdig eingestuften Rechnern über ein unsicheres Netzwerk (siehe Bild 2.1). Auch binäre Daten, wie etwa zu kopierende Dateien oder X11-Verbindungen, können auf diese Weise sicher zwischen dem eigenen und anderen Rechnern ausgetauscht werden. Dabei wird nicht nur der Datentransfer, sondern auch das Passwort verschlüsselt.

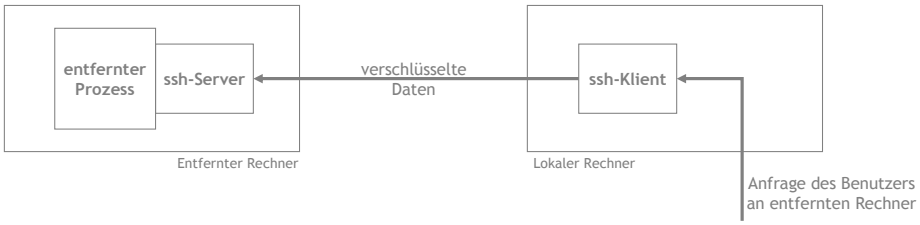

Abbildung 2.1.: Schematische Darstellung der Arbeitsweise von ssh

Ssh in der Version 2 bietet verschiedene Methoden an, mit denen sich der Anwender gegenüber dem entfernten Rechner identifizieren kann: per Passwort, PKI, Zertifikate, Smartcards oder PAM. Während des Aufbaus der sicheren Verbindung müssen sich sowohl der Klient als auch der Server gegenseitig authentifizieren. Wenn die sichere Verbindung einmal aufgebaut wurde, können mehrere Kanäle über diese Verbindung geöffnet werden. Zurzeit gibt es mit scp und sftp zwei verschiedene Möglichkeiten Dateien per ssh zu transportieren. Scp ist eine sichere Version des UNIX Kommandos rcp und sftp ist ein Ersatz für ftp. Eine Gemeinsamkeit dieser beiden Programme ist, dass die zu übertragenden Daten vorher verschlüsselt werden, was zu einer Reduzierung der Leistung bei der Dateiübertragung führen kann. In Umgebungen, die eine hohe Latenz bei gleichzeitig niedriger Bandbreite aufweisen, kann der Benutzer eine optionale Kompression der Daten vornehmen.

In Tabelle 2.1 sind die Vergleichskriterien (siehe Kapitel 2.1) für die Verwendung von ssh in Grid ähnlichen Strukturen zusammengestellt.

Der Benutzer muss sich bei der Verwendung von Ressourcen, die mit ssh zugänglich

AUTHENTIFIZIERUNG UND AUTORISIERUNG	AUFFINDEN VON RESSOURCEN	JOB SUBMISSION	JOB ÜBERWACHUNG	DATEI ÜBERTRAGUNG
Passwort, PKI, Zertifikate etc.	-	-	per Hand über stdout/stderr	scp, sftp

Tabelle 2.1.: Eigenschaften von SSHv2

sind, zuerst auf dem entfernten Rechner per Passwort oder Zertifikat anmelden. Danach öffnet sich ein Eingabefenster (Shell) in dem die Kommandos auf dem entfernten Rechner abgesetzt und Rückgabemeldungen empfangen werden können. Bei einer verteilten Anwendung kann dieses Verfahren sehr aufwendig sein, da die Prozedur für jeden Rechner einzeln ausgeführt werden muss. Falls es sich bei den Jobs um langlaufende Anwendungen handelt, was z. B. bei vielen wissenschaftlichen Simulationen die Regel ist, muss sich der Benutzer nach Beendigung des Jobs wieder einzeln auf den Rechnern anmelden, um die Ergebnisse zu kontrollieren oder weiter zu verarbeiten. Die Dateitransfers müssen gesondert gestartet werden. Bei der Überwachung von laufenden Jobs kann es außerdem aufgrund der vielen geöffneten Fenster mitunter schwierig werden, den Überblick zu behalten. Um eine gewisse Automation der Abläufe zu erreichen, verwenden erfahrene Benutzer in der Regel Skripte. Das erfordert allerdings die genaue Kenntnis der verwendeten Ressourcen und der Anwendung.

2.3. Globus Toolkit (Version 2.0)

Globus [27] ist eine Open Source Initiative mit dem Ziel, eine Standard Grid Architektur für verteilte Ressourcen zu entwickeln. In den vergangenen Jahren hat sich dabei das Globus Toolkit (GT) als weltweiter *de facto* Standard für Grid Middleware herausgebildet, um Anwendungsprogramme (Globus)Grid-fähig zu machen. Es stellt über Bibliotheken Schnittstellen für C- und FORTRAN-Programmierer bereit, um Programme innerhalb eines Globus Grids verwenden zu können. Das Globus Toolkit enthält zusätzlich zahlreiche Kommandozeilen-Programme (überwiegend Skripte), um ein Globus Grid zu verwalten und ohne Änderungen an bestehenden Programmen sofort benutzen zu können. Die Anwendungen können dabei die Globus Grid Services nutzen, ohne an ein bestimmtes Programmiermodell gebunden zu sein. Eine wichtige Eigenschaft eines Globus Grids sind viele und genaue Informationen („*informationrich environment*“) über die verfügbaren Ressourcen. Für die gemeinsame Notation wurde die Beschreibungssprache *RSL (Resource Specification Language)* [44] entwickelt, um Ressourcen zu beschreiben und Jobs zu konfigurieren.

Globus lässt sich am besten mittels Sanduhr-Modell (siehe Abbildung 2.2) beschreiben, das aus zueinander komplementären Modulen besteht. Am Boden dieser Sanduhr befin-

den sich die lokalen Services. Sie werden meistens vom Betriebssystem zur Verfügung gestellt und ermöglichen ressourcenspezifische Operationen.

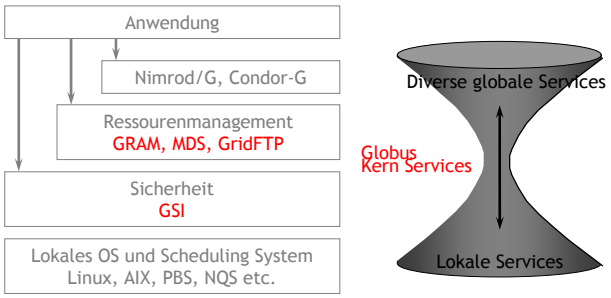

Abbildung 2.2.: Globus Sanduhr Modell

Die in Globus enthaltenen Kernservices befinden sich im Hals der Sanduhr und stellen Schnittstellen zwischen den geografischen und administrativen Gebieten dar, um den Zugang zu den darunter liegenden Zielressourcen zu ermöglichen. Die höher liegenden (komplexen) Services bauen dann auf diesen auf. Die Kernservices sollen möglichst einfach sein, um die Entwicklung von Anwendungen zu erleichtern. Deshalb sind die vier Schlüsselelemente des Globus Toolkit an diese Kernservices gebunden:

- Die *Grid Security Infrastructure (GSI)* stellt die Sicherheitsinfrastruktur sowohl für Verbindungen als auch für die Ressourcen zur Verfügung. Das GT stellt einen auf Delegation basierten Authentifizierungsmechanismus zur Verfügung, der in die GSI eingebettet ist. Die GSI kümmert sich sowohl um die gegenseitige Authentifizierung zwischen den lokalen und entfernten Rechnern als auch um die Autorisierung auf den entfernten Rechnern. Die gesamte Globus Sicherheitsinfrastruktur basiert auf X.509 Zertifikaten. Diese Zertifikate werden benutzt, um temporäre Zertifikate - so genannte Proxyzertifikate zu erzeugen - die dann im Namen des Benutzers verwendet werden. Da die gesamte Sicherheitsinfrastruktur auf credential delegation (Übertragung der Benutzungsrechte) basiert, ist der dynamische und transparente Einbau von neuen Kapazitäten durch verteilte Services möglich. Das bedeutet, einer Ressource wird der Zugang zu weiteren Ressourcen unter Verwendung der temporären Benutzungsrechten ermöglicht, genauso als würde sie diesen Zugang direkt anfordern.
- Das *Grid Resource and Allocation Management (GRAM)* ist für die Ressourcenanforderung und das Prozess-Management zuständig. Deshalb ist dieser Dienst eng an das lokale Batchsystem einer Rechenressource gebunden. Innerhalb eines Globus Grids existieren in der Regel mehrere GRAMS, die die lokalen Ressourcen verwalten.

Die GRAM Schnittstelle zum Batchsystem ist der so genannte Job Manager. Dieser kann z. B. benutzt werden, um Informationen zum Status von Jobs abzufragen.

- Der *Monitoring and Discovery Service (MDS)* ist ein auf LDAP [47] basierender Grid Informationsdienst (GIS), der es erlaubt Globus Grid Ressourcen mittels eines hierarchischen Ansatzes aufzufinden und passende auszuwählen. Der MDS stellt eine Standardschnittstelle für verschiedenste Informationsservices und -quellen dar. Er stellt Mechanismen für die Verbreitung und den Zugang sowohl zu statischen als auch dynamischen Systemen und Anwendungen innerhalb eines Globus Grids zur Verfügung. Es gibt zwei Arten von Informationsservern: den Grid Resource Information Service (GRIS) und den Grid Index Information Service (GIIS). GRIS sammelt Informationen einer Ressource, die er z. B. von dem dieser Ressource zugeordneten GRAM erhält. Mehrere GRIS's können in einem GIIS vereinigt werden, um z. B. die Ressourcen einer VO zu repräsentieren und eine effiziente Abfrage zu ermöglichen. Um die Ausfallsicherheit zu erhöhen, existieren meistens so genannte *S*hadow GIIS.
- *GridFTP* wird für das Datenmanagement verwendet. Es handelt sich dabei um ein sicheres auf ftp basierendes Protokoll, dass im Rahmen des Globus Projektes erweitert wurde, um mehrere Datenströme gleichzeitig benutzen zu können, damit der Durchsatz beim Dateitransfer steigt. Mit GridFTP kann man auch Datenströme nicht Grid basierter Speichersysteme kontrollieren, damit Dateien direkt zwischen Servern transferiert werden können. Speziell für Daten-Grids wurden weitere Datenverwaltungsservices und -protokolle entwickelt: Globus Replica Catalog und Globus Replica Management als Schnittstellen zu Replikationsmechanismen sowie Globus Access to Secondary Storage (GASS) für den Dateizugriff.

Neben diesen vier wichtigsten Services findet man in einem Globus Grid in der Regel noch die oben schon erwähnten komplexen Services: z. B. für Kommunikation (Nexus), zur Systemüberwachung (Heart Beat Monitor HBM), für Netzwerk Monitorung (Gloperf). Für übergeordnete (Vorab-)Reservierung und Scheduling von Ressourcen zwischen mehreren Rechnern wurden diverse Globus Grid Broker entwickelt: Nimrod/G [17] und Condor-G [35]. Über den Verteilerdienst DUROC (Dynamically Updated Request Online Coallocator) wird die Interaktion dieser Broker mit GRAM und MDS ermöglicht.

Mit MPICH-G2 [54] gibt es eine MPI-Implementierung mit der MPI-parallelisierte Programme ohne größere Änderungen in einer Globus Grid Umgebung verteilt ausgeführt werden können. Dabei handelt es sich um eine spezielle Version von MPICH [25], die einzelne Globus Services nutzt um z. B. Jobs zu starten oder Datenkonvertierungen vorzunehmen.

Mit Hilfe der so genannten Globus Commodity Kits können Benutzer auch direkt Grid-fähige Anwendungen schreiben. Über die Commodity APIs können Entwickler von

Java und Python Programmen die vorhandenen Globus Grid Ressourcen direkt in ihre Anwendungen integrieren. Basierend auf dem Java CoG Kit [56] wurde z. B. das Grid Portal Development Kit (GPDK) [67] geschaffen, mit dessen Hilfe Webbrowser basierte Portale für Grid Anwendungen implementiert werden können.

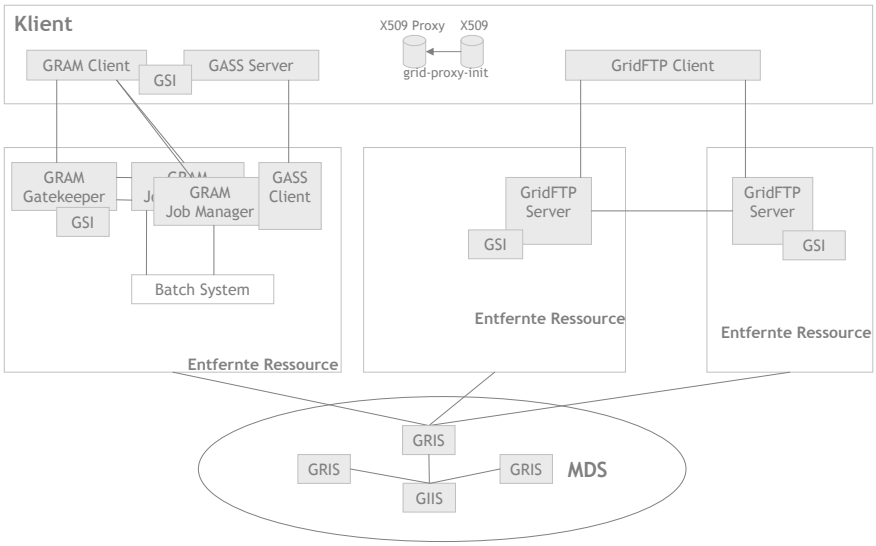

Abbildung 2.3.: Beispielarchitektur eines Globus Grids

Abbildung 2.3 gibt einen konzeptionellen Überblick einer Globus Grid Architektur wieder. Da die Globus Services relativ unabhängig voneinander auf jedem Knoten innerhalb eines Globus Grids verwendet werden können, ergeben sich eine große Anzahl verschiedener Anwendungskombinationen.

Um einen Job innerhalb der Globus Umgebung zu submittieren, kann der Benutzer mittels Kommandozeilenschnittstelle auf die Kernservices und einige Komponenten auf der Benutzer-Ebene zurückgreifen. Zur Spezifikation der Ressourcenanforderungen wird die o. g. Spezifikationssprache *RSL* benutzt. GRAM sorgt dann beispielsweise für die Abwicklung der Job Aufträge (Verschicken und Starten von Jobs, Verbindung zu schon existierenden Jobs, Abbrechen von Jobs). Das schließt auch die Durchführung der gegenseitigen Authentifizierung und die Abbildung der temporären Benutzer-Zertifikate auf Benutzernamen, so wie sie auf dem Zielsystem existieren, ein. Diese Arbeit wird von einem speziellen GRAM Service, dem so genannten *Gatekeeper* erledigt.

Der Klient verbindet sich zuerst mit dem Gatekeeper, um die Erstellung eines Benutzerauftrages vom Job Manager zu initiieren. Danach kommunizieren der Klient und der Jobmanager um den Job zu verwalten und Ausgaben des Jobs an den Klienten zurück

AUTHENTIFIZIERUNG UND AUTORISIERUNG	AUFFINDEN VON RESSOURCEN	JOB SUBMISSION	JOB ÜBERWACHUNG	DATEI ÜBERTRAGUNG
Credential delegation/ X509 Zertifikate	MDS	GRAM mit RSL	GRAM	GASS/GridFTP

Tabelle 2.2.: Eigenschaften von Globus

zu leiten. Ab Version GT2.2 wird der Transfer von Eingabe/Ausgabedaten automatisch erledigt. Der Gatekeeper startet einen GASS-Server, um das Management von Benutzerdateien zu ermöglichen. GASS kommt zum Einsatz, um Ausführungsdateien auf den Zielsystemen bereit zu stellen oder um Eingabe/Ausgabedateien zu transferieren. Dateien können damit transparent im Globus Grid gespeichert werden, ohne dass der Benutzer ihren genauen Ablageort kennen muss. Diese Eigenschaft ist wichtig, um das temporäre Speichern und Wiederauffinden von Dateien zu ermöglichen, die für die Ausführung von Jobs benötigt werden. Über die *gsissh* (eine in Globus erweiterte Version von openssh [68]) kann sich der Benutzer außerdem passwortfrei in die verschiedensten Rechner des Globus Grids interaktiv einloggen - damit ist *S*ingle-Sign-On (einmaliges, zentrales Anmelden) möglich.

In Tabelle 2.2 sind die wichtigsten Vergleichskriterien (siehe Kapitel 2.1) für die Globus Grid Umgebung noch einmal zusammengestellt.

2.4. UNICORE

UNICORE (UNiform Interface to COmputing Resources) [84] wurde Ende der 90er Jahre in Zusammenarbeit mit deutschen Rechenzentren als Desktopumgebung entwickelt, die den Zugang zu Hochleistungsrechnern erleichtern soll. Man kann es als verteiltes Hochleistungsrechen-System bezeichnen, da ein UNICORE Workflow die Ausführung von mehrfachen, miteinander verbundenen Jobs, die auf viele Hochleistungsrechner verteilt sind, beschreibt. UNICORE ist nicht für Daten-intensives Grid Computing konzipiert worden. Entsprechend seiner ursprünglichen Spezifizierung erfüllt es die Mehrzahl der Benutzeranforderungen an Grid Umgebungen [61]. Im Gegensatz zu Globus, das mehr ein Entwickler-orientiertes System darstellt, ist UNICORE ein Benutzer-orientiertes System.

Die UNICORE Architektur (siehe Bild 2.4) ist dreistufig aufgebaut, wobei der UNICORE Klient die oberste, der Gateway und NJS die mittlere Ebene und der TSI die unterste Ebene bilden. Im Folgenden sind die einzelnen Komponenten und ihre Aufgaben detailliert dargestellt:

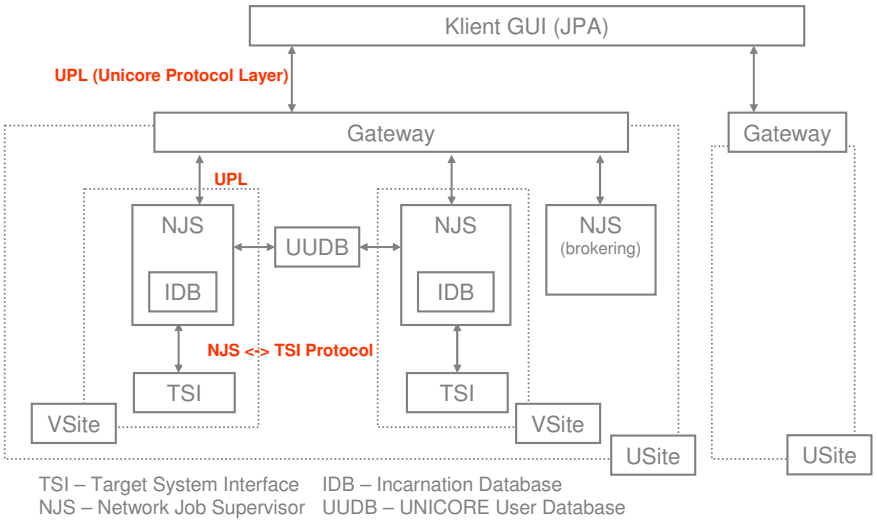

Abbildung 2.4.: Architektur von UNICORE

- Der Klient mit dem *Job Preparation Agent (JPA)* ist die am meisten beachtete Komponente von UNICORE und dient zur Definition und Überwachung von verteilten Aufträgen innerhalb des UNICORE Grids. Er erlaubt die komfortable Definition (auch off-line) von komplexen verteilten Arbeitsabläufen, in denen die Ressourcen und Datenbestände mehrerer Zentren beteiligt sein können. Diese in Form von so genannten *AJO*s (Abstract Job Objects) definierten und mittels Benutzerzertifikat signierten Jobs werden mit einer SSL-Verbindung auf Basis des UPL (Unicore Protocol Layer) Protokolls verschlüsselt an das UNICORE System verschickt.
- *Gateway* ist eine Komponente der Sicherheitsinfrastruktur, die dazu dient, die Authentifizierung der vom Benutzer verschickten Jobs sicherzustellen. Er fungiert als eine Art „Filter“, der sich problemlos in bestehende Firewallkonfigurationen eines Rechenzentrums integrieren lässt. Jobs der Benutzer können den Gateway nur passieren, wenn diese mit einem Zertifikat signiert wurden, das von einer vom Gateway anerkannten CA (Certification Authority) ausgestellt wurde.
- *Unicore User Database (UUDB)* Diese Komponente speichert für jede Ressource die Nutzer mit Zugriffsberechtigung und bildet das Benutzerzertifikat auf Unix Benutzer-Accounts ab.
- *Network Job Supervisor (NJS)* Der Network Job Supervisor kontrolliert die Ausführung der definierten Arbeitsabläufe und die Umsetzung der als AJOs formulierten Aufträge in spezifische, ausführbare Aufträge auf den jeweiligen Ressourcen.

Er fungiert auch als Autorisierungskomponente der Sicherheitsinfrastruktur, da er mit Hilfe der UUDB überprüft, welche Berechtigungen ein vom Benutzer erstellter Job hat.

- *Target System Interface (TSI)* Diese Komponente ist in Perl implementiert und bildet die Schnittstelle zwischen dem NJS und dem Computersystem, auf dem ein Auftrag ausgeführt werden soll, z. B. die Schnittstelle zu einem Batchsystem. Der TSI empfängt direkt die vom NJS aufbereiteten Kommandos und startet sie auf dem Zielsystem als Prozesse, die im Namen des Benutzers ausgeführt werden. Deswegen muss der TSI selbst als root-Prozess auf dem jeweiligen System laufen. Um die Datenintegrität zur Ausführungszeit des Jobs zu gewährleisten, wird für jeden Job auf dem Zielsystem ein so genannter Uspace (User Space) als Arbeitsbereich eingerichtet. Der TSI ist an das jeweilige Scheduling-System gekoppelt und damit in der Lage, Informationen über den Status der Jobs zu sammeln und an den NJS weiterzuleiten und Datenübertragungen durchzuführen. Für die Kommunikation mit dem NJS wird das NJS-TSI-Protokoll verwendet.
- *Incarnation Database (IDB)* Jede in einem UNICORE Grid verwendete Rechnerressource wird in einer IDB Datei beschrieben und diese Beschreibung wird vom NJS für die Umsetzung der allgemeinen Auftragsbeschreibungen in systemspezifische benutzt.

Jeder administrativ eigenständige Bereich wie z. B. ein Rechenzentrum wird als *USite* (UNICORE Site) bezeichnet und normalerweise durch einen Gateway geschützt. Eine USite kann mehrere so genannte *VSites* (Virtual Sites) enthalten, die aus einem NJS und den auf den eigentlichen Zielsystemen installierten TSI(s) bestehen.

Die vertikale Architektur des Systems ist zugleich Stärke und Schwäche des Systems. Sie ermöglicht ein starkes Sicherheitsmodell. Allerdings macht es die vertikale Architektur und das Design der Schnittstellen relativ schwierig, das System zu erweitern oder zusätzliche Services dafür zur Verfügung zu stellen. Damit wird zurzeit auch der Punkt aus der Definition von Grid, die nach der Konzeption von UNICORE entstanden, beinhaltet das „zur Verfügung stellen von neuen Kapazitäten in dynamischer und transparenter Weise durch verteilte Services“. Momentan wird dieser Punkt von UNICORE nicht voll unterstützt. Das integrierte Wesen der Komponenten, einschließlich des Klienten, ermöglicht eine problemlose Installation sowohl für Systemadministratoren als auch Anwender. In diesem Zusammenhang ist auch von Vorteil, dass das System in Java implementiert wurde.

Das Sicherheitsmodell von UNICORE basiert auf X.509 Zertifikaten, die dazu benutzt werden, einen Job zu signieren bevor er an das eigentliche Zielsystem übergeben wird. UNICORE verwendet keine von einem Proxy erzeugten Identitäten und die in UNICORE benutzten privaten Schlüssel verbleiben nur auf dem Klient. Damit könnte man meinen, das das Sicherheitsmodell von UNICORE besser ist, als das von Globus. Der

entscheidende Fakt ist aber, dass UNICORE, wie auch Globus, entfernten Benutzern erlaubt, Programme auf Ressourcen auszuführen. Wie schon in [77] diskutiert wurde, lautet daher die Empfehlung: kritische Ressourcen direkt vor Angriffen zu schützen und sich potentiellen Sicherheitsrisiken ständig bewusst zu sein. Deshalb kann man die per UNICORE zugängigen Rechner so administrieren, dass es nicht möglich ist, vom Benutzer per UNICORE selbst erstellte Skripte direkt auszuführen, sondern nur die in der IDB definierten Softwareressourcen zu verwenden. Das führt zu einer zusätzlichen Abstraktionsschicht, die auch die Sicherheit des Gesamtsystems verbessert. Die Virtualisierung der Softwareressourcen soll durch die Integration von OGSA basierten Services im Rahmen des UniGrids Projektes [73] weiter verbessert werden.

Die Integration bestehender Anwendungen in das UNICORE System erfolgt durch eine speziell entworfene Plugin-Schnittstelle, die das dynamische Laden von Softwaremodulen im Klienten zur Laufzeit ermöglicht. Diese Schnittstelle erlaubt die Benutzung der Basisfunktionalitäten des Klienten wie z. B. Authentifizierung, Datentransfer etc. und stellt zusätzliche Methoden zur Verfügung, die das Vorhandensein von Software überprüfen oder die Anwendung von externen Werkzeugen wie z. B. Bildbetrachter gestatten. Die Plugins können die Eingabedaten der Benutzer automatisch generieren und überprüfen, die Anwendungen auf ausgewählten Rechnern aufrufen und die Ausführung dieser Jobs überwachen. Das Konzept der Plugin-Schnittstelle wurde auch in den NJS übernommen, so dass ein Administrator kommerzielle Anwendungen ohne Modifizierung in das UNICORE System integrieren kann. Der Anwendungsbenutzer muss sich damit nicht mehr um programm- oder systemspezifische Konfigurationen kümmern. Zurzeit existieren u. a. Plugins für folgende Anwendungen: CPMD, Gaussian, Nastran, Fluent und StarCD. Eine direkte Unterstützung für die Ausführung von verteilten MPI-parallelisierte Anwendungen wird nicht angeboten.

Authentifizierung und Autorisierung	Auffinden von Ressourcen	Job Submission	Job Überwachung	Datei Übertragung
Signieren von Jobs/ X.509 Zertifikate	-/Resource Broker Plug-in	JPA mit AJO	NJS	AJO/FTM und GridFTP

Tabelle 2.3.: Eigenschaften von UNICORE

In Tabelle 2.3 sind die Vergleichskriterien (siehe Kapitel 2.1) für die UNICORE Umgebung zusammengefasst. Um einen Job an das Batchsystem eines Rechners zu übergeben, muss der Benutzer den gesamten Ablauf vorher mit Hilfe des Klienten konstruieren. Dafür werden in der grafischen Benutzeroberfläche verschiedene Jobkonstrukte wie Tasks, Sub-Jobs, Schleifen u. ä. angeboten. Zusätzlich kann der Benutzer spezifizieren, welche Daten wann benötigt werden. UNICORE generiert daraus automatisch Transportbefehle. Nach der Konstruktion seines Jobs wählt der Benutzer unter den angebotenen Zielsystemen das passende aus, macht evtl. Angaben zu benötigten Ressourcen (CPU,

Speicher etc.) und submittiert seinen Job. Der Klient wandelt die Benutzervorgaben in das UNICORE spezifische AJO-Format um, signiert den Job mit dem Benutzerzertifikat und versendet ihn. Das UNICORE System stellt keine dynamischen Informationen über den Status und die Ergebnisse des gerade laufenden Jobs zur Verfügung. Aber diese Information kann vom Benutzer durch den Klienten über direkte Anfrage beim jeweiligen NJS abgerufen werden. Die Integration eines dynamischen Informationsservices wäre wünschenswert und kann dazu beitragen die Entwicklung eines ausgereiften Resource Brokers, wie sie im Eurogrid Projekt [58] schon begonnen wurde, voranzutreiben. Nach Beendigung seines Job werden die Ergebnisse automatisch an vorher spezifizierte Speicherorte transferiert.

2.5. TME/ITBL

TME [50] steht für „Task Mapping Editor“ und stellt eine visuelle Programmierumgebung dar, die bei JAERI (Japan Atomic Energy Research Institute) entwickelt wurde. Es ermöglicht die einfache Integration von parallelen Programmen als eine Art „Metaprogramm“ in eine verteilte Umgebung. Die Anwender müssen dafür lediglich in einem grafischen Benutzerinterface die Zusammenhänge zwischen einzelnen Programmbausteinen und die zu verwendenden Rechnerressourcen spezifizieren. Die Ausführung des „Metaprogramms“ wird dann basierend auf dieser Spezifikation automatisch gesteuert.

Die Kommunikations- und Sicherheitsinfrastruktur von TME basiert auf dem ITBL (Information Technology Based Laboratory) System [78]. ITBL ist aus dem STA (Seamless Thinking Aid) Projekt entstanden und unterstützt die Kommunikation zwischen heterogenen Rechnern. In ITBL sind diverse Werkzeuge integriert, die der Benutzer innerhalb der Umgebung verwenden kann, als würde er an einem einzelnen Rechner arbeiten. In Abbildung 2.5 ist der Aufbau des ITBL Systems anhand verschiedener Schichten dargestellt. Um den sicheren und einfachen Zugang zu den Ressourcen zu gewährleisten, wurde ein Single-Sign-On Mechanismus und eine Kommunikationsinfrastruktur entwickelt, die auf einem VPN-Netzwerk basiert. Diese Kommunikationsinfrastruktur erlaubt es auch zwischen verschiedenen Protokollen zu wechseln. Die Grundlage dieser Infrastruktur bildet die Starpc [79] Bibliothek, eine RPC basierende Kommmunikationsbibliothek für Cluster von Parallelrechnern. Diese Bibliothek unterstützt die Kommunikation sowohl zwischen Java Applets und C Programmen als auch zwischen C Programmen selbst. Um verschiedene Netzprotokolle zu unterstützen wurde die Nexus Bibliothek [29] eingebunden.

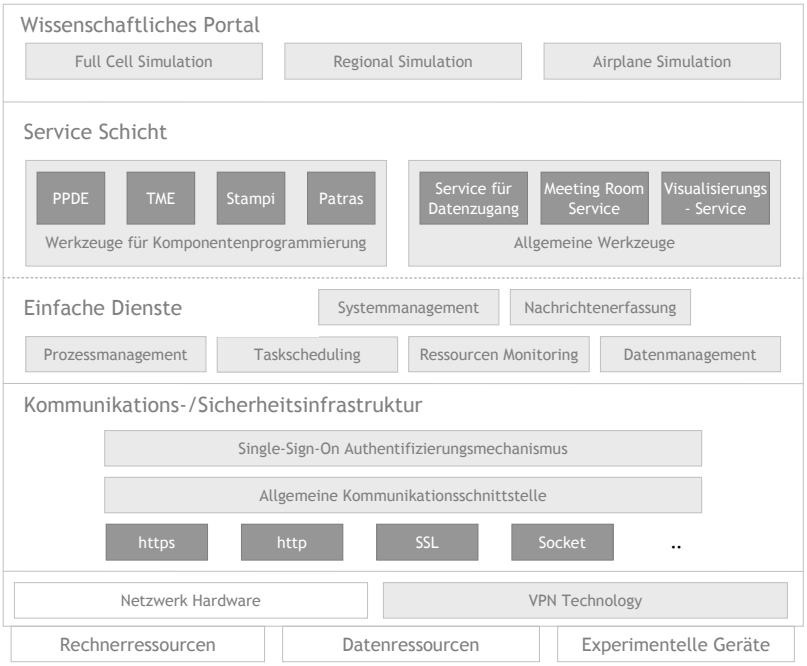

Abbildung 2.5.: Schichtenmodell des ITBL Systems

Die Sicherheitsinfrastruktur von ITBL beruht auf drei Hauptbestandteilen: https und SSL basierte sichere Verbindungen, der Einsatz von X.509 Zertifikaten und die Integration in bestehende Firewalls. Ein spezieller ITBL Server, der innerhalb einer DMZ (Demilitarisierten Zone) installiert werden kann, führt die Authentifizierung der Benutzer mittels Zertifikat durch. Dieser Server ist mit einer Benutzerdatenbank verbunden, auf der sich alle relevanten Informationen befinden, damit sich Benutzer nur einmal am System anmelden müssen, um Zugang zu allen Ressourcen zu erhalten (Single-Sign-On).

Als Basisdienste wurden z. B. Prozessmanagement, Taskscheduler, Ressourcenüberwachung und Datenmanagement implementiert. Zusätzlich gibt es innerhalb des Systems zwei Gruppen von Werkzeugen, die direkt mit dem Benutzer interagieren damit dieser die darunter liegenden Dienste verwenden kann. Das sind zum einen die Werkzeuge zur Komponentenprogrammierung, die für die Entwicklung und die Ausführung von Anwendungen innerhalb des ITBL Systems zum Einsatz kommen. Des Weiteren gibt es die so genannten allgemeinen Werkzeuge, die den Austausch von Informationen und Daten zwischen Forschern erleichtern sollen.

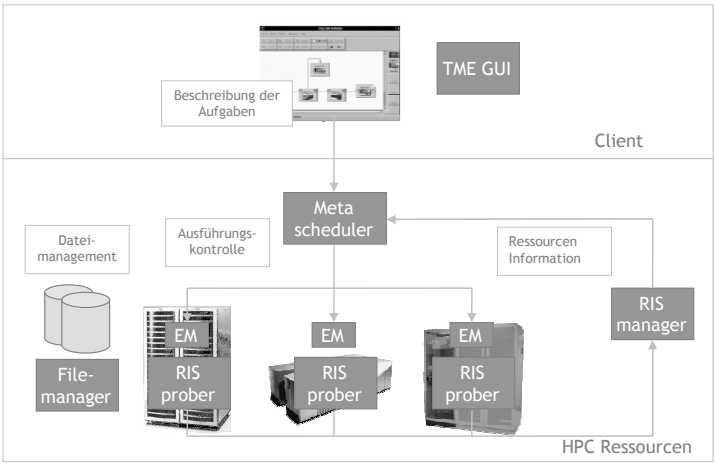

Abbildung 2.6.: Aufbau von TME

Im Folgenden wird der Aufbau von TME (siehe Abbildung 2.6) beschrieben:

- Die *TME-GUI* ist eine webbasierte grafische Schnittstelle, um die komponentenbasierten Metaprogramme zu erstellen. Innerhalb dieser Oberfläche werden Programme und Datenressourcen (Eingabe-/Ausgabedateien) als Icons dargestellt. Die grafische Darstellung des Metaprogramms wird in eine Aufgabenbeschreibung umgewandelt und an den Meta-Scheduler geschickt.
- Der *Meta-Scheduler* hat die Aufgabe festzulegen, wann die einzelnen Teile eines Metaprogramms entsprechend ihrer beschriebenen Abhängigkeiten ausgeführt werden sollen. Er überwacht auch die Ausführung der gestarteten Programme und wählt passende Ressourcen aus, falls dies vom Benutzer gewünscht wird. Der Meta-Scheduler wird direkt auf den Rechnerressourcen installiert.
- Der *Resource Information Monitor (RIM)* sammelt alle Informationen über Rechner und Netzwerke, die in einem ITBL basierten Grid vorhanden sind. Deshalb befinden sich Teile des RIM auf jeder beteiligten Ressource. Diese senden von Zeit zu Zeit ihre gesammelten Daten an den RIM Klienten, welcher die Informationen speichert und an den Meta-Scheduler oder die Benutzer weiterleitet. RIM kann zwei verschiedene Arten von Ressourcen-Informationen verarbeiten. Zum einen sind das statische Informationen wie CPU-Leistung, Speichergröße oder Beschreibungen der Hardware-Architektur eines Rechners. Des Weiteren können dynamische Informationen wie Latenz und Durchsatz des Netzwerkes, Auslastung von Rechnern oder Anzahl der Jobs in einer Batchqueue überwacht werden. Außerdem ist RIM in der Lage, Informationen über das zukünftige Verhalten der Ressource anhand der gesammelten Daten im Voraus zu berechnen.

- Ein *Execution Manager* befindet sich jeweils auf den Rechnerressourcen. Gemeinsam mit dem Meta-Scheduler steuert er die Ausführung der „Metaprogramme“, d. h. er führt Eingabedaten zusammen, startet die einzelnen Programme, überwacht Programmabbrüche und transferiert Ausgabedaten. Falls ein Programm im Batchmodus gestartet werden muss, generiert der Execution Manager die erforderlichen Batchskripte basierend auf Informationen, die der Benutzer in der Spezifizierungssphase angegeben hat.

- Der *File Manager* verwaltet den Dateitransfer von Eingabe/Ausgabedateien.

Innerhalb von TME gibt es eine spezielle Unterstützung für verteilte MPI-parallelisierte Programme. Für die Erzeugung der einzelnen, verteilten Prozesse stellt TME drei verschiedene Mechanismen zur Verfügung, die auf MPI-Semantiken basieren: spawning, client/server Verbindung und MPI-1 basierter Startup. Es gibt außerdem verschiedene Möglichkeiten, um I/O zu realisieren: Dateitransfer, Pipe-Lining und MPI-I/O. Die Erzeugung der verteilten Prozesse wie auch MPI-I/O wird über die Stampi[51] Bibliothek realisiert.

Zurzeit gibt es zwei Anwendungsszenarien, die in TME implementiert wurden: ein Risikoabschätzungssystem für Umweltkatastrophen und ein Datenanalysesystem für die Simulation einer nuklearen Plasmafusion.

AUTHENTIFIZIERUNG UND AUTORISIERUNG	AUFFINDEN VON RESSOURCEN	JOB SUBMISSION	JOB ÜBERWACHUNG	DATEI ÜBERTRAGUNG
Single-Sign-On/ X509 Zertifikate	RIM	Meta-Scheduler/ EM	EM	FM

Tabelle 2.4.: Eigenschaften von TME/ITBL

In Tabelle 2.4 sind die Vergleichskriterien (siehe Kapitel 2.1) für die Benutzung von TME/ITBL zusammengestellt. Um einen Job innerhalb von TME zu submittieren, muss der Benutzer zuerst die erforderlichen Zusammenhänge zwischen den einzelnen Programmbausteinen definieren – d. h. er muss angeben, welche Teile eines parallelen Programms wo ausgeführt werden sollen. Außerdem muss er angeben, wo sich die erforderlichen Eingabedaten befinden und an welche Stelle Ausgabedaten gespeichert werden sollen. Falls gewünscht, kann er mit Hilfe des Meta-Schedulers nach passenden Ressourcen suchen. Des Weiteren kann er die Reihenfolge der Ausführung festlegen und Abhängigkeiten definieren. Nach dem Starten der Jobs können die Benutzer über die TME-GUI mit dem Meta-Scheduler kommunizieren, um die Ausführung ihrer Programme zu überwachen. Der Transfer von Ergebnissen erfolgt automatisiert über den File Manager.

2.6. Avaki/Legion

Avaki ist ein objektorientiertes Grid System dessen Entwicklung auf dem Legion Projekt [46] basiert. Legion wurde seit 1993 an der University of Virginia entwickelt. Seit 2001 wird die Software unter dem Namen Avaki als kommerzielles Grid System vertrieben und weiterentwickelt. Zurzeit existieren drei verschiedene Produktlinien: das „Avaki Data Grid“ welches für datenintensive Anwendungen verwendet wird, das „Avaki Compute Gid“ welches für rechenintensive Aufgaben zur Anwendung kommt und eine Kombination aus beiden - das „Avaki Comprehensive Grid“.

Das Ziel von Avaki ist es, die Komplexität bei der Verwendung der in einem Grid verteilten Ressourcen (Scheduling, Datentransfer, Kommunikation etc.) vor dem Benutzer zu verbergen. Damit der Eindruck eines „globalen virtuellen Rechners“ entsteht, führt Avaki eine spezielle Abstraktionsschicht ein. Diese Schicht befindet sich oberhalb des nativen Dateisystems der in einem Avaki Grid zusammengeführten Ressourcen. Dadurch wird eine Betriebsumgebung geschaffen, die es dem Benutzer z. B. erlaubt, Zugang zu Dateien anhand eines virtuellen Namens in einem virtuellen Verzeichnis zu bekommen. Er braucht nicht zu wissen, wo sich die Datei physikalisch befindet.

Die Avaki Grid Software besteht insgesamt aus drei verschiedenen Software-Schichten (siehe Abbildung 2.7):

- Die *Grid Protokoll* Schicht stellt Adapter zu verschiedenen Netzprotokollen/ Programmiersprachen, die Sicherheitsumgebung und Mechanismen für die Namensgebung zur Verfügung.
- Innerhalb der *System Management* Schicht werden Ausführung von Jobs und Management der verteilten Ressourcen überwacht und verwaltet.
- Die Schicht der *Anwendungsdienste* stellt verschiedene höherwertige Dienste zur Verfügung, die dazu benutzt werden können, um z. B. Dateizugriff, Kollaboration oder Durchsatzrechnen innerhalb eines Avaki Grids zu realisieren.

Ein Avaki Grid ist aus virtuellen, verteilten Objekten aufgebaut. Das bedeutet, dass alle in diesem Grid enthaltenen Ressourcen durch einzelne, aktive Objekte repräsentiert werden, die über einen einheitlichen RMI-Dienst miteinander kommunizieren. Durch diese Vorgehensweise wird die Interoperabilität der Komponenten zwischen verschiedenen Programmiersprachen und Ausführungsplattformen ermöglicht. Physikalisch kann ein Avaki Grid aus einzelnen Rechnern oder Clustern aufgebaut werden. Die Avaki Software kann mit vorhandenen Batchsystemen interagieren indem es virtuelle Queues erzeugt. Diese virtuellen erlauben auch die Zusammenführung dieser Ressourcen, um maximale Leistung zu erzielen.

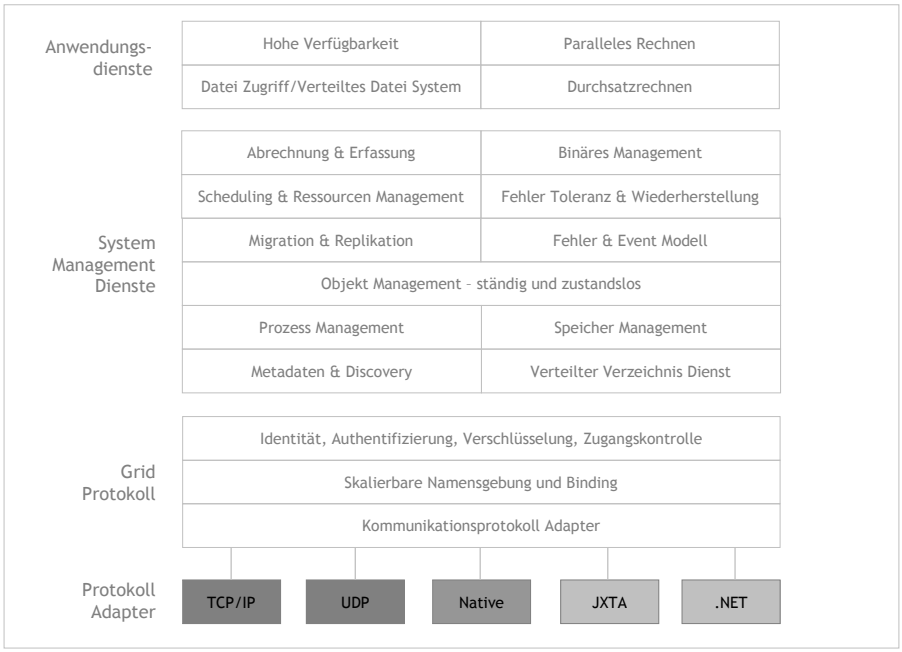

Abbildung 2.7.: Software Schichten in Avaki

Das Sicherheitskonzept von Avaki basiert auf PKI-Authentifizierungsmechanismen. Jede Ressource, die für ein Avaki Grid zur Verfügung gestellt wird, erhält eine eindeutige Kennung, die dazu benutzt wird, die Rechte einzelner Benutzer abzubilden. Das heißt, für jeden Dateizugriff, das Starten von Programmen oder des Einstellen von Jobs in eine Queue wird von Avaki anhand dieser Kennung überprüft, ob ein Benutzer entsprechende Rechte besitzt. Die Konfiguration dieser Zugangskontrolle kann von einem Grid Administrator bis in kleinste Details vorgenommen werden, was das selektierte Bereitstellen von Grid-Ressourcen sehr vereinfacht. Auch Endbenutzer sind damit z. B. in der Lage, ihre Dateien anderen Teilnehmern zur Verfügung zu stellen. Alle Nachrichten, die innerhalb eines Avaki Grids versendet werden, können verschlüsselt werden.

Um sich die verteilte Rechenleistung direkt zu Nutze zu machen, bietet Avaki zwei Formen für die Ausführung von parallelen Programmen an: Zum einen wird die Erstellung von parametrisierten Programmen unterstützt, damit Durchsatzrechnen möglich ist. Allerdings gibt es keine Optionen, um Abhängigkeiten oder Workflows für diese Jobs zu definieren. Zum anderen werden spezielle „native MPI-Wrapper“ angeboten, damit existierende MPI-Programme in einem Avaki Grid ausgeführt werden können. Es gibt zurzeit aber keine Avaki eigene MPI-Implementierung, die verteilte Jobs in einer

heterogenen Umgebung unterstützt.

AUTHENTIFIZIERUNG UND AUTORISIERUNG	AUFFINDEN VON RESSOURCEN	JOB SUBMISSION	JOB ÜBERWACHUNG	FILE TRANSFER
PKI/ Kennung	Infoobjekte	register/run	stdout/stderr	mv

Tabelle 2.5.: Eigenschaften von Avaki

In Tabelle 2.5 sind die Vergleichskriterien (siehe Kapitel 2.1) für die Avaki Umgebung zusammengefasst. Avaki bietet als Benutzerinterface die Kommandozeile an. Damit ein Benutzer seine Anwendungen innerhalb eines Avaki Grids ausführen kann, muss er sie zuerst mit einem speziellen Kommando registrieren (`register`). Danach kann er für diese Anwendung bestimmte Rechte definieren, um sie anderen Gruppen/Benutzern zur Verfügung zu stellen. Nachdem sich ein Benutzer einmal in das Avaki System eingeloggt hat, kann er Jobs innerhalb eines Avaki Grids interaktiv mit dem Befehl `run` ausführen oder in eine der virtuellen Avaki Queues einstellen. Über Infoobjekte kann er sich Informationen über vorhandene Ressourcen (Rechner und Anwendungen) anzeigen lassen und geeignete auswählen. Ein/Ausgabedaten werden automatisch von den verteilten Ressourcen gelesen bzw. geschrieben.

2.7. Symphony

Symphony [62] ist ein in Java implementiertes, komponentenorientiertes System zur Erstellung und Verarbeitung von Jobs für Computational Grids. Es stellt eine Abstraktionsschicht für existierende Grid Middleware Lösungen und Betriebssysteme zur Verfügung, um Anwendungen unabhängig von der vorhandenen Grid-Middleware innerhalb des Grids ausführen zu können. Symphony basiert auf JavaBeans (ein auf Java basierendes Komponentenmodell), die vom Benutzer angepasst und zu einem Metaprogramm zusammengeführt werden müssen. Zurzeit existieren JavaBean-Implementierungen, die es erlauben Programme auf Symphony- und Globus-Ressourcen auszuführen. Das objektorientierte Design der Symphonybeans sollen die einfache Anpassung und Erweiterbarkeit des Systems unterstützen.

Die Sicherheitsinfrastruktur von Symphony basiert auf PKI-Techniken. Das Bild 2.8 zeigt einen konzeptionellen Überblick des Symphony Systems. Es besteht im Wesentlichen aus vier Bestandteilen:

- Die *Metaprogramm Schicht* besteht aus verschiedenen Kernbeans, die die entfernten Ressourcen unabhängig von den darunter liegenden Diensten abstrahieren.
- Die *Anpassungsschicht* besteht aus einem Set von Schnittstellen und Anpassungsdiensten, die es erlauben, darunter liegende Grid Middleware Dienste oder ent-

fernte Betriebssystemen zu verwenden.

- Die eigentliche *Kompositions- und Manipulationsumgebung*, in der sich die von Benutzer hauptsächlich verwendeten Symphonybeans befinden.
- Das *Symphony Backend*, das zur Erzeugung und Kontrolle von Prozessen benutzt wird, die die eigentlichen Befehle auf den entfernten Ressourcen ausführen.

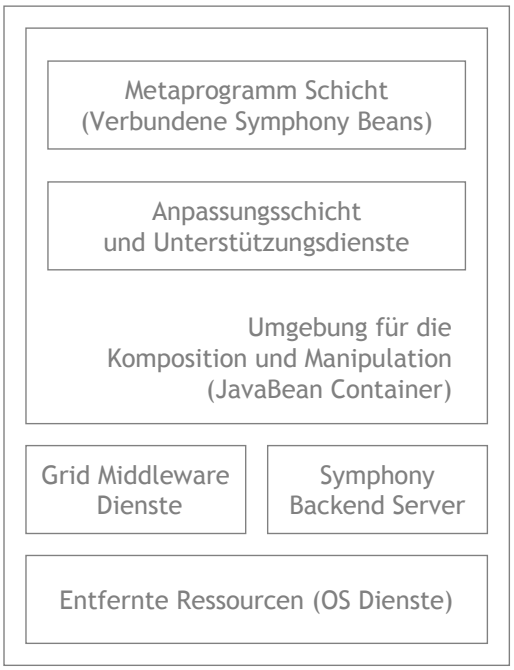

Abbildung 2.8.: Überblick über das Symphony System

Um einen Job als Metaprogramm im Symphony System zu konstruieren, stehen dem Benutzer Programm- und Datei-Beans sowie Beans für die Standardein-/ausgabe und die Fehlerausgabe zur Verfügung. Die Ressourcenbeschreibung sowie die Angaben für das Starten von Anwendungen erfolgt in den Programm-Beans. Diese können statische Ressourcenbeschreibungen aus einer XML-Konfigurationsdatei lesen oder Globus Informationsdienste auswerten. Für die Erzeugung von einfachen Workflows können die Beans mittels „Connections“ verbunden werden. Das Monitoring von laufenden Programmen erfolgt durch die Standardein-/ausgabe-Beans, die Ausgabeströme an die grafische Oberfläche weiterleiten. Für den Dateitransfer wird ftp oder GridFTP benutzt. Es gibt keine spezielle Unterstützung für die Verteilung von MPI-parallelisierten Anwendungen innerhalb des Symphony Systems.

2.8. Zusammenfassung und Bewertung der Projekte

Ssh bietet keinerlei zusätzliche Services für den Endbenutzer, denn das Auffinden von Ressourcen oder die Jobverarbeitung sind natürlich nicht im ssh Protokoll vorgesehen. Es handelt sich also nicht um Grid Middleware im klassischen Sinne. Ssh wird allerdings, wie oben schon erwähnt, sehr oft als direkter Zugangsmechanismus für Rechnerressourcen verwendet und hat damit auch für den Aufbau von Grids eine große Bedeutung.

Aufgrund der weltweiten Verbreitung des Globus Toolkits ist eine große Anwendergruppe vorhanden. Das wiederum hat dazu geführt, dass Globus heute den de-facto Standard für Grid Computing darstellt. Auf Basis der GT Bausteine kann eine komplette Grid Infrastruktur aufgebaut und verwaltet werden kann. Mit Hilfe dieser Services lassen sich dann auch höhere (komplexe) Services, Grid Werkzeuge und Anwendungen entwickeln. Für die Steuerung der Umgebung muss der Benutzer allerdings alle Schritte selbst planen, was z. T. sehr aufwendig und kompliziert sein kann, denn in Globus wird die Heterogenität der Grid Komponenten nicht versteckt sondern muss (auch vom Benutzer) verwaltet werden. Globus ist kein fertiges Produkt, es unterliegt einer kontinuierlichen Weiterentwicklung, da die neuesten Entwicklungen im Bereich des Grid Computing eingearbeitet und bestehende Standards erweitert bzw. integriert werden. Das hat zur Folge, dass der Einsatz als Produktionsplattform nicht immer sinnvoll ist, da die Stabilität der Software aufgrund der ständigen Neuerungen zum Teil nicht gewährleistet ist.

Die UNICORE Software stellt eine gut etablierte Grid Lösung dar, die es den Benutzern erlaubt, komplexe, komponentenorientierte Hochleistungsrechenjobs zu erstellen. Insbesondere werden Benutzer unterstützt, die die Ressourcen im „traditionellen“ Sinne (d. h. Batchjobs für einzelne Ressourcen) verwenden wollen. Zusätzlich wird eine gute Unterstützung für die Erstellung von Workflows einschließlich automatischer Ausführung spezieller Aufgaben und Speicherung bzw. Transfer von Dateien angeboten. Die Stärke von UNICORE liegt in der Abstraktion eines Modells von Batchsystemen zur Jobverarbeitung. Aber es gibt kaum direkte Unterstützung für parallele Anwendungen, die auf mehrere Ressourcen gleichzeitig verteilt ausführbar sind. Aufgrund seines Architekturmodells ist es schwierig, UNICORE um spezielle Services zu erweitern oder momentan nicht unterstützte Funktionen zu implementieren. Diese Schwierigkeiten sollen mit der Anpassung von UNICORE an den OGSA Standard weitestgehend beseitigt werden, ohne die zurzeit vorhandene Stabilität und Produktionsreife des Systems zu gefährden [73]. Hingegen bietet das Modell von Globus sowohl die Möglichkeit der Integration anderer Module als auch deren Erweiterbarkeit.

TME ist hauptsächlich darauf ausgerichtet, die Integration von parallelen Programmen in eine verteilte Umgebung zu vereinfachen. Mit Hilfe der TME-GUI können Benutzer

die Abläufe ihrer Anwendungen zusammenstellen. TME realisiert daraus ein schematisches Meta-Programm, das dann auf einem Cluster aus Hochleistungsrechnern kontrolliert ausgeführt wird. Die Kommunikations- und Sicherheitsinfrastruktur von TME basiert auf der ITBL Umgebung. Dadurch wird zwar z. B. die Unterstützung von verschiedenen Netzprotokollen und die Integration mit diversen Werkzeugen erreicht, aber die weltweite Verbreitung von TME wird eingeschränkt.

Avaki ist eines der wenigen zurzeit kommerziell vertriebenen Grid Systeme. Es stellt Techniken zur Verfügung, die es erlauben Ressourcen zu „sammeln“ und danach über eine einheitliche Umgebung zu nutzen. Dabei ist es möglich, einzelne Teile der Ressourcen sehr feingranular innerhalb des Avaki Grids zur Verfügung zu stellen. Speziell für Datengrids wird eine gute Unterstützung angeboten. Im Ganzen gesehen, bietet Avaki allerdings nicht die vergleichbare Menge an Funktionalitäten wie sie Globus und UNICORE bieten und für die Ausführung von verteilten Jobs in heterogenen Umgebungen wird keine spezielle Unterstützung angeboten.

	SSH	GLOBUS	UNICORE	TME	AVAKI
Authentifizierung und Autorisierung	Passwort, PKI, Zertifikate etc.	Credential delegation/ X509 Zertifikate	Signieren von Jobs/ X509 Zertifikate	Single-Sign-On/ X509 Zertifikate	PKI/ Kennung
Auffinden von Ressourcen	-	MDS	-/Resource Broker plug-in	RIM	Infoobjekte
Job Submission	-	GRAM mit RSL	JPA mit AJO	Meta-Scheduler/EM	register/run
Job Überwachung	stdout/stderr	GRAM	NJS	EM	stdout/stderr
File Transfer	scp, sftp	GASS/ GridFTP	AJO/FTM und GridFTP	FM	mv

Tabelle 2.6.: Vergleich von ssh, Globus, UNICORE, TME, und Avaki

In Tabelle 2.6 sind noch einmal die Vergleichskriterien für alle diskutierten Umgebungen zusammengefasst. Keine der vorab aufgeführten Middlewarelösungen erfüllt zurzeit alle Anforderungen, die ein Benutzer benötigt, um komfortabel und effizient verteilte, parallele wissenschaftliche Anwendungen im Grid auszuführen. Speziell für mit MPI parallelisierte Anwendungen ist die angebotene Unterstützung entweder zu speziell (Globus, TME) oder die Integration von bestehenden Anwendungen für einzelne Benutzer gestaltet sich schwierig (UNICORE, Avaki). Die Unterschiede in den Architekturmodellen und Implementierungen machen es außerdem schwer, wirkliche Interoperabilität zwischen den einzelnen Grid Umgebungen zu ermöglichen. Im GRIP Projekt [66] gibt es Ansätze, diese zumindest zwischen UNICORE und Globus herzustellen. Der Benutzer ist allerdings weitgehend allein gestellt, wenn er seine Anwendung auf Ressourcen ver-

teilen will, die in verschiedene Grid Umgebungen integriert sind. Auch die vorgestellte Multi-Grid Lösung Symphony lässt sich dafür nur bedingt einsetzen, da sie nicht für die Verteilung von MPI-parallelisierten Anwendungen konzipiert wurde.

2.9. Standardisierungsaktivitäten

Die Erstellung und Steuerung von Grid Strukturen aus unzähligen heterogenen Rechner-, Datenbank- und Speicherkomponenten in der Gestalt eines einzigen einheitlichen Computers bedarf einer wohl durchdachten Middleware und stabiler Standard-Schnittstellen, da die Zusammensetzung in der Realität viele verschiedene Unternehmen und Institutionen umfassen kann.

Für die Standardisierung der Protokolle und Funktionen wurde das Global Grid Forum (GGF) [43] eingerichtet. Das GGF versucht, die weltweite Verbreitung von Grid Technologien zu fördern indem Grid Spezifikationen geschaffen werden, die zu akzeptierten Standards und der Entwicklung interoperabler Software führen. Ein weiteres Ziel des GGF ist die Schaffung einer internationalen Gemeinschaft um Ideen, Erfahrungen und Anforderungen auszutauschen.

Die Entwicklung der Grid-Standardisierung der letzten Jahre ist zu einem großen Teil eine Reaktion auf die relativ weite Verbreitung der Webservices, u. a. auch in kommerziellen Bereichen. Dabei geht es um die gemeinsame Nutzung und Interaktion von Diensten auf verschiedenen geographisch verteilten Rechnern. Grid-Dienste lassen sich grundsätzlich als Webservices realisieren. Deswegen entwickelte das Global Grid Forum an die Anforderungen von Grids angepasste Web-Service-Spezifikationen. OASIS [69] betreibt in Partnerschaft mit dem GGF die Entwicklung des gemeinsamen Standards WSRL (Web Services Resource Framework).

Firmen wie Fujitsu-Siemens arbeiten zusammen mit AMD, Cisco, EMC, HP, Oracle, Sun und einigen anderen IT-Schwergewichten sehr intensiv innerhalb der 'Enterprise Grid Alliance' (EGA), um solche Standards voran zu treiben. Inwieweit Grid-Technologien als Standard letztendlich Einzug in die allgemeine wissenschaftliche Arbeit halten, wird abzuwarten sein. Die angekündigte Unterstützung dieser Technik durch eine große Anzahl von Firmen wie u. a. IBM, Sun oder auch Microsoft deuten aber darauf hin, dass in Zukunft durchaus auch mit Produkten zu rechnen ist, die Grid Computing unterstützen.

2.9.1. Global Grid Forum (GGF)

Das Global Grid Forum ist ein offener Zusammenschluss von Anwendern, Entwicklern und kommerziellen Anbietern, die weltweit federführend die Standardisierung

für Grid Computing vorantreiben wollen. Die Mitglieder des GGF stammen aus Forschungseinrichtungen, Universitäten und der Industrie und repräsentieren mehr als 400 Organisationen aus 50 Ländern. Das Global Grid Forum entstand aus einer Serie von Gesprächen, Workshops und speziellen Sitzungen, die sich mit Themen des Grid Computing befassten. Eines der ersten dieser Treffen fand bei der Supercomputing Konferenz 1998 statt und führte zur Entstehung des Grid Forums, einer Gruppe von Grid Entwicklern und Anwendern in den USA, die daran interessiert waren Standards und Verfahren für Grid Computing zu definieren und zu verbreiten. Im Jahr 2000 vereinigte sich das Grid Forum mit dem in Europa ins Leben gerufenen European Grid Forum (eGrid) und dem Asia-Pacific Grid Forum zum Global Grid Forum. Das erste reguläre Global Grid Forum Meeting wurde im März 2001 abgehalten. Seitdem hat das GGF eine Reihe von Standardisierungs- und Spezifikationsdokumenten erarbeitet und mehr als ein Dutzend Treffen in der ganzen Welt abgehalten.

Die Arbeit des GGF wird hauptsächlich von an Standardisierung interessierten Arbeitsgruppen, auf Technologie fokussierten Forschungsgruppen und spezifischen Anwendergruppen ausgeführt. Die Arbeitsgruppen konzentrieren sich jeweils auf eine sehr spezielle Technologie oder Grid Thema und entwickeln Vorgaben, Anleitungen oder Spezifikationen, die zu weit verbreiteten Standards und interoperabler Software führen. Die Arbeit der Forschungsgruppen ist auf wissenschaftliche oder kommerzielle Anwendungen ausgerichtet, die Grids benutzen oder sie führen Forschungsarbeiten auf einzelnen Aspekten von Grid Technologien oder Lösungen durch. Die Anwendergruppen repräsentieren jeweils spezifische Benutzergruppen für Grid Anwendungen. Jede Gruppe gehört zu einem bestimmten Fachbereich (Area), der von einem Bereichsleiter (Area Director) koordiniert wird. Tabelle 2.7 listet die derzeit im Global Grid Forum existierenden Fachbereiche auf.

Die Koordinierung des GGF erfolgt durch einen Steuerungsauschuss, die Grid Forum Steering Group (GFSG). Die GFSG wird von den Mitgliedern gewählt und dieses Management unterstützt die Erstellung der Standards und die Aktivitäten innerhalb des GGF. GFSG besteht aus dem GGF Chair, einem oder mehreren GGF Vice-Chairs und den Bereichsleitern. Diese Bereichsleiter leiten die Arbeits- und Forschungsgruppen, die die eigentliche Kernarbeit des GGF ausführen. Der Wirkungsbereich und die zeitliche Planung der Arbeit einer Gruppe werden in der Charta einer Gruppe festgelegt. Dieses Charta Dokument stellt eine Art Vertrag mit der Grid Forum Steering Group (GFSG) dar.

Data	Data Access and Integration Services (DAIS-WG) Grid File Systems (GFS-WG) Data Format Description Language (DFDL-WG) GridFTP-WG Grid Storage Management (GSM-WG) Information Dissemination (INFOD-WG) Transaction Management (TM-RG) OGSA Data (OGSA-D-WG) Byte IO (ByteIO-WG)
Compute	Grid Resource Allocation Agreement Protocol (GRAAP-WG) Job Submission Description Language (JSDL-WG) Grid scheduling Architecture (GSA-RG) OGSA Basic Execution Services (OGSA-BES-WG) OGSA Resource Selection Services (OGSA-RSS-WG)
Research Applications	Application Developers and Users (APPS-RG) Astronomy Applications (Astro-RG) Humanities, Arts, and Social Science (HASS-RG) Life Sciences Grid (LSG-RG) Particle and Nuclear Physics Applications (PNPA-RG) Preservation Environments(PE-RG)
Management	Application Contents Service (ACS-WG) Configuration Description, Deployment, and Lifecycle Management (CDDLM-WG) Grid Economic Services Architecture (GESA-WG) OGSA Resource Usage Service (RUS-WG) Usage Record (UR-WG)
Community Affairs	GGF Process-WG Grid Benchmarking (GB-RG)
Grid Operations	CA Ops (CAOPs-WG) Production Grid Services (PGS-RG)
Technology Innovators	Advanced Collaborative Environments (ACE-RG) Appliance Aggregation (APPAGG-RG) Grid Computing Environments (GCE-RG) User Program Development Tools for the Grid (UPDT-RG) Semantic Grid (SEM-RG) Workflow Management (WFM-RG)
Applications	Grid Remote Procedure Call (GridRPC-WG) Grid Information Retrieval (GIR-WG) Distributed Resource Management Application API (DRMAA-WG) Simple API for Grid Applications (SAGA-RG) Grid Checkpoint Recovery (GridCPR-WG)
Industry Applications	Enterprise Grids Requirements (EGR-RG) Telecomm Community Group (Telco-CG)
Architecture	Open Grid Services Architecture (OGSA-WG) OGSA Naming (OGSA-Naming-WG)
Infrastructure	IPv6 (IPv6-WG) Network Measurement (NM-WG) Grid High-Performance Networking (GHPN-RG) Network Measurements for Applications (NMA-RG)

Security	Open Grid Service Architecture Authorization (OGSA AUTHZ-WG) Firewall Issues (FI-RG) Trusted Computing (TC-RG)
Liaison	Standards development organizations Collaboration on networked Resources Management Working Group (SCRM-WG)

Tabelle 2.7.: Arbeitsgruppen im Global Grid Forum (Stand Mai 2006)

2.9.2. Organisation for the Advancement of Structured Information Standards (OASIS)

OASIS ist ein internationales Konsortium, dass die Entwicklung, Konvergenz und Verabschiedung von e-Business Standards zum Ziel hat. Das Konsortium bringt hauptsächlich Standards für Webservices hervor. Es werden aber auch Standards für Sicherheitsaspekte im e-Business Bereich und für anwendungsorientierte Märkte geschaffen. Gegründet wurde OASIS 1993 unter dem Namen SGML Open als ein Konsortium von Anbietern und Nutzern, die sich mit der Entwicklung von Richtlinien für Interoperabilität zwischen Produkten unter Verwendung von SGML (Standard Generalized Markup Language) beschäftigten. OASIS hat diesen Namen 1998 geändert, um den erweiterten Fokus der technischen Arbeit innerhalb des Konsortiums widerzuspiegeln. OASIS hat heute mehr als 5000 Mitglieder, die über 600 Organisationen und Einzelmitglieder aus 100 Ländern repräsentieren.

OASIS zeichnet sich durch eine transparente Steuerung und Verfahrensweisen aus. Die Mitglieder legen das technische Programm von OASIS selbst fest. Der Prozess der Standardisierung zielt ausdrücklich darauf ab, einen Konsens für die beteiligten Industrien zu schaffen. Abgeschlossene Arbeiten werden durch offene Abstimmungen ratifiziert. Die Leitungsgremien (Board of Directors und das Technical Advisory Board) sind unbeschränkt in ihrer Arbeit und werden für zwei Jahre gewählt. Die Mitgliedschaft in der Leitung des Konsortiums basiert auf individueller Leistung und ist nicht von der Höhe der finanziellen Zuwendung oder dem Ansehen einzelner Firmen abhängig.

Das Konsortium betreibt zwei der meist genutzten Informationsportale für XML und Webservice Standards. Zurzeit existieren technische Ausschüsse in folgenden Kategorien (Stand Mai 2006):

- Adoption Services
- Computing Management
- Document-Centric Applications
- e-Commerce
- Law & Government

- Localisation
- Security
- Service Oriented Architecture (SOA)
- Standards Adoption
- Supply Chain
- Web Services
- XML Processing

3. Das Konzept und die Implementierung eines Grid Configuration Managers

„Any sufficiently advanced technolgy is indistinguishable from magic“.
Arthur C. Clarke

Parallele Programme, die zur gleichen Zeit verteilt auf verschiedenen Rechnern laufen, stellen einen wichtigen Hintergrund des Grid Konzeptes dar. Die Notwendigkeit für solche, auch als „parallele multi-side Jobs“ bezeichnete, Programme stammt aus Anwendungsbereichen, die einen hohen Bedarf an Rechnerressourcen haben oder um den Durchsatz auf bestimmten Rechnern zu erhöhen bzw. die „turnaround“ Zeiten (Zeit vom Start einer Aufgabe bis zu deren Fertigstellung) zu verkürzen. Wie in Kapitel 1.2 dargestellt wurde, stehen die Benutzer bei der Ausführung dieser Programme im Grid vor der Herausforderung, die z. B. durch die Heterogenität verursachten Probleme, zu bewältigen.

In diesem Kapitel wird die Entwicklung des Grid Configuration Managers (GCM) beschrieben, der die Handhabung der o. g. Jobs auf Basis der Kommunikationsbibliothek PACX-MPI [39] signifikant erleichtert und unterstützt. Das erste Unterkapitel beschreibt detailliert die Ziele, die mit der Implementierung des Grid Configuration Managers verwirklicht werden sollen. Obwohl der Entwurf des GCM nicht nur auf PACX-MPI beschränkt wurde, bietet die jetzige Implementierung besondere Unterstützung für die Konfiguration und das Starten von PACX-MPI-Anwendungen. Deshalb werden in einem Abschnitt das Konzept und die Handhabung von PACX-MPI gesondert dargestellt. Für die Vorhersage des Laufzeitverhaltens von parallelen Anwendungen innerhalb von Grid Umgebungen verwendet der Grid Configuration Manager das Programm Dimemas [41]. Dimemas ist ein Programm zur Vorhersage von Leistungsdaten einer Anwendung, das in Abschnitt 3.3 gesondert vorgestellt wird. Im nachfolgenden Hauptteil dieses Kapitels wird das Konzept des Grid Configuration Managers erläutert und Implementierungsdetails dargestellt. Danach wird die Umsetzung des Grid Configuration Managers als Plugin für die UNICORE Grid Umgebung vorgestellt.

3.1. Ziele des Grid Configuration Managers

Um Benutzer bei der effizienten Durchführung von verteilten Anwendungen in heterogenen Grid Umgebungen zu unterstützen, muss eine intelligente Umgebung geschaffen werden, die die folgenden Punkte erfüllt:

1. Unterstützung bei der **Konfiguration von verteilten Jobs**, d. h. Zusammenstellung von Daten für die Ausführung einer Anwendung in einer heterogenen Grid Umgebung. Bereits erstellte oder verwendete Konfigurationen sollen abgespeichert und/oder exportiert werden können, damit die Wiederverwendbarkeit gewährleistet wird.
2. **Vorhersage von Leistungsdaten** (Laufzeitverhalten) einer Anwendung auf Basis von Tracedaten mit Hilfe von Dimemas. Das Starten und die Konfiguration von Dimemas sollen innerhalb der Umgebung durchgeführt werden können. Die Eingabedaten für die Vorhersage sowie die Ergebnisse sollen in einem gesonderten Metadatenformat abgespeichert werden, damit sie für eine spätere Verwendung zur Verfügung stehen.
3. Unter Beachtung der vom Benutzer vorgeschlagenen Rechnerressourcen, den auf jedem Rechner verfügbaren Knoten und der Kommunikationscharakteristik des Rechners bzw. zwischen den einzelnen Rechnern soll **automatisch festgestellt werden, was die beste (effizienteste) Konfiguration darstellt**. Anhand der Vorhersage kann der Benutzer eine Anpassung der Konfiguration vornehmen.
4. Die für das erfolgreiche **Starten der Jobs** erforderlichen Konfigurationsdateien sollen automatisch erstellt und an alle, an der Konfiguration beteiligten Rechner, verschickt werden. Das Starten der Jobs soll zentral mit vorgegebenen Startkommandos von einer Stelle aus erfolgen. Weiterhin soll es die Möglichkeit geben, Befehle nur an einzelne Rechner der Konfiguration zu schicken um vorbereitende Maßnahmen (Pre-processing) durchzuführen. Damit die Benutzer die für die Anwendungen notwendigen Eingabedateien an die dafür vorgesehenen Orte transferieren können, soll eine zentrale Schnittstelle für den Dateitransfer entwickelt werden.
5. Ausgabe von Rück- und Fehlermeldungen der Anwendung, um eine **Überwachung der Jobs** zu ermöglichen. Diese Meldungen sollen zusammengefasst dargestellt werden, um die Handhabung zu erleichtern.

Außerdem soll der Grid Configuration Manager innerhalb einer heterogenen Grid Umgebung funktionieren und muss deshalb verschiedene Zugangsmechanismen unterstützen. Dabei ist es wichtig, dem Benutzer diese Umgebungen mit der gleichen Ansicht zu präsentieren, so dass die Unterschiede in den einzelnen Umgebungen vor ihm verborgen bleiben.

Die Umsetzung des Konzeptes soll als grafische Oberfläche erfolgen, um eine möglichst einfache und durchdachte Benutzerführung zu ermöglichen.

3.2. Die Kommunikationsbibliothek PACX-MPI

Normalerweise werden parallele Anwendungen für den Einsatz auf traditionellen Parallelrechnern entwickelt. Falls diese Programme für Rechner mit verteiltem Hauptspeicher unter Verwendung des Message Passing Interfaces (MPI) geschrieben worden sind, können sie prinzipiell auch auf verteilte Grid-Umgebungen portiert werden. Im folgenden Abschnitt wird die Kommunikationsbibliothek PACX-MPI vorgestellt, die speziell für diesen Zweck entwickelt wurde.

3.2.1. Konzept von PACX-MPI

PACX-MPI [39] ist eine am Höchstleistungsrechenzentrum Stuttgart (HLRS) entwickelte Implementierung des MPI-Standards mit Optimierungen für Meta- und Grid Computing Umgebungen. Diese Umgebungen zeichnen sich dadurch aus, dass aus Sicht der Kommunikation gesehen, ein großer Unterschied in der Charakteristik der verschiedenen Schichten der Kommunikation hinsichtlich Latenz und Bandbreite besteht. Das heißt, wenn Kommunikation zwischen zwei Prozessen auf demselben Rechner stattfindet, hat die Latenz eine Größenordnung von Mikrosekunden und die Bandbreite kann bis zu mehreren Gigabyte/s betragen. Wenn die Kommunikationspartner hingegen auf verschiedene Rechner verteilt sind, treten Latenzen in der Größenordnung von Millisekunden auf und die Bandbreite erreicht im günstigsten Fall noch einige MByte/s. Wenn man Rechner koppelt, die z. B. auf mehrere Kontinente verteilt sind (wie in [72] beschrieben), liegen diese Werte noch darunter.

PACX-MPI berücksichtigt diese Charakteristik, indem es zwei unabhängige Schichten für die Kommunikation in seiner Architektur einführt (siehe Abb. 3.1): eine Schicht für die Kommunikation zwischen Prozessen auf dem selben Rechner (interne Kommunikation) und eine zweite für die Kommunikation zwischen Prozessen auf verschiedenen Rechnern (externe Kommunikation).

Für die interne Kommunikation verwendet PACX-MPI die MPI-Implementierung des jeweiligen Rechners, da dieses Vorgehen eine hohe Effizienz und Portabilität garantiert. Für die externe Kommunikation hat PACX-MPI zwei so genannte Kommunikations-Dämonen eingeführt, die die MPI-Nachrichten in TCP/IP Nachrichten übersetzen und andersherum. Weiterhin sind in PACX-MPI Optimierungen für die höherwertigen Operationen des MPI-Standards für Grid Umgebungen vorhanden, z. B. für die kollektiven Operationen oder benutzerdefinierte Datentypen [40].

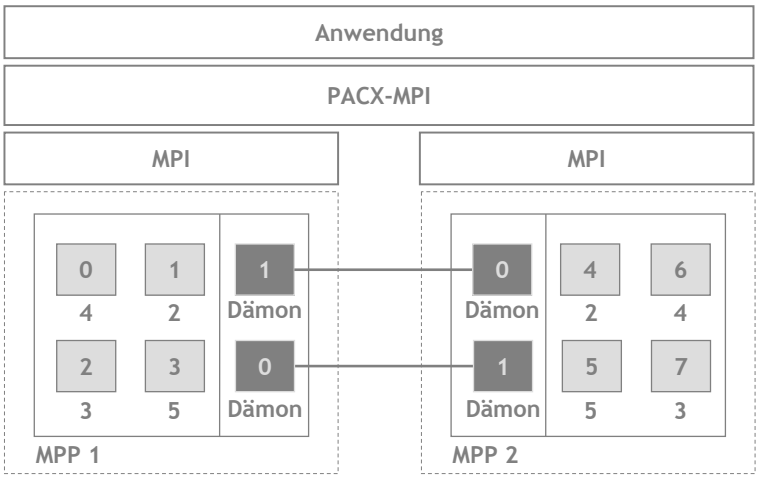

Abbildung 3.1.: Konzept von PACX-MPI

3.2.2. Verwendung und Konfiguration von PACX-MPI

PACX-MPI ist als eine Zwischenschicht zwischen dem lokalen MPI und dem Anwendungsprogramm implementiert (siehe Abb. 3.1). Durch das Schichtenkonzept ist die Benutzung von PACX-MPI transparent, d. h. es sind keine Änderungen am Programmcode notwendig. Allerdings muss eine spezielle `Include` Datei in den Code eingefügt werden, denn die Realisierung des Schichtenkonzeptes erfolgt während der Übersetzung. MPI-Aufrufe in der Anwendung werden dabei, wo es notwendig ist, automatisch durch Aufrufe von PACX-MPI ersetzt. Des Weiteren muss das Programm zusätzlich zur MPI-Bibliothek mit der Bibliothek von PACX-MPI gelinkt werden. Es stehen Bibliotheken für C und Fortran-Programme zur Verfügung. Das so erstellte PACX-MPI-Programm muss auf den zu verwendenden Rechnern vorliegen und wird jeweils mit den bekannten Startskripten für MPI-Programme (wie z. B. mpirun) gestartet.

Wenn eine verteilte MPI-Anwendung auf verschiedenen Rechnern gleichzeitig gestartet wird, liest PACX-MPI zunächst spezielle Konfigurationsdateien ein, die die folgenden Informationen liefern:

- Welche Rechner sind am aktuellen Lauf beteiligt?
- Welches ist der Rang jedes einzelnen Rechners in der gegebenen Konfiguration?
- Welches Netzprotokoll wird vom jeweiligen Rechner für die Kommunikation mit dem anderen Host benutzt?

Mit Hilfe der ersten Frage kann PACX-MPI herausfinden, welche Rechner an der aktuellen Konfiguration beteiligt sind. Die zweite Frage hat damit zu tun, dass jedem

MPI-Prozess ein globaler, eindeutiger Rang innerhalb der aktuellen globalen Konfiguration zugewiesen werden muss. Dieses Problem wird in PACX-MPI dadurch gelöst, dass auf Basis der in der Konfigurationsdatei hinterlegten Information zunächst ein eindeutiger Host-Rang für jeden beteiligten Rechner festgelegt wird. Zusammen mit den Rängen der Prozesse in der lokalen MPI-Umgebung ergibt dies eine eindeutige Zuordnung von globalen Rängen, die für die Kommunikation innerhalb der Anwendung unabdingbar ist. Die dritte Frage liefert den Namen des Kommunikationsprotokolls welches zwischen jedem Rechner-Paar verwendet wird und dessen Attribute. Diese Informationen bestehen aus einer Liste von Protokollen, die von PACX-MPI unterstützt werden (zurzeit sind das TCP/IP, multi-threaded TCP/IP, ein natives ATM Protokoll und ein auf SSL basierendes sicheres Protokoll) sowie den vom jeweiligen Protokoll abhängenden Attributen, wie z. B. die Port-Nummer unter der der Rechner per TCP/IP erreicht werden kann. Alle diese Daten sind verteilt auf zwei Konfigurationsdateien: dem so genannten *hostfile*, das die Informationen über die beteiligten Rechner enthält, und dem *netfile*, das die Beschreibung der Netzwerkparameter für den aktuellen Lauf enthält. Ein Beispiel für diese Dateien ist in Abb. 3.2 zu sehen.

```
volvox 16
hwwt3e 32
```

```
BEGIN 1;
  HOST=2, PROTOCOL=IMPI;
  HOST=2, PROTOCOL=TCP;
  HOST=2, PROTOCOL=TCP;
  HOST=2, PROTOCOL=TCP;
END

BEGIN 2;
  HOST=1, PROTOCOL=IMPI;
  HOST=1, PROTOCOL=TCP;
  HOST=1, PROTOCOL=TCP;
  HOST=1, PROTOCOL=TCP;
END
```

Abbildung 3.2.: Beispiel für PACX-MPI Konfigurationsfiles

Während das *hostfile* in jedem Fall vorhanden sein muss, ist die zweite Konfigurationsdatei optional. Falls das *netfile* nicht vorliegt, nimmt PACX-MPI an, dass alle Rechner standardmäßig das TCP/IP Protokoll mit Standardports verwenden.

PACX-MPI stellt zwei verschiedene Mechanismen für die Startprozedur zu Verfügung:

(a) für Rechnerarchitekturen, die jeweils eine einzige IP-Adresse als Zugang zu allen ihren Knoten verwenden (wie z. B. Cray T3E oder SGI Origin) kann die Verbindung zwischen den einzelnen Rechnern direkt aufgesetzt werden, da der Name der Maschine eindeutig auf eine IP-Adresse abgebildet werden kann.

(b) für Rechnerarchitekturen, bei denen die Knoten eines Rechners jeweils eigene IP-Adressen haben (Cluster) reicht die Information in der vorher genannten Form nicht mehr aus. In diesem Fall ist die Netzwerkadresse der Knoten, auf der die PACX-Dämonen laufen werden, nicht bekannt, bevor die Anwendung gestartet wurde und somit kann keine Verbindung zwischen ihnen aufgesetzt werden. Deshalb verwendet PACX-MPI in diesen Fällen einen so genannten Startup-Server. Der Startup-Server ist ein eigenständiges Programm, das die IP-Adressen aller beteiligten Knoten „einsammelt“ und diese Information an alle Prozesse verteilt. Bei dieser Vorgehensweise muss der Rechner auf dem der Startup-Server läuft, eindeutig über seinen Namen oder die IP-Adresse identifiziert werden.

Entsprechend kann das *hostfile* zwei verschiedene Formate haben: im Fall (a) besteht das *hostfile* aus einer Liste der beteiligten Rechner sowie die Anzahl der auf diesen Rechnern verwendeten Prozesse. Im Fall (b) enthält das *hostfile* den Namen bzw. die IP-Adresse des Rechners, auf der der Startup-Server gestartet wurde sowie den globalen Rang dieses Rechners.

3.3. Das Leistungsvorhersagewerkzeug Dimemas

Um die Effizienz von parallelen Programmen zu gewährleisten, ist es in der Regel notwendig, eine Analyse der Leistungsdaten (Performance Analyse) vorzunehmen. Allerdings kann die Ausführung dieser Programme in Produktionsumgebungen mit dem Ziel der Leistungsanalyse sehr aufwendig sein. Dieser Fakt spielt umso mehr eine Rolle, wenn als Ausführungsumgebung nicht nur ein einzelner paralleler Rechner sondern ein Grid gewählt wird. Deshalb ist es sinnvoll, eine Vorhersage über die Effizienz einer parallelen Anwendung machen zu können, ohne dafür die Zielplattform wirklich benutzen zu müssen. Im folgenden Abschnitt wird das Programm Dimemas [41] vorgestellt, das speziell für diese Aufgabe verwendet werden kann.

3.3.1. Konzept von Dimemas

Dimemas ist ein am Barcelona Supercomputing Center (BSC) entwickeltes Simulationsprogramm zur Vorhersage von Leistungsdaten für Message Passing Programme. Im Rahmen des DAMIEN Projektes wurde Dimemas erweitert, um Vorhersagen auch für Grid Umgebungen zu liefern [4]. Die Vorhersagen von Dimemas basieren auf der Charakterisierung der (Hardware) Architektur, auf der eine parallele Anwendung ausgeführt wird, und der Beschreibung der parallelen Anwendung selbst. Anhand dieser Daten ist Dimemas in Lage, das zeitliche Verhalten eines Programms nachzubilden.

Für die Charakterisierung der Hardware werden spezielle, konfigurierbare Parameter verwendet, die die Zielarchitektur modellieren können. Die Zielarchitektur wird anhand

dieser Parameter so vereinfacht, das die Genauigkeit der Vorhersage von Ergebnissen erhalten bleibt, ohne die Komplexität des Modells zu hoch anzusetzen da dies zu nicht vertretbaren Laufzeiten des Simulators führen würde. Das Modell der Zielarchitektur enthält nicht nur Informationen zur Hardware (Architekturmodell), sondern auch Kenndaten zur Beschreibung der verwendeten Middleware (Kommunikationsmodell).

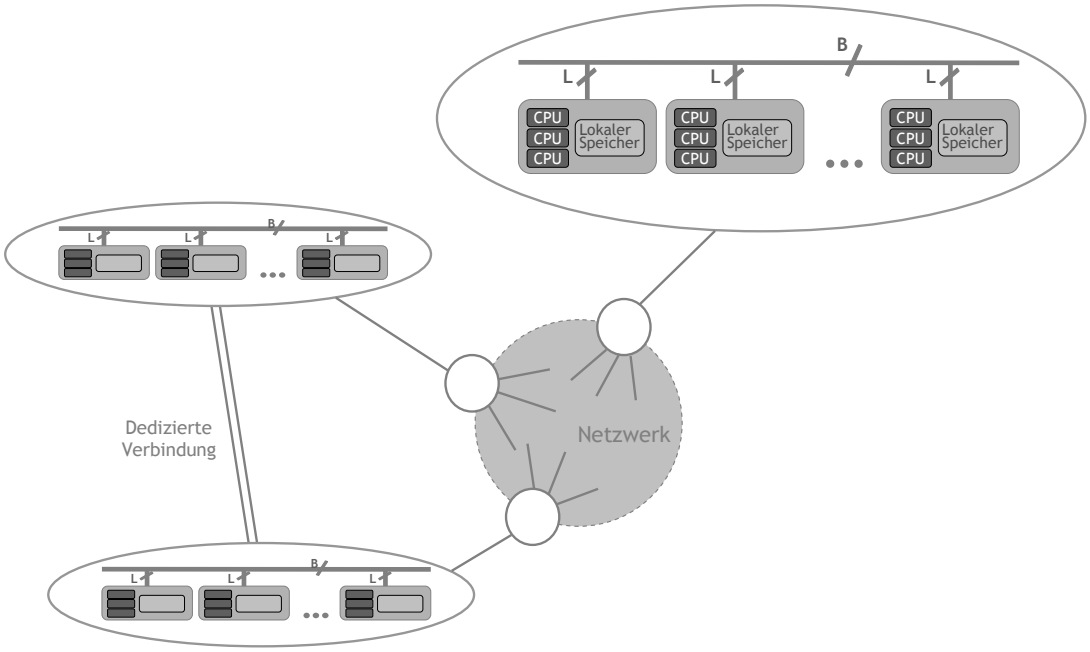

Abbildung 3.3.: Dimemas Modell zur Beschreibung der Zielumgebung/architektur [6]

Das Dimemas Architekturmodell einer Grid Umgebung besteht aus einzelnen Knoten die einen Rechner repräsentieren. Die Rechner wiederum werden als Netzwerke von SMP (Symmetric Multi Processing) Systemen beschrieben (siehe Abb. 3.3). Jedes dieser SMP Systeme besteht aus mehreren, identischen Prozessoren und dem lokalen Speicher, der von außerhalb nicht zugänglich ist. Um unterschiedliche Typen dieser SMP Systeme zu simulieren, wird eine relative Prozessorgeschwindigkeit verwendet. Sie ist ein Maß für die unterschiedliche Prozessorgeschwindigkeit von zwei Prozessoren und wird als Verhältnis der Ausführungszeiten angegeben, die ein Programm auf den zwei Prozessoren benötigt. Die Verbindung der Knoten zum Netzwerk wird durch Links repräsentiert: Eingabe-Links für eingehende Informationen und Ausgabe-Links für ausgehende Daten. Dieses Modell wird für die Beschreibung der Verbindungen innerhalb eines SMP Knotens als auch für das Netzwerk, das die Knoten verbindet, verwendet.

Die Modellierung des Netzwerkes, welches die SMP Knoten verbindet, erfolgt auf

Basis der Anzahl von Bussen (B) des Netzwerkes, die durch eine gegebene Bandbreite charakterisiert werden und den Links (L), mit denen die Knoten mit dem Netzwerk verbunden sind. Diese werden durch eine gegebene Latenz beschrieben. Des Weiteren findet die Latenz und Bandbreite zwischen den einzelnen Prozessoren und dem Speicher Beachtung. Damit ergeben sich folgende Parameter für die Beschreibung der Architektur eines Rechners:

- Anzahl der Knoten in einer Maschine
- Anzahl der Busse, die die Knoten verbinden
- Kommunikationsbandbreite der Busse
- Anzahl der Prozessoren innerhalb eines Knotens
- Anzahl der Links die einen Knoten mit dem Bussystem verbinden (Maximale Anzahl von gleichzeitig möglichen ankommenden/ausgehenden Nachrichten zum/ vom Knoten)
- Latenz für den Nachrichtenaustausch innerhalb eines Knotens
- Latenz beim Nachrichtenaustausch zwischen den Knoten
- Kommunikationsbandbreite für den Nachrichtenaustausch innerhalb eines Knotens
- relative Prozessorgeschwindigkeit

Das Weitverkehrsnetz (Wide Area Network - WAN), über das die Rechner in einer Grid Umgebung verbunden sind, wird durch eine maximale Bandbreite charakterisiert. Diese maximale Bandbreite kann aber aufgrund des aktuellen Nachrichtenverkehrs manchmal nur zum Teil erreicht werden. Um diese Bedingung im Modell nachzubilden kann der Benutzer eine Verkehrsfunktion für das Netzwerk angeben. Durch diese Funktion werden die Werte für Latenz und Bandbreite über die Zeit angepasst. Die Verbindung der Rechner zum WAN wird über einen einzelnen Link modelliert, der durch eine Latenz und die so genannte „Flighttime“ beschrieben wird. Der letzte Wert repräsentiert eine Latenz, die nicht aufgrund von Berechnung entsteht, sondern die Zeit, die benötigt wird, um eine Nachricht an das Ziel zu übertragen. Durch die „Flighttime“ kann man die durch Ausbreitung und Übertragung entstehende Verzögerung angeben [63]. Damit ergeben sich für die Beschreibung der Grid Umgebung die folgenden Parameter:

- Anzahl der Rechner im Grid
- typischer Wert für die Bandbreite des WAN
- Art der Funktion des Verkehrs (konstant, exponentiell, logarithmisch oder linear)
- maximaler Wert für den Nachrichtenverkehr

Neben den mit Hilfe des WAN allgemein beschriebenen Verbindungen, können einzelne Paare von Rechnern auch durch dedizierte Verbindungen miteinander verbunden sein.

Für die Modellierung wird das gleiche Modell wie für Verbindungen innerhalb eines Rechners zugrunde gelegt. Die dedizierten Verbindungen werden durch bestimmte QoS (Quality of Service) mit garantierter Bandbreite und, ähnlich wie beim WAN, einer gegebenen Latenz und „Flighttime“ charakterisiert. Um zu entscheiden, welche Nachrichten über die dedizierten Verbindungen verschickt werden und welche über das externe Netzwerk (WAN) müssen zusätzliche Parameter angegeben werden. Damit ergeben sich für die Beschreibung der dedizierten Verbindungen die folgenden Parameter:

- Anzahl der dedizierten Verbindungen
- Bandbreite (garantiert durch QoS)
- Liste von Nachrichten Tags (Bezeichner), die über die dedizierte Verbindung verschickt werden
- Nachrichtengröße für Nachrichten, die über die dedizierte Verbindung verschickt werden
- Liste von Kommunikatoren für kollektive Operationen, die die dedizierte Verbindung verwenden
- Flighttime
- Latenz

Es gibt verschiedene Kommunikationsmodelle für die Modellierung von Punkt-zu-Punkt Kommunikationen und für die kollektiven Operationen. Abbildung 3.4 zeigt beispielhaft das Modell für die Punkt-zu-Punkt Kommunikation. Darin sind die verschiedenen Zeitanteile, die in die Berechnung des Kommunikationsaufwandes einfließen, dargestellt. In diesem Modell werden die Anteile der Latenz in zwei Komponenten aufgeteilt: eine Latenz, die aufgrund des Sendens von der Startmaschine bei Verwendung von Ressourcen der Maschine entsteht (Latenz der Maschine) und eine Latenz („Flighttime“), die aufgrund der Entfernung zur Zielmaschine auftritt und nicht durch Verwendung von Ressourcen der Maschine entsteht.

Die konkurrierende Verwendung der Netzwerk-Ressourcen einer Maschine wird in Dimemas durch eine Zeit simuliert, die abhängig von der Verfügbarkeit von Links und Bussen ist. Falls die für die Kommunikation benötigten Links und Busse zur Verfügung stehen, blockiert Dimemas diese für die Zeit der Übertragung. Die Zeit für die Übertragung hängt von der Größe der Nachricht und der Bandbreite der Verbindung ab. Der Einfluss des Verkehrs innerhalb des WANs wird im Modell durch eine zusätzliche Funktion erfasst. Tatsächlich wird dieser Einfluss durch einen Faktor, der die Bandbreite der Verbindung reduziert, berechnet. Nach Ablauf der „Flighttime“ erfolgt schließlich die Freigabe der Ressourcen.

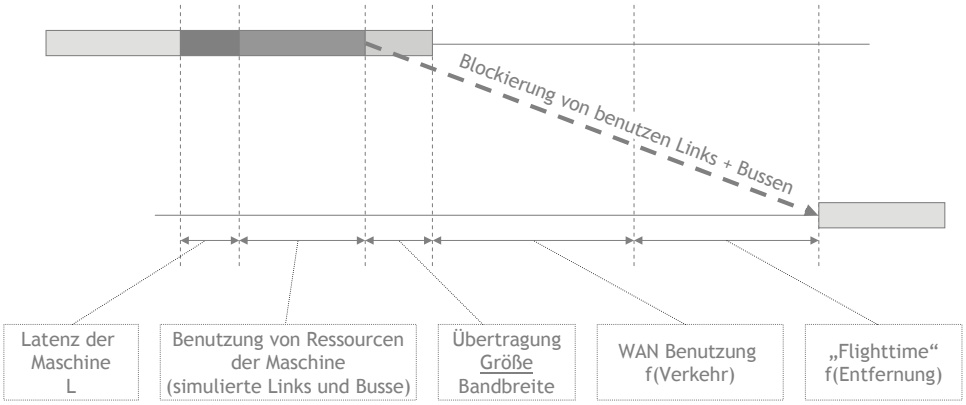

Abbildung 3.4.: Dimemas Modell für die Modellierung der Punkt-zu-Punkt Kommunikation [5]

Die Parameter für die in Dimemas enthaltenen Kommunikationsmodelle erlauben es, verschiedene Implementierungen von Kommunikationsbibliotheken und Netzwerkcharakteristiken abzubilden. Diese werden in [42] und [8] beschrieben. Sowohl für die Punkt-zu-Punkt Kommunikationen als auch für die kollektiven Operationen werden zwei unterschiedliche Ebenen berücksichtigt: zum einen für den Nachrichtenaustausch, der zwischen zwei Prozessen stattfindet, die innerhalb eines Rechners laufen und zum anderen eine Ebene für die entfernte Kommunikation zwischen Prozessen, die auf unterschiedlichen Rechnern laufen. Außerdem wird noch eine Klassifizierung für die lokale Kommunikation eingeführt: zum einen die Kommunikation, die innerhalb eines SMP-Knotens stattfindet und zum anderen, die Kommunikation, die über das interne Netzwerk eines Rechners (Interconnect) stattfindet. Für die Simulation ergeben sich damit folgende Parameter:

- Kommunikationsmodell, das zur Simulation von kollektiven Operationen zwischen den Knoten benutzt wird (Logarithmisch, linear oder konstant)
- Kommunikationsmodell, das zur Simulation von kollektiven Operationen innerhalb des WAN benutzt wird (Logarithmisch, linear oder konstant)

Eine detaillierte Modellierung der kollektiven Operationen ist möglich, wenn spezifische Daten (Modell und Nachrichtenlängen) für die einzelnen Operationen vom Benutzer zur Verfügung gestellt werden. Diese werden in einer gesonderten Konfigurationsdatei beschrieben und als optionaler Parameter angegeben.

Die Beschreibung der Anwendung wird aus Aufzeichnungen der wichtigsten Kenndaten der Anwendung in Form von konkret erzeugten Tracing-Daten gewonnen. Programm-Tracing ist eines der Standardverfahren zur Analyse von Programmläufen. Dabei wird während eines Programmlaufs das Auftreten festgelegter Ereignisse zusammen mit dem

Zeitpunkt (Zeitstempel) und Ort (z. B. Prozess oder Knoten), sowie beliebigen weiteren Detail-Informationen in so genannten Tracedateien protokolliert. Bei Aufzeichnungen von Nachrichten-Ereignissen enthalten diese typischerweise Details über die aktuelle Nachricht (z. B. den Quell- oder Zielort und den Nachrichtenbezeichner). Diese Daten stehen anschließend für die Analyse oder, wie im Fall von Dimemas, für die Simulation zur Verfügung. Da beim Tracing die Ereignisse in räumlichen und zeitlichen Zusammenhang protokolliert werden, ist damit eine Rekonstruktion des Laufzeitverhaltens einer Anwendung möglich. Tracedateien für Dimemas bestehen aus Einträgen (Records) die jeweils ein Ereignis definieren. Es existieren verschiedene Records für die verschiedenen Ereignis-Typen: CPU burst, Sends, Receives, Beginn von Blöcken, Ende von Blöcken u. s. w. Jeder Record ist mit einem Task und der Thread ID (Knoten und Prozessor) verknüpft.

Der für die Ermittlung der Tracedaten konkret durchzuführende Lauf kann auf jeder Art von Rechner, unabhängig von der zu modellierenden Zielarchitektur, gemacht werden. Es ist sogar möglich, diese Tracing-Daten auf einem Rechner mit nur einem Prozessor zu erzeugen, bei dem die MPI-Prozesse auf einen Prozessor abgebildet werden. Obwohl die Leistung bei diesem Lauf sehr niedrig sein wird, können die dabei erzeugten Daten für Dimemas verwendet werden. Ein einzelner Lauf eines Programms hat damit das Potential die benötigte Menge an Informationen zu liefern, ohne diesen Lauf in einer Produktionsumgebung durchführen zu müssen.

Das für die Beschreibung der Anwendung verwendete Modell (Anwendungsmodell) hat drei verschiedene Ebenen: Anwendung, Prozess und Thread (siehe Abb. 3.5). Es erlaubt die Spezifikation der Programmiermodelle Message Passing, Shared Memory oder einen Mix aus beiden. Die Ebene der Anwendung stellt dabei ein einzelnes paralleles Programm während seiner Ausführung dar und ist gleichzeitig die Instanz, die Ressourcen zugewiesen bekommt. Eine Anwendung besteht aus einzelnen Prozessen. Ein Prozess ist ein logischer Adressraum in dem verschiedene Kontrollflüsse (Steuerung des Ablaufs innerhalb eines Programms) ausgeführt werden können. Ein Prozess wird auf einen Knoten innerhalb einer Anwendung abgebildet, wobei alle seine Thread auf jedem Prozessor innerhalb des SMP Knotens laufen können. Ein Thread ist der sequentielle Kontrollfluss innerhalb eines Prozesses. Alle diese Threads teilen sich denselben Adressraum.

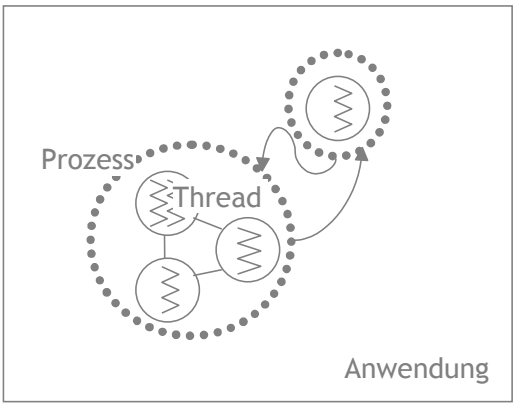

Abbildung 3.5.: Dimemas Modell zur Beschreibung der Anwendung [7]

3.3.2. Verwendung von Dimemas

Die für die Dimemas-Simulation verwendeten Ereignisse müssen zum Zeitpunkt ihres Auftretens aufgenommen werden. Aus diesem Grund muss eine Anwendung instrumentiert werden, damit sie in der Lage ist an ausgewählten Punkten die betreffenden Ereignisse zu speichern. D. h. um Tracing-Dateien für die Verwendung mit Dimemas zu erzeugen, muss der Benutzer eine Tracing-Bibliothek bei der Erzeugung des Programmcodes hinzufügen (linken). Dadurch wird das Programm vom Compiler mit Aufrufen von zusätzlichen Mess- und Zählroutinen angereichert. Unter Angabe von zusätzlichen Parametern oder Umgebungsvariablen (abhängig von der verwendeten Tracing-Bibliothek) muss dann, wie im vorherigen Abschnitt beschrieben, ein Programmlauf durchgeführt werden, bei dem die Tracing-Daten erzeugt werden.

Basierend auf der initialen Tracedatei, kann der Benutzer verschiedene Parameter angeben, um eine Zielkonfiguration zu simulieren. Die Parameter zur Beschreibung der Zielarchitektur werden in einem speziellen Konfigurationsfile, das dem SDDF [3] Format entspricht, zusammengestellt. Dimemas kann nun eine Abschätzung des Verhaltens der Anwendung anhand der modifizierten Parameter vornehmen. Die Ausgaben von Dimemas sind zum einen die Ergebnisse der Vorhersage von Ausführungszeiten in Form von Text. Zusätzlich kann sich der Benutzer auch eine Tracedatei als Ausgabe generieren lassen und mit einem Analyseprogramm (z. B.Vampir [13]) anzeigen lassen, um die Resultate der Vorhersage zu untersuchen.

3.4. Konzept und Implementierung des Grid Configuration Managers

Aus den in Abschnitt 3.1 aufgeführten Zielen wurden fünf Punkte für das Grundkonzept des Grid Configuration Managers abgeleitet. In Abb. 3.6 sind diese fünf Punkte in einem Ablaufdiagramm dargestellt:

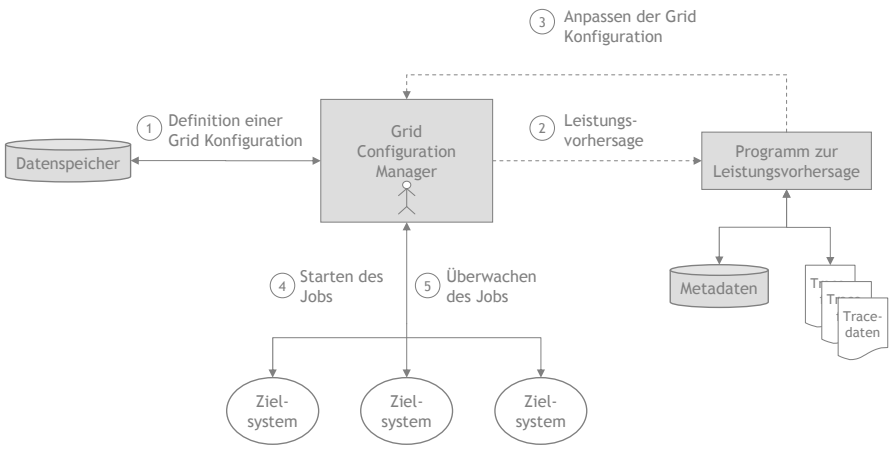

Abbildung 3.6.: Ablaufdiagramm des Grid Gonfiguration Managers

Nach der Definition einer Grid Konfiguration[1] (1) durch den Benutzer können die Daten in einer Datei abgespeichert werden (Datenspeicher). Danach erstellt der GCM in einem optionalen Schritt mit Hilfe von Dimemas Vorhersagen für das Laufzeitverhalten der Anwendung (2). Dazu verwendet er Tracedaten und, falls vorhanden, Angaben aus früheren Vorhersagen in Form von Metadaten. Je nach Ergebnis der Vorhersage kann der Benutzer eine Anpassung der Konfiguration durchführen (3). Die aktuelle Grid Konfiguration wird auf den Zielsystemen mit Hilfe des GCM gestartet (4). Der GCM wird auch dazu verwendet, eine Überwachung der laufenden Anwendung bzw. eine Sichtung der Ergebnisse (5) durchzuführen. Aus diesem Grundkonzept ergeben sich die drei Hauptkomponenten des Grid Configuration Managers:

(A) Definition einer Grid Konfiguration

(B) Leistungsvorhersage mit Dimemas

(C) Starten und Überwachen einer Grid Anwendung

[1] Eine Grid Konfiguration ist die Zusammenstellung der für die Durchführung einer Anwendung verwendeten Ressourcen.

Alle Komponenten des Konzeptes sind in einem Programm integriert und können so miteinander interagieren bzw. von einer zentralen Stelle aus verwendet werden. Um eine Unterstützung für unterschiedliche, heterogene Grid Umgebungen zu ermöglichen, wurde eine spezielle Abstraktionsschicht in das Konzept eingeführt. Die drei Hauptkomponenten sowie das Abstraktionskonzept und deren Implementierungen werden in den Unterabschnitten dieses Kapitels detailliert erläutert.

Die Umsetzung des Grid Configuration Managers in eine grafische Oberfläche folgt dem Model-View-Controller Konzept (MVC). Dieses Konzept trennt Eingabe, Verarbeitung und Ausgabe in grafisch orientierter Software. Deshalb ergeben sich für die Implementierung des MVC die folgenden Grundelemente:

- Das *Model* (Datenmodell), enthält die dauerhaften (persistenten) Daten der Anwendung. Es kennt weder die View noch den Controller und weiß nicht, wie, ob und wie oft es dargestellt und verändert wird. Änderungen im Model werden über einen Aktualisierungsmechanismus bekannt gegeben, indem ein Event (Ereignis) ausgelöst wird.
- Die *View* (Darstellungsschicht) realisiert die Wiedergabe des Modells z. B. auf einem Bildschirm. Sie kennt das Model und ist dort registriert, um sich selbständig aktualisieren zu können.
- Der *Controller* verwaltet die Sichten, nimmt von ihnen Benutzeraktionen entgegen, wertet diese aus und hat schreibenden Zugriff auf das Modell. Er enthält die Intelligenz und steuert den Ablauf (engl. Workflow) der Anwendung.

Die Implementierung des Grid Configuration Managers erfolgte in der Programmiersprache Java [2] unter Verwendung der grafischen Bibliothek Swing [64]. Um eine möglichst große Abstraktionsfähigkeit zu erreichen, wurden objektorientierte Ansätze wie Definition von Schnittstellen, abstrakte Klassen und Vererbung konsequent eingesetzt. Aus den o. g. Grundelementen des MVC wurden zunächst drei Hauptklassen für die Implementierung abgeleitet: die Klasse `ContentModel` als Modell, die Klasse `GcmFrame` als Sammelpunkt für die Ausgabe und mehrere im Modell integrierte `Listener` Klassen, die für die Kontrolle der Benutzereingaben verwendet werden. Sie sind zusammengefasst in einem Klassendiagramm in Abb. 3.7 dargestellt.

In der Klasse `ContentModel`, die als reine Schnittstelle entwickelt wurde, werden alle Daten gesammelt, verarbeitet und für die Darstellung in den Ausgabeklassen aufgearbeitet. Sie stellt die oberste Schicht einer Abstraktion von Datenmodellen für den Grid Configuration Manager dar. Die eigentlichen Grid Konfigurationen werden in Klassen zusammengefasst, die auf der Schnittstelle `ConfigurationModel` basieren. Daten, die der Beschreibung der Anwendung dienen, werden in der Klasse `ApplicationModel` zusammengefasst. Für die Darstellung gibt es neben der Klasse `GcmFrame` noch die abstrakte Klasse `ConfigurationCanvas` die für die grafische Wiedergabe der Grid Konfigurationen verwendet wird.

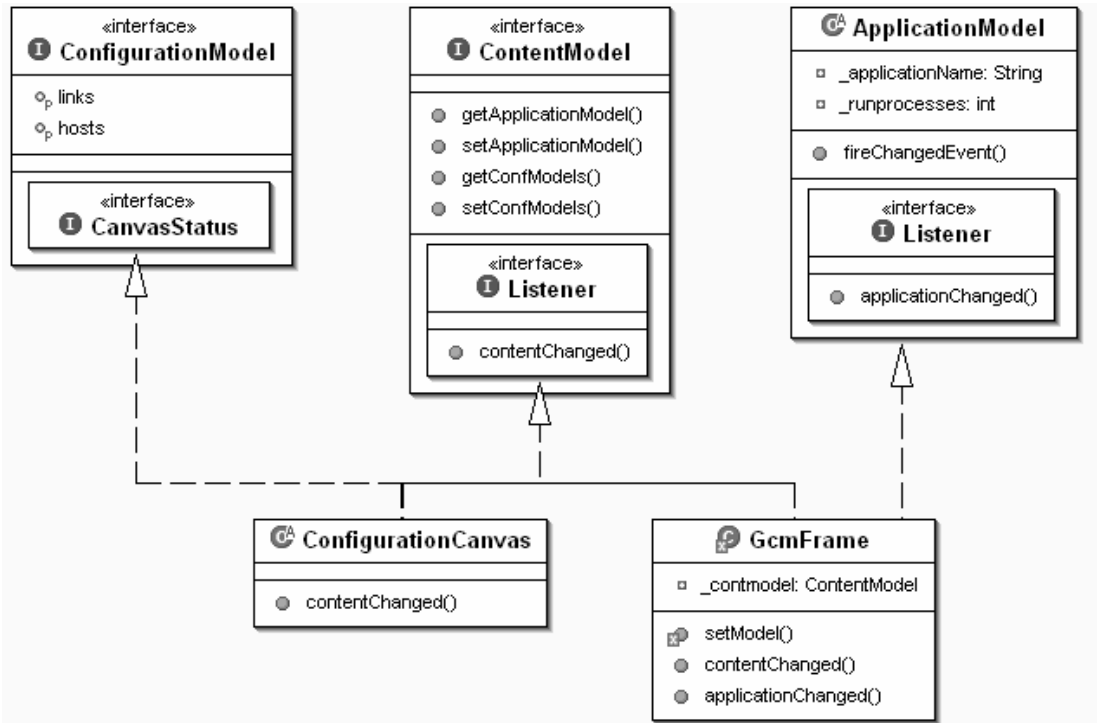

Abbildung 3.7.: Ausschnitt aus dem Klassendiagramm des Grid Configuration Managers (Oberste Abstraktionsschicht)

Zum jetzigen Zeitpunkt existieren zwei Implementierungen der genannten Schnittstellen bzw. der abstrakten Klassen. Zunächst wurde eine „DefaultImplementierung“ vorgenommen, mit der grundsätzliche Funktionalitäten wie öffnen/speichern von Konfigurationen u. ä. getestet wurden. Danach wurde eine komplette Implementierung für die Erstellung von Grid Konfigurationen auf Basis von PACX-MPI entwickelt. Für das „PACX-Modell“ wurden folgende Klassen verwirklicht: `PacxContentModel`, `PacxConfigurationModel` und `PacxApplicationModel`. Dieses Modell setzt alle Funktionalitäten um, die für die Unterstützung von PACX-MPI benötigt werden (siehe Kap. 3.4.6). Grundsätzlich ist es aber auf Grund des objektorientierten Ansatzes des GCM möglich, weitere Implementierungen vorzunehmen und damit auch andere Kommunikationsbibliotheken, wie z. B. MPICH-G2 [54] oder Stampi [51], zu unterstützen.

3.4.1. Definition einer Grid Konfiguration

Eine Konfiguration mit dem Grid Configuration Manager besteht aus *Hosts* und *Links*. Diese Objekte repräsentieren die an einer Konfiguration beteiligten Rechner bzw. deren

Verbindungen und enthalten alle Daten zur Beschreibung derselben einschließlich der Zugangspasswörter. Die Informationen, die in den Host- und Linkklassen gesammelt werden, können sowohl für die grafische Darstellung als auch für die Weiterverarbeitung z. B. für die Erstellung von Konfigurationsdateien verwendet werden.

Um eine Abstraktion für verschiedene Rechner zu ermöglichen, wurde die Klasse `Host` als reine Schnittstelle definiert, die nur über Funktionalitäten zum Speichern/Rückgabe von folgenden Informationen verfügt:

- Name des Hosts (z. B. Rechnername)
- Index des Hosts (innerhalb der Grid Konfiguration)
- Status (ausgewählt/nicht ausgewählt) und Position des Hosts für die Darstellung

Da das Ziel der jetzigen Implementierung auch die Unterstützung der Kommunikationsbibliothek PACX-MPI umfasst, wurde zusätzlich die abstrakte Klasse `PacxHost` geschaffen. Diese erweitert die Host-Schnittstelle um Funktionalitäten für die Handhabung der Daten für ein „PACX-Modell“:

- Anzahl von Prozessen, die von einer PACX-MPI-Anwendung auf diesem Rechnern benutzt werden sollen
- „Sleeptime“ (Wartezeit, um Problemen beim Starten einer Anwendung vorzubeugen)
- Pfad der Ausführungsdatei einer PACX-MPI-Anwendung auf diesem Rechner
- Startkommando für das Starten einer PACX-MPI-Anwendung auf diesem Rechner

Um die Einbindung einer zweiten Abstraktionsebene für die Unterstützung verschiedener Grid Zugangsmechanismen zu ermöglichen, werden diese Schnittstellen erst in der nächsten Ebene konkret implementiert (siehe Kapitel 3.4.2).

Für die Verbindungsobjekte wurde ebenfalls eine abstrakte Klasse entworfen. Die Klasse `Link` enthält Funktionalitäten zum Speichern/Rückgabe von verbindungspezifischen Informationen. Da Verbindungen zwischen zwei Rechnern auch durch mehrere Kanäle beschrieben werden können, bestehen die Links wiederum aus einzelnen *Connections*. In der Klasse `Connection` können folgende Daten verwaltet werden:

- Netzprotokoll, das für die Verbindung benutzt wird
- Portnummer für den Verbindungsaufbau
- zusätzliche Attribute

Für die Speicherung einer Konfiguration wird das XML-Format verwendet, da dieses sicherstellt, dass ein einfacher Datenaustausch und Lesbarkeit erreicht werden kann. Für die Implementierung wurde die Bibliothek XStream [88] verwendet. Beim Speichern einer Konfiguration können die vom Benutzer angegebenen Zugangspasswörter

verschlüsselt abgespeichert werden. Dafür wurden die in Java 1.4 enthaltenen Java Cryptography Extension (JCE) und die Java Secure Socket Extension (JSSE) verwendet. Zusätzlich existiert die Möglichkeit, Konfigurationen ohne Speicherung der Passwörter zu exportieren bzw. zu importieren. Diese Vorgehensweise ist sinnvoll, wenn man Konfigurationen an andere Benutzer weitergeben will.

Um die Erstellung der Grid Konfigurationen weiter zu verbessern, wurde das Konzept der *Toolbox* entwickelt, mit der der Benutzer vorkonfigurierte *Hosts* verwalten kann, die unterschiedliche Zugangsmechanismen unterstützen. Diese Hostobjekte werden in der *Toolbox* abgelegt damit sie z. B. anderen Benutzern zur Verfügung stehen. Sie können aber auch dazu verwendet werden, um Rechnerressourcen zu beschreiben, die in einem Rechenzentrum vorhanden sind. Die vorkonfigurierten Hostobjekte werden in einer speziellen XML-Datei (`confHost.xml`) gespeichert. Diese Datei wird beim Starten des GCM von einer Stelle geladen, die der Benutzer spezifizieren kann.

3.4.2. Unterstützung verschiedener Grid Zugangsmechanismen

Unterschiedliche Grid Zugangsmechanismen werden innerhalb des GCM durch spezielle Schnittstellen definiert, die alle von der Klasse `Host` abgeleitet werden. Dadurch entsteht eine neue Abstraktionsebene, die eine einfache Erweiterung des *Host* Konzeptes erlaubt.

Zurzeit unterstützt der Grid Configuration Manager Globus, UNICORE und ssh basierende Umgebungen. Die Schnittstelle `InteractiveHost` erweitert die Klasse `Host` um Methoden für die Handhabung von ssh Verbindungen bzw. den Dateitransfer per scp oder ftp. Entsprechend existieren auch die Schnittstellen `UnicoreHost` und `GlobusHost`.

`PacxInteractiveHost`, `PacxUnicoreHost` und `PacxGlobusHost` stellen Implementierungen der o. g. Schnittstellen dar. Wenn ein Benutzer seiner Grid Konfiguration ein neues *Hostobjekt* hinzufügt wird dieses von der Klasse `HostFactory` anhand des ausgewählten Typs erzeugt. Die Klasse `HostFactory` erzeugt gleichzeitig das grafische Dialogobjekt für die Konfiguration dieses Hosts. In diesem Zusammenhang wird in der Implementierung des GCM das so genannte Factory(Fabrik)-Pattern verwendet. Dabei handelt es sich um ein fundamentales softwaretechnisches Design-Muster zur Erzeugung von Objekten. Die Aufgabe des Factory-Patterns besteht darin, die konkreten Klassen von den erzeugenden zu isolieren, um nicht von deren konkreten Implementierungen abhängig zu sein (im Fall des GCM z. B. die Host-Klassen). Ein weiterer Vorteil des Factory-Pattern besteht darin, dass die Erzeugung von Objekten auf eine Stelle konzentriert wird. Dies macht das Design Änderungen gegenüber wesentlich robuster [21]. In Abbildung 3.8 ist eine Zusammenstellung der Host-Klassen zu sehen.

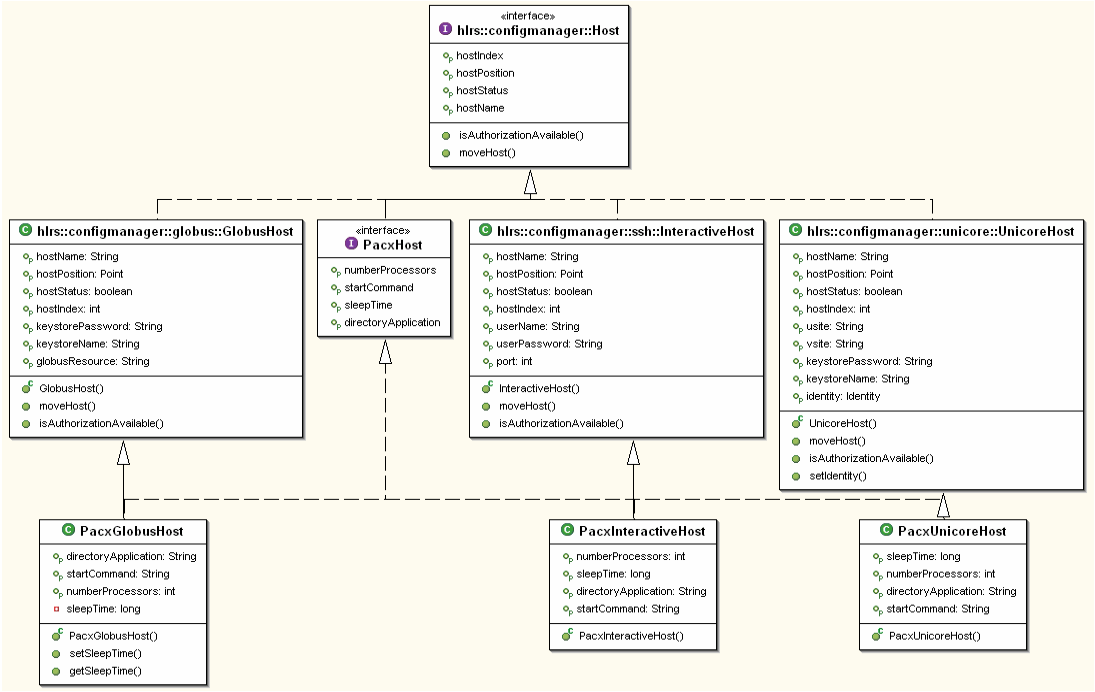

Abbildung 3.8.: Die Klassen der *Hostobjekte*

Im folgenden Abschnitt wird die Integration der vom GCM unterstützten Zugangsmechanismen kurz erläutert. Da ssh auch heute noch eine bedeutende Rolle als Zugangsmechanismus zu Hoch- und Höchstleistungsrechnern darstellt, wurde diese auch in den GCM integriert. Für die Implementierung wurde die Bibliothek JSch [53] verwendet. JSch ist eine Implementierung des ssh Protokolls in Java. Sie erlaubt Benutzern, Verbindungen zu ssh Servern herzustellen, Portweiterleitung, X11-Weiterleitung und Dateitransfer etc. auszuführen. Seine Funktionalitäten werden in der Klasse `SSHCommander` des GCM gekapselt, die für die Ausführung von ssh-Kommandos zuständig ist. In den Implementierungen der Schnittstelle `InteractiveHost` werden Daten für die Konfiguration einer ssh-Verbindung gespeichert:

- Loginname auf dem ausgewählten Rechner
- Passwort für diesen Rechner
- Portnummer für den Verbindungsaufbau

UNICORE wurde mit Hilfe der Kommandozeilenschnittstelle Arcon [36] als Zugangsmechanismus in den GCM eingebaut. Mit Hilfe der Arcon Bibliothek werden in der Klasse `UnicoreCommander` die UNICORE spezifischen AJOs, die eine Beschreibung des Jobs enthalten, generiert, an die UNICORE Rechner verschickt und die Ausgaben verarbeitet.

Um eine Verbindung zu den UNICORE Servern aufzubauen und Jobs zu submittieren werden in den Implementierungen der Schnittstelle `UnicoreHost` folgende Informationen gespeichert:

- Speicherort des Schlüsselspeichers (Keystore) mit den UNICORE-Benutzerzertifikaten
- Passwort zum Öffnen des Benutzerkeystores
- Name des Gateways der VSite, die benutzt werden soll
- Name der USite (Rechner), die benutzt werden soll

Bei der Benutzung von UNICORE muss man einen wichtigen Unterschied zu ssh beachten. UNICORE unterstützt derzeit keinen interaktiven Zugang zu seinen Ressourcen. Das bedeutet, dass alle Jobs (selbst einfache Kommandos wie `ls`) an ein Batchsystem übergeben werden. Da die (Vorab-)Reservierung von Ressourcen, die über mehrere Standorte verstreut sind, nicht möglich ist, heißt das auch, dass der Benutzer dafür sorgen muss, dass die Ressourcen zu einem bestimmten Zeitpunkt zur Verfügung stehen.

Für die Integration von Globus als Zugangsmechanismus in den GCM wurde das Java Commodity Grid Kit (CoG Kit) [56] aus dem Globus Toolkit verwendet. Das Java CoG Kit stellt Schnittstellen zur Verfügung, um die Proxy-Zertifikate für den Benutzer zu generieren und zu verwalten. Außerdem gibt es noch spezielle Klassen, um Zugang zu GRAM und GASS Diensten zu erhalten und Ausgabe- und Fehlermeldungen von laufenden Jobs aufzufangen. Diese Funktionalitäten werden von der Klasse `GlobusCommander` benutzt, um Befehle in Globus Grids abzusetzen. Außerdem werden in dieser Klasse auch die von Globus benötigte generische Beschreibung des Jobs im Globus Format erstellt. Ähnlich wie in der Implementierung für UNICORE werden in den Implementierungen der Schnittstelle `GlobusHost` alle für den Aufbau einer Globus-Verbindung benötigten Informationen gesammelt:

- Speicherort des Schlüsselspeichers (Keystore) mit den Globus-Benutzerzertifikaten
- Passwort zum Öffnen des Benutzerkeystores
- Beschreibung der Globus Ressource

3.4.3. Leistungsvorhersage mit dem Grid Configuration Manager

Der Grid Configuration Manager wurde mit dem Leistungsvorhersageprogramm Dimemas integriert, damit eine (automatisierte) Vorhersage des Laufzeitverhaltens von parallelen Anwendungen möglich ist. Prinzipiell erlaubt das softwaretechnische Design des GCM aber auch die Integration anderer Programme zur Leistungsvorhersage, da die entsprechenden Funktionalitäten in gesonderten Klassen gekapselt wurden. Dafür wurde zunächst eine spezielle Implementierung der Klasse `ConfigurationModel`

vorgenommen, die Daten für ein „Dimemas-Modell“ verwalten. In diesem Modell werden alle für die Konfiguration von Dimemas notwendigen Parameter (siehe Kapitel 3.3.1) gesammelt und verarbeitet. Da diese Parameter sehr umfassend sind und um die Übersichtlichkeit zu erhöhen, wurden weitere (Unter-) Modellklassen entwickelt:

- Die Klasse `Node` zur Beschreibung eines einzelnen SMP-Knotens.
- Die Klasse `WANConfiguration` zur Beschreibung der allgemeinen Parameter für das externe Netzwerk
- Die Klasse `FileSystem` für die Parameter zur Beschreibung des Filesystems.
- Die Klasse `ConfigurationFiles` für die Datenhaltung von optionalen Konfigurationsdateien für Dimemas wie z. B. für die Beschreibung der kollektiven Operationen.
- Die Klasse `SimulatorInformation` für Daten, die beim Aufruf des Simulators verwendet werden sollen, z. B. ob als Ausgabe eine Trace-Datei geschrieben werden soll.

In der Klasse `ApplicationModel` werden die Angaben für die Beschreibung der Anwendung zusammengefasst:

- Name(n) und Speicherort der Tracing-Datei(en)
- Verteilung der Prozesse auf die Maschinen

3.4.3.1. Konfiguration und Starten von Dimemas mit dem GCM

Die für die Erstellung von Dimemas-Konfigurationsdateien benötigten Parameter werden aus den im „Dimemas-Modell“ enthaltenen Host- und Linkobjekten sowie den (Unter-)Modellklassen extrahiert. Für die Eingabe der Parameter wurden Eingabedialoge für Dimemas Hostobjekte sowie weitere spezielle Eingabedialoge für die zugehörigen (Unter-)Modelle entwickelt. In die Konfiguration werden außerdem die in der Klasse `ApplikationModel` enthaltenen Angaben eingearbeitet. In einem „Mapping“-Dialog kann der Benutzer festlegen, wie die einzelnen Prozesse, die von der Anwendung verwendet werden, auf die zu Verfügung stehenden Maschinen (Hosts) und die darin enthaltenen Knoten verteilt werden. Die Klasse `DimemasResultsWindow` ermöglicht die Ausgabe der von Dimemas ermittelten Ergebnisse oder Fehlermeldungen innerhalb des GCM. Damit kann der GCM jetzt auch als grafische Oberfläche für den Dimemas Simulator verwendet werden.

In einer speziellen Klasse `Dimemas` sind alle Methoden gekapselt, die für die Durchführung von Dimemas-Simulationen und der Speicherung von Ergebnissen benötigt werden (siehe Abb. 3.9):

- Die Methode `generateCommand()` generiert das Kommando zum Starten von Dimemas. Dazu werden die in der Klasse `SimulatorInformation` gesammelten Daten ausgewertet.
- Die Methode `runDimemas()` startet Dimemas mit dem vorher generierten Kommando und Konfigurationsfile. Das Starten des Dimemas- Simulators kann auf Anforderung durch den Benutzer erfolgen oder automatisiert von den im folgenden Kapitel beschriebenen Heuristiken verwendet werden. Dafür muss jeweils sichergestellt sein, dass sich eine lauffähige Version von Dimemas auf dem Rechner des Benutzers befindet.
- Die Methode `createDimemasConfFile()` erstellt das Konfigurationsfile für die Simulation. Dazu werden die im „Dimemas-Modell“ gesammelten Daten ausgewertet. Der für die im SDDF-Format zu erstellenden Konfigurationsdateien notwendige Header, der die allgemeinen Deklarationen zur Benutzung der Konfigurationsdatei für Dimemas enthält, wird aus einer speziellen Textdatei (`header.txt`) generiert.
- Die Methode `writeExecTimes()` extrahiert die Ausführungszeiten aus dem von Dimemas produzierten Ausgabedaten und speichert sie in einem internen Format.

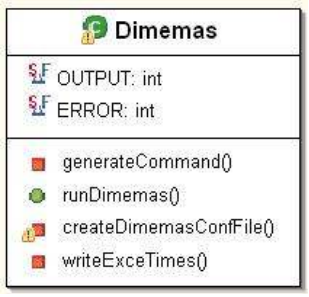

Abbildung 3.9.: Methoden der Klasse Dimemas

3.4.3.2. Automatische Leistungsvorhersage mit Hilfe von Heuristiken

Das Ziel dieser Komponente des Grid Configuration Managers ist die automatisierte Ausarbeitung eines Vorschlages für eine Konfiguration, die eine möglichst optimale Ausführungsumgebung für die ausgewählte Anwendung darstellt. Dazu werden verschiedene „Test-Konfigurationen“ für die Verteilung der erforderlichen Prozesse auf die vorhandenen Maschinen erstellt und die damit zu erzielenden Ausführungszeiten anhand von Dimemas-Simulationen ermittelt und verglichen. Die für die Simulationen erforderlichen Konfigurationsdateien werden automatisch erstellt, das Programm gestartet und die Ergebnisse (Ausführungszeiten) gesammelt. Am Ende wird die Konfiguration,

die als Ergebnis die kürzeste Laufzeit ergibt, als Vorschlag in eine PACX-Konfiguration umgewandelt, die der Benutzer als Grundlage für das Starten der Anwendung verwenden kann. In Abb. 3.10 ist diese prinzipielle Vorgehensweise noch einmal als Schema dargestellt.

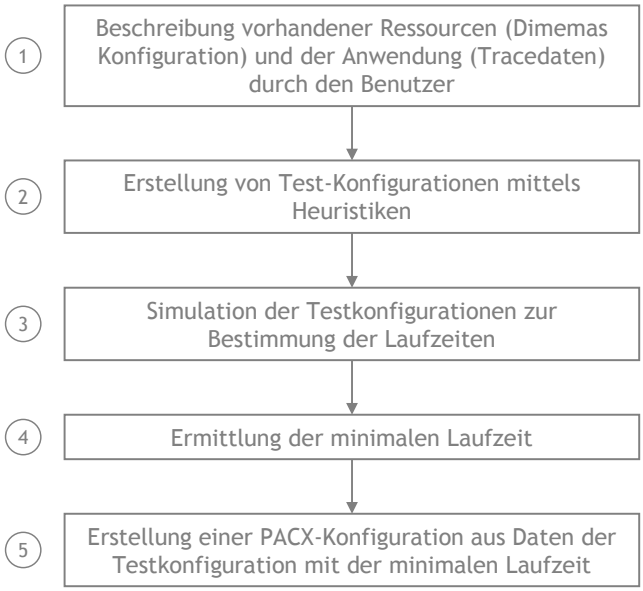

Abbildung 3.10.: Prinzipielle Vorgehensweise des GCM bei der automatischen Leistungs-vorhersage

Die Erstellung der Testkonfigurationen stellt prinzipiell ein Optimierungsproblem dar, dessen Lösung nur sehr aufwendig zu berechnen ist, da die Möglichkeiten für die Verteilung der Prozesse und die Schwankungen der Eingangsdaten für die Simulation (Bandbreite, Latenz) eine große Spannweite umfassen können. Deshalb verwendet der Grid Configuration Manager Heuristiken für die Erstellung der zu testenden Konfigurationen. Heuristiken können für Optimierungsprobleme relativ gute Lösungen mit weniger Rechenaufwand liefern. Sie können aber nur eingeschränkte Aussagen darüber machen, wie gut die gefundene Lösung tatsächlich ist. Dabei wird also ein Kompromiss zwischen Rechenaufwand für die Optimierung und Güte der gefundenen Lösung eingegangen.

Die Eingangsdaten für die Erstellung der Heuristiken werden aus den durch den Benutzer angegeben Beschreibungen der Maschinen und der Anwendung extrahiert. Dabei handelt es sich um:

- Anzahl der Prozesse, die die Anwendung verwenden möchte

- Anzahl der Maschinen, die zur Verfügung stehen und

- Anzahl der auf diesen Maschinen vorhandenen Prozessoren

Für die Erstellung der Testkonfigurationen werden zwei verschiedene Heuristiken verwendet: „Blockweise Verteilung“ und „Einfache Verteilung“.

Blockweise Verteilung

Viele parallele Programme verwenden eine Prozessverteilung, bei denen die Prozesse in einem n-dimensionalen kartesischen Gitter angeordnet sind (Blockstrukturen). Die dominierende Kommunikation bei diesen Programmen ist damit die Nachbarschaftskommunikation. D. h. ein Prozess tauscht Nachrichten mit seinem rechten und linken Nachbar, mit dem oberen und unteren Nachbar usw. aus. Deshalb hat die Verteilung der Prozesse einen entscheidenden Einfluss auf das Kommunikationsverhalten einer parallelen Anwendung. Falls die Prozesse über mehrere Maschinen verteilt sind, spielen diese Kommunikationsmuster eine wichtige Rolle für die zu erzielenden Ausführungszeiten, da die Kommunikation über Maschinengrenzen hinweg i. d. R. langsam ist.

In Abb. 3.11 ist beispielhaft für eine auf einem 2-dimensionalen Gitter basierende Prozessverteilung dargestellt, welchen Einfluss die Verteilung der Prozesse über zwei Maschinen haben kann. In diesem Beispiel sind 24 Prozesse in einem 6×4 Gitter angeordnet. Im Fall a) sind die Prozesse so verteilt, dass sich 12 Prozesse auf dem ersten Rechner und 12 Prozesse auf dem zweiten Rechner befinden. Da sich somit ein Vielfaches der Anzahl der Prozesse in y-Richtung auf einem Rechner befindet, ist über Rechnergrenzen hinweg nur Kommunikation in x-Richtung erforderlich. Im Fall b) befinden sich 10 Prozesse auf dem ersten Rechner und 14 Prozesse auf dem zweiten Rechner. Damit befindet sich kein Vielfaches der Anzahl der Prozessoren in y-Richtung auf einem Rechner und es muss zusätzlich zur Kommunikation in x-Richtung noch einmal in y-Richtung über Maschinengrenzen hinweg kommuniziert werden. Deshalb ist es wünschenswert für diesen Fall immer ein Vielfaches der Anzahl der Prozesse in y-Richtung auf die Rechner zu verteilen.

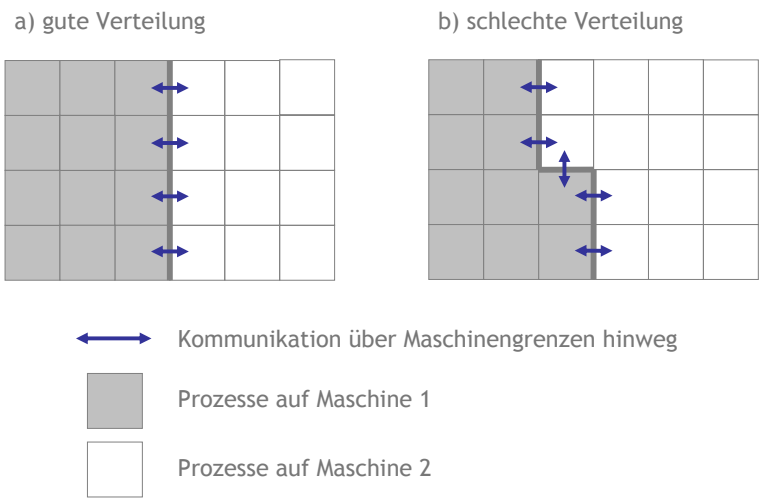

Abbildung 3.11.: Kommunikation bei unterschiedlicher Verteilung von Prozessen bei Verwendung von Blockstrukturen

Wenn Kenntnisse über das Kommunikationsmuster einer parallelen Anwendung zur Verfügung stehen, sollten die vom GCM verwendeten Heuristiken zusätzlich diese Werte in die Vorhersage einbeziehen, da es damit möglich ist, eine Grid Konfiguration zu finden, die besser an die Bedürfnisse der Anwendung angepasst ist. Der Algorithmus, der für die Heuristiken entwickelt wurde, enthält deshalb zusätzlich zu den oben genannten Eingangswerten eine Blockgröße für die Verteilung der Prozesse. Diese Blockgröße entspricht der Größe einer Blockstruktur innerhalb des Kommunikationsmusters der Anwendung, die in eine Richtung kommuniziert und muss vom Benutzer angegeben werden. Voraussetzung für den Einsatz dieser Methode ist somit, dass der Benutzer die Kommunikationsstruktur der Anwendung kennen muss.

Die für die Minimierung der Kommunikation optimale Verteilung der Prozesse hängt allerdings nicht nur von der Blockgröße ab. Vielmehr hat die Abbildung der Prozesse auf das kartesische Gitter ebenfalls einen entscheidenden Einfluss auf das Kommunikationsaufkommen. In Abb. 3.12 ist dieser Einfluss beispielhaft für eine 2-dimensionale Prozessverteilung dargestellt: 24 Prozesse sind in einem 6×4 Gitter angeordnet. Im Fall a) werden die Prozesse entlang der x-Achse verteilt, im Fall b) entlang der y-Achse. Die x-Achse entspricht hier der längeren Schnittkante und das bedeutet mehr Kommunikation über Maschinengrenzen hinweg. D. h., dass die Aufteilung b) der Aufteilung a) vorzuziehen ist. Für jeden dieser Fälle ergibt sich auch eine unterschiedliche optimale Blockgröße. Im Fall a) ist es 6 und im Fall b) ist es 4. Optimaler Weise sollten parallele Anwendungen, die in Grid Umgebungen verteilt ausgeführt werden, die Möglichkeit anbieten, die Verteilung der Prozesse in die einzelnen Dimensionen des kartesischen

Gitters dynamisch den Anforderungen der Laufzeitumgebung anzupassen.

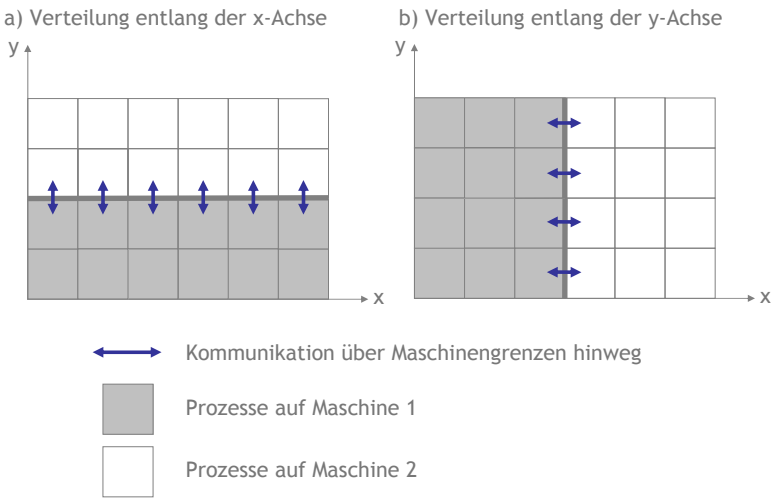

Abbildung 3.12.: Abbildung der Prozesse auf ein karthesisches Gitter und deren Einfluss auf die Kommunikation

Der vom GCM verwendete Algorithmus verteilt die benötigten Prozesse prinzipiell mit einem Round-Robin-Verfahren auf die zur Verfügung stehenden Maschinen. D. h. die zu Verfügung stehenden Prozessoren werden der Reihe nach belegt, wobei das Ziel ist, insgesamt so wenig Maschinen wie möglich zu verwenden. Diese Art der Verteilung führt erfahrungsgemäß zu optimalen Ausführungszeiten für verteilte, parallele Anwendungen, da die Kommunikation zwischen den Maschinen auf ein Minimum reduziert wird.

In Abb. 3.13 ist beispielhaft dargestellt, wie die Verteilung mit vorgegebener Blockgröße aussieht. Auf vier zur Verfügung stehenden Maschinen mit jeweils 6 Prozessoren sollen 15 Prozesse unter Beachtung der $Blockgröße = 5$ verteilt werden. Damit ergeben sich die folgenden Testkonfigurationen, von denen die ersten beiden in Abb. 3.13 dargestellt sind:

- je 5 Prozesse auf den Maschinen 1, 2 und 3
- je 5 Prozesse auf den Maschinen 2, 3 und 4
- je 5 Prozesse auf den Maschinen 3, 4 und 1
- je 5 Prozesse auf den Maschinen 4, 1 und 2

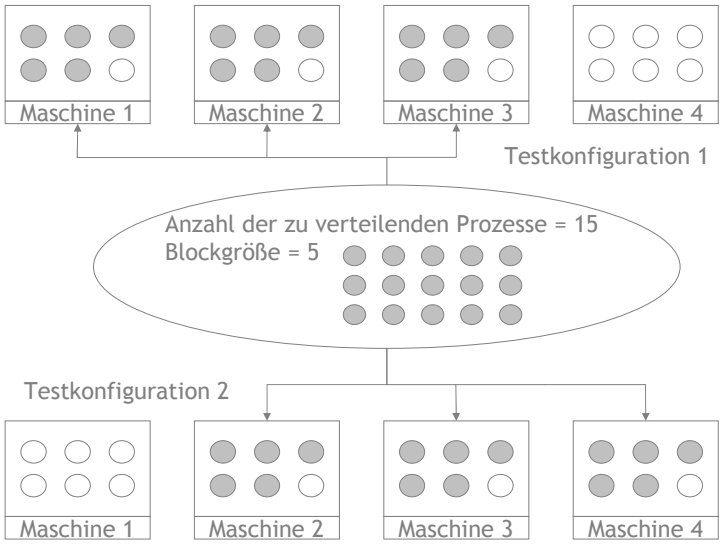

Abbildung 3.13.: Beispiel für die Verteilung von Prozessen für den Fall „Blockweise Verteilung“

Einfache Verteilung

Das Verteilungsprinzip „Einfache Verteilung“ stellt einen Sonderfall der vorher beschriebenen Blockverteilung für die $Blockgröße = 1$ dar. Es fließen somit keinerlei Informationen über die Kommunikationsmuster einer parallelen Anwendung in die Erstellung der Testkonfigurationen ein. Dementsprechend unspezifisch werden die Testkonfigurationen gewählt und die Vorhersage ermittelt nicht unbedingt die effizienteste Umgebung für die Anwendung.

Ein Beispiel für diese Art der Verteilung ist in Abb. 3.14 dargestellt. Auf vier zur Verfügung stehenden Maschinen mit jeweils 6 Prozessoren sollen 16 Prozesse verteilt werden. Damit ergeben sich die folgenden Testkonfigurationen, die mit Dimemas evaluiert werden sollen:

- je 6 Prozesse auf den Maschinen 1 und 2; 4 Prozesse auf Maschine 3
- je 6 Prozesse auf den Maschinen 2 und 3; 4 Prozesse auf Maschine 4
- je 6 Prozesse auf den Maschinen 3 und 4; 4 Prozesse auf Maschine 1
- je 6 Prozesse auf den Maschinen 4 und 1; 4 Prozesse auf Maschine 2

Die ersten beiden Testkonfigurationen sind in Abb. 3.13 dargestellt. Das Ergebnis der Dimemas-Vorhersagen wird von den Kommunikationsparametern (Latenz und Bandbreite) zwischen den Rechnern abhängen.

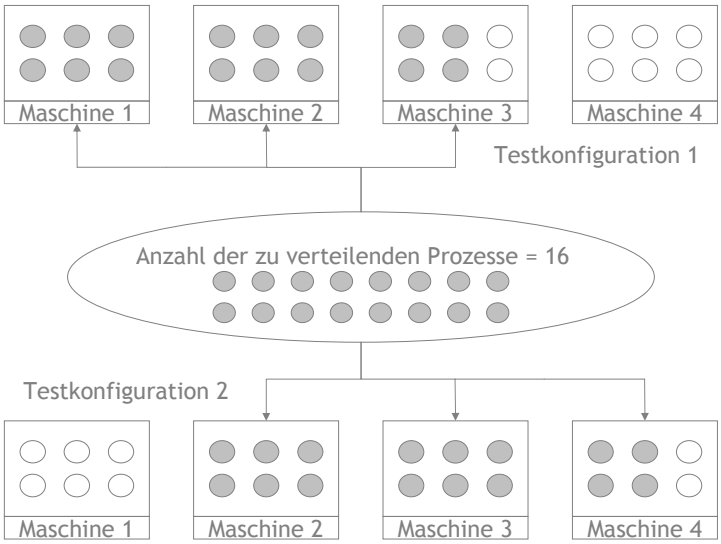

Abbildung 3.14.: Beispiel für die Verteilung von Prozessen für den Fall „Einfache Verteilung“

3.4.3.3. Der GCM als Entscheidungshilfe in der Software-Entwicklungsphase

Der GCM bietet noch eine dritte Möglichkeit für die Vorhersage an. Diese basiert auf der Annahme, dass für parallele Anwendungen z. T. verschiedene Algorithmen oder Implementierungen entwickelt werden, wenn sie für den Einsatz in verteilten Umgebungen konzipiert werden. Diese Anwendungen können angepasst werden, je nachdem in welcher Umgebung bzw. welcher Art der Verteilung (Konfiguration) sie ausgeführt werden sollen. Um diese Flexibilität der Anwendungen in die Vorhersage einzubeziehen, kann der Grid Configuration Manager verschiedene, vom Benutzer zur Verfügung gestellte, Tracedaten auswerten. Anhand der aktuell gewählten Konfiguration (Verteilung der Prozesse) kann damit ermittelt werden, welche(r) Implementierung/Algorithmus am Besten für diese Umgebung geeignet ist. Der Benutzer muss dazu für jede vorhandene Implementierung Tracedateien erzeugen und im GCM den Speicherort angeben. Der GCM erstellt dann automatisch für jede vorhandene Tracedatei Eingabedateien für Dimemas, führt eine Simulation durch und speichert die Ergebnisse, so dass ein Vergleich möglich ist.

Die Verwendung dieser Vorhersagemethode kann insbesondere während der Entwicklung von Anwendungen für Grid Umgebungen von Nutzen sein. Nach der Entwicklung verschiedener Versionen ihrer Programme testen Entwickler diese in der Regel für kleine und mittlere Problemgrößen auf einzelnen Parallelrechnern. Da für die Ausführung

dieser Anwendungen in Grid Umgebungen die Art der Verteilung eine große Rolle spielt (siehe Abschnitt „Blockverteilung“ auf Seite 65), muss man verschiedene Tests machen, um herauszufinden, welche Version der Anwendung in welcher Umgebung die besten Ergebnisse erzielt. Mit Hilfe des Grid Configuration Managers können Entwickler diese Aufgabe effizient durchführen und die dabei gewonnenen Kenntnisse in Form von gespeicherten Grid Konfigurationsdateien an die Benutzer weitergeben.

3.4.3.4. Speicherung von Daten in einem Metadaten Repository

Der Grundgedanke dieser Komponente basiert auf einem globalen Gedächtnis für die Wiederherstellung und Weiterverarbeitung von Daten. Diese Daten, Job Knowledge genannt, bestehen aus Informationen über die Anwendung selbst (Anwendungsbeschreibung), gesammelten Kenntnissen über Laufzeitverhalten der Anwendung (Ausführungszeiten) und einer Beschreibung von aktuellen/früheren Konfigurationen (Sessionbeschreibung) um Daten zu reproduzieren.

Es wurde eine Datenstruktur entworfen, in der alle Daten gesammelt werden. Die Klasse `PerfData` verwaltet diese Datenstruktur (siehe Abb. 3.15) und enthält die folgenden Komponenten:

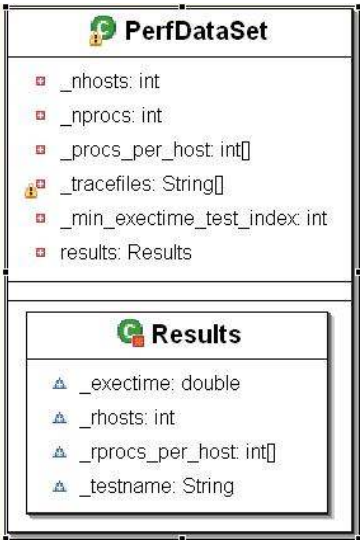

Abbildung 3.15.: Datenstruktur für das Abspeichern von Metadaten im GCM

- Anzahl der zur Verfügung stehenden Hosts
- Anzahl der Prozessoren, die die Anwendung verwenden möchte

- Anzahl der Prozessoren, die pro Host zur Verfügung stehen
- Tracedatei(en), die für die Vorhersage verwendet werden können
- Nummer des Tests mit der minimalen Ausführungszeit
- In der Struktur *Results* werden die Ergebnisse von Leistungsvorhersagen zusammengefasst.
  - Ausführungszeit
  - Anzahl der verwendeten Hosts
  - Anzahl der verwendeten Prozessoren pro Host
  - Name des Tests

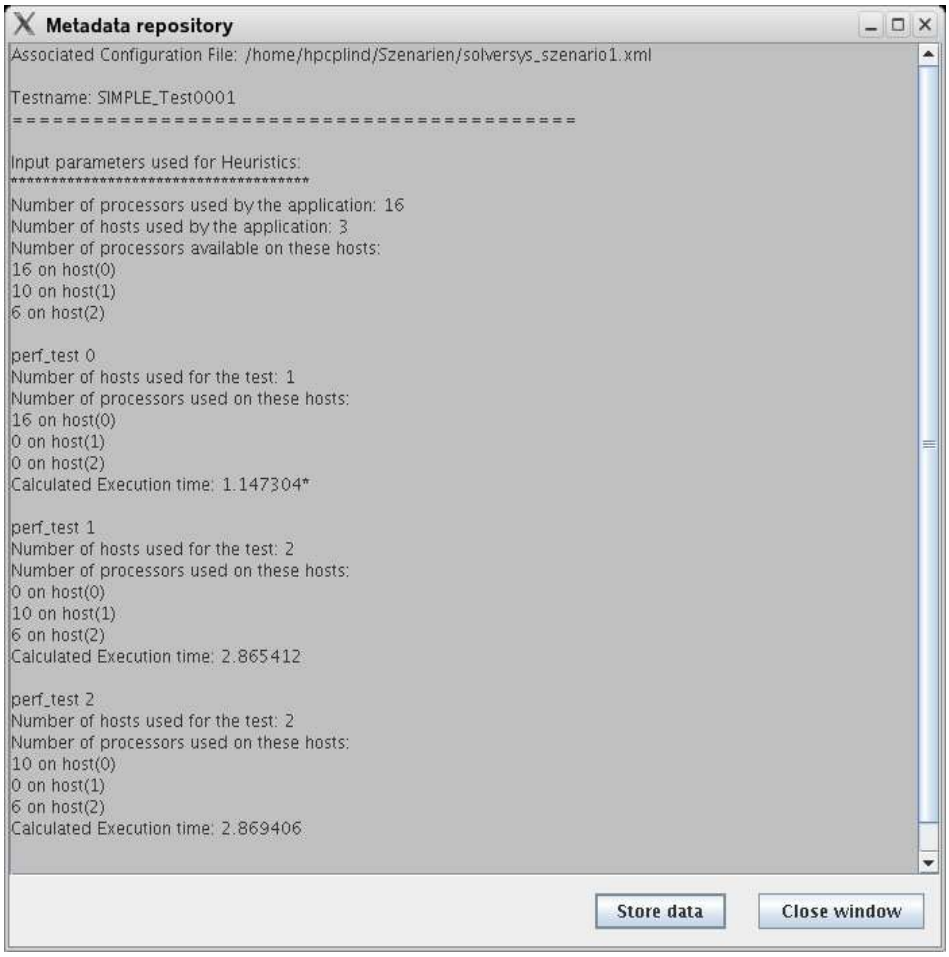

Abbildung 3.16.: Beispiel für die Ausgabe von Metadaten

In der zurzeit existierenden Implementierung des Grid Configuration Managers können diese Daten während der Durchführung von Vorhersageläufen gesammelt werden und vom Benutzer in einem Fenster angezeigt (siehe Abb. 3.16) und für die spätere Verwendung abgespeichert werden. Sie werden momentan noch nicht in die automatische Vorhersage einbezogen. Der Benutzer kann allerdings die gespeicherten Daten von Hand in die Konfigurationen einbauen und so wieder verwenden.

3.4.4. Starten und Überwachen der Anwendung

Für das Starten und Überwachen der Anwendung wurde ein spezielles Grid Terminal entworfen, das den Benutzer bei der Ausführung von Anwendungen in Grid Umgebungen unterstützt. Das Grid Terminal stellt eine „zentrale“ Benutzerschnittstelle für das Versenden von Kommandos an die entfernten Rechner und die gesammelte Ausgabe von Rückmeldungen dar. Benutzer müssen sich nicht mehr einzeln auf den verschiedenen Rechnern anmelden oder unterschiedliche Grid-Portale verwenden, um die Anwendungen zu starten und zu überwachen. Das Grid-Terminal verwendet dazu die Zugangsdaten, die in den einzelnen Host-Objekten der aktuellen Grid-Konfiguration gespeichert wurden.

Für die Ausführung von Kommandos gibt es zwei verschiedene Möglichkeiten:

1. Die Benutzung von Kommandos, die bei der Konfiguration jedes einzelnen *Hosts* eingegeben wurden, damit die Startkommandos für die Anwendung nicht jedes mal vom Benutzer neu eingegeben werden müssen. Da die Host-Objekte mit den Konfigurationen gespeichert werden, stehen diese Informationen auch für spätere Läufe zur Verfügung.
2. Die Ausführung von Kommandos, die direkt im Grid-Terminal eingegeben werden, so als würde der Benutzer an einer Konsole arbeiten. Damit können Pre- und Post-Processing Aufgaben ausgeführt werden, wie z. B. entfernte Dateien anschauen, um die Ergebnisse von Berechnungen zu überprüfen. Der Benutzer hat die Möglichkeit, Kommandos nur an ausgewählte Rechner oder gleichzeitig an alle zu verschicken.

3.4.5. Komponente für den Dateitransfer

Um den Benutzer bei Aufgaben, die Dateitransfer erfordern (Bereitstellung von Eingabedaten/Speicherung von Ausgabedaten) zu unterstützen, wurde eine spezielle grafische Komponente für den Dateitransfer entwickelt. Der Benutzer kann innerhalb dieser Oberfläche verschiedene Mechanismen für den Dateitransfer auswählen und auch untereinander kombinieren. Momentan werden die Protokolle scp, ftp und FTM (File Transfer

Management) [59] unterstützt. FTM ist ein Dateitransfermechanismus, der im Rahmen des UNICORE Projektes entwickelt wurde. Die Zugangsdaten für die einzelnen Rechner werden dabei aus den Hostobjekten, die sich in der aktuellen Grid Konfiguration befinden, automatisch übernommen um ein effizientes Arbeiten zu ermöglichen. Es ist aber auch möglich, zusätzliche Rechner für den Dateitransfer zu konfigurieren.

NIFTI (Network Interface to File Transfer in the Internet) [59] ist aufgrund seiner modularen Architektur vom Prinzip her unabhängig davon, wie die Services, die es darstellen kann, implementiert wurden. Dafür wurde ein objektorientiertes Implementierungskonzept entwickelt, das eine einfache Erweiterbarkeit und Integration der Komponente in andere Software-Projekte ermöglicht. Ausgehend von einem abstrakten Datei-Browser-Modell wurden für die o. g. Protokolle entsprechende Implementierungen für die Oberflächenkomponenten vorgenommen. Für den eigentlichen Dateitransfer wurde eine Schnittstelle *TransferHandler* entworfen, die dann für die einzelnen Protokolle implementiert wurde (siehe Abb. 3.17). Mit Hilfe eines integrierten Zwischenspeicherkonzeptes ist eine Kombination der Transfermechanismen möglich. Das ist z. B. sinnvoll, wenn Benutzer Daten zwischen Rechnern mit unterschiedlichen Zugangsmöglichkeiten transferieren wollen.

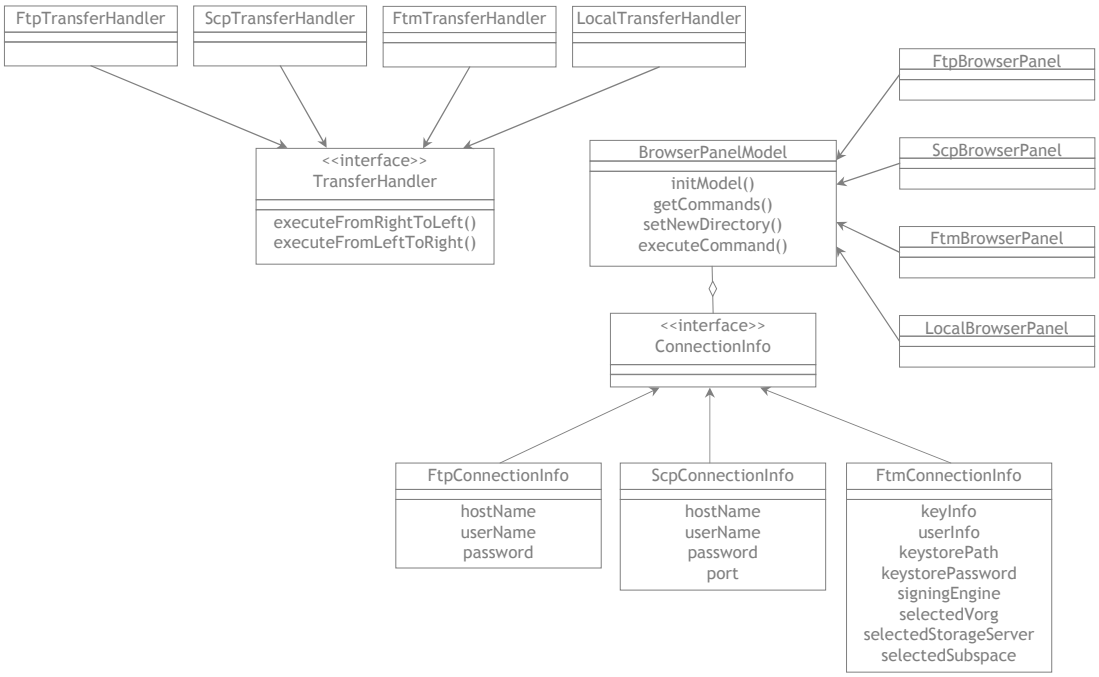

Abbildung 3.17.: Klassendiagramm der Dateitransferkomponente NIFTI

3.4.6. Ausführung von verteilten parallelen Anwendungen auf Basis von PACX-MPI

Der Grid Configuration Manager unterstützt den Startprozess von PACX-MPI-Jobs auf verschiedenen Rechnern während mehrerer Schritte und erleichtert damit die Handhabung von parallelen, verteilten Jobs signifikant. Die verschieden Schritte sind in Abb. 3.18 dargestellt.

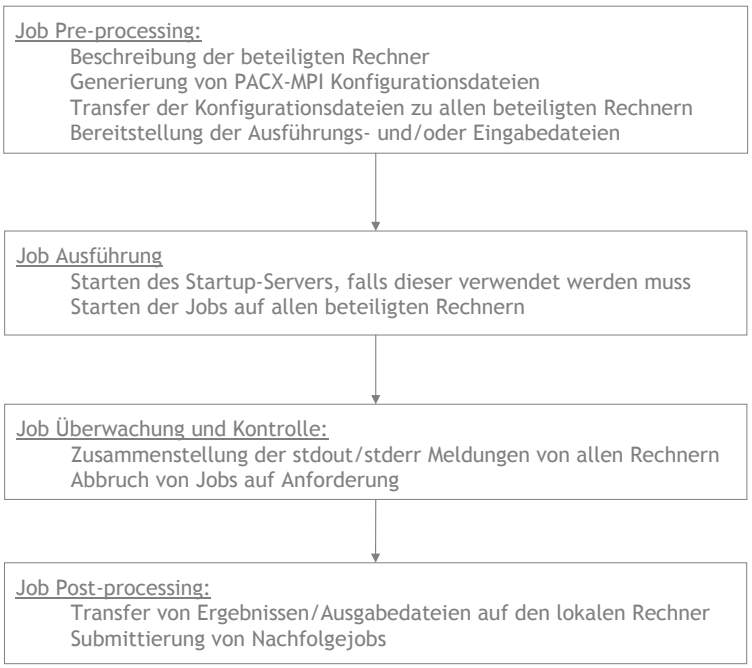

Abbildung 3.18.: Aufgaben, die GCM während der Ausführung von PACX-MPI-Jobs unterstützt

Zunächst kann der Benutzer mit Hilfe des Grid Configuration Managers die Umgebung beschreiben, in der er seine Anwendung ausführen will. Die von PACX-MPI benötigten Konfigurationsdateien werden dann automatisch erstellt und nach Aufforderung durch den Benutzer automatisch verschickt. Mit dem Grid Terminal können Benutzer die PACX-MPI-Jobs starten, überwachen und Ergebnisse abfragen.

3.4.6.1. Erstellen der PACX-Konfigurationsdateien

Nachdem der Benutzer die geplante Konfiguration mit Hilfe der *Host-* und *Linkobjekte* für seine Jobs beschrieben hat, generiert der GCM automatisch die PACX-MPI-Konfigurationsfiles in dem in Kap. 3.2.2 beschriebenen Format. Dafür werden die aktuell im PACX-Konfigurationsmodell zusammengefassten Daten benutzt. Falls der Benutzer mit einem speziellen *Hostobjekt* die Verwendung eines Startup-Servers anfordert, passt der GCM das Format des `hostfiles` an. Damit unterstützt der GCM zwei verschiedene Typen von Rechnern für PACX-MPI: Rechner mit Single-System-Image und Cluster.

Für die Erstellung der `netfiles` wurde zusätzlich die Möglichkeit für die Unterstützung von mehrfachen Netzwerkverbindungen (multiple connections) und Parameter für QoS Einstellungen, wie sie von PACX-MPI unterstützt werden, integriert. Dafür wurde die im Modell für *Links* vorhandene Klasse `Connection` entsprechend erweitert.

3.4.6.2. Starten von PACX-MPI-Anwendungen

Innerhalb des GCM wurde ein Startup-Server implementiert, der direkt von PACX-MPI verwendet werden kann, um das Starten von PACX-MPI auf clusterähnlichen Rechnern zu ermöglichen (siehe Kap. 3.2.2). Die Klasse `StartupServer` implementiert die Schnittstelle für Hostobjekte. Damit wird eine einfache Integration in das „Pacx-Modell“ ermöglicht.

Falls ein Benutzer während der Konfiguration der Grid Umgebung ein Startup-Server-Objekt hinzufügt, nimmt der GCM an, dass die interne Startup-Server Implementierung verwendet werden soll. Diese Standardeinstellung kann aber jederzeit überschrieben und ein Startup-Server, der auf einem anderen Rechner läuft, unter Angabe des Hostnamen oder der IP-Adresse konfiguriert werden. In diesem Fall wird der Startup-Server vom GCM als ein separater Job behandelt und automatisch vor dem eigentlichen Job gestartet. Die Ergebnisse, die vom Startup-Server auf der Standardausgabe `stdout` und der Standardfehlerausgabe `stderr` erzeugt werden, sammelt der GCM und zeigt sie in einem Ausgabefenster (Logging window) an.

Die Verteilung der PACX-Konfigurationsdateien soll erst erfolgen, wenn der Benutzer die gesamte Konfiguration fertig gestellt hat. Deshalb wurde dafür ein gesonderter Menüpunkt geschaffen. Die Verteilung selbst erfolgt automatisiert, damit die Konfiguration auf allen beteiligten Rechnern konsistent gehalten wird.

Mit dem in der Konfigurationsphase durch den Benutzer für jeden Host als „startup command“ angegebenen Kommando kann der GCM den verteilten Job auf allen Rechnern gleichzeitig starten (siehe Kap. 3.4.4. Da der GCM keine Implementierung für einen Scheduler verwendet, ist der Benutzer selbst dafür verantwortlich, dass zu diesem Zeitpunkt alle Ressourcen zur Verfügung stehen.

Die Ergebnisse werden im Grid Terminal in einem einzigen Terminalfenster ausgegeben, was die Überwachung der Gesamtanwendung sehr erleichtert, da der Benutzer nicht einzelne Terminalfenster für alle beteiligten Rechner öffnen muss.

3.5. Das UNICORE Metacomputing-Plugin

3.5.1. Konzept und Implementierung

Im Rahmen des UNICOREPlus Projektes [84] wurde am HLRS eine Erweiterung des UNICORE Klienten entwickelt, die es ermöglicht, PACX-MPI-Jobs in UNICORE Grids zu starten. Das Metacomputing-Plugin erstellt alle notwendigen Konfigurationsfiles für PACX-MPI und transferiert sie automatisch zu den vom Benutzer ausgewählten UNICORE Ressourcen. Für das Metacomputing-Plugin wurde eine im DAMIEN Projekt [38] realisierte erste Implementierung des Grid Configuration Managers verwendet.

Für die Implementierung wurde die Plugin-Schnittstelle von UNICORE benutzt, mit deren Hilfe, neue Funktionalitäten in den UNICORE Klienten eingebaut werden können, ohne Änderungen an der Basissoftware von UNICORE vorzunehmen [71]. Die Implementierung eines solchen Plugins gestaltet sich relativ unkompliziert, da man so viele Komponenten des UNICORE Klienten wieder verwenden kann. Im Falle das Metacomputing-Plugins wurde die Klasse „Task Plugin“ aus der UNICORE Plugin-Schnittstelle verwendet, mit deren Hilfe ein neuer Metacomputing-Job Task in den Klienten eingebaut wurde. Für das Task-Plugin mussten folgende Klassen implementiert werden:

- Die Klasse `MetacomputingPlugin`, die für das Starten des Plugins verantwortlich ist, die Ressourcen verwaltet und die Benutzereingaben bei Beendigung des Tasks abspeichert.
- Die Klasse `MetacomputingContainer`, die die Teile des AJOs konstruiert, die die Beschreibung des Metacomputing-Jobs enthalten.
- Die Klassen `MetacomputingJPAPanel` und `MetacomputingCanvas`, die die grafische Darstellung des Metacomputing-Plugins innerhalb des UNICORE Klienten ermöglichen.
- Die Klassen `MetacomputingDefaults` sowie `MetacomputingDefaultsDialog` die einen Konfigurationsdialog erzeugen, damit der Benutzer Einstellungen für das Plugin dauerhaft abspeichern kann.

Abb. 3.19 zeigt den prinzipiellen Ablauf bei der Ausführung des Metacomputing-Plugins in UNICORE. Mit Hilfe der o. g. Container Klasse ist der UNICORE Klient in der Lage, aus den Benutzereingaben für die ausgewählten Ressourcen (ResourceSet) und

dem Metacomputing-Task einen UNICORE Job in Form eines AJOs zu generieren. Dieser wird an den vom Benutzer ausgewählten NJS verschickt. Der NJS wandelt die Angaben innerhalb des AJO mithilfe der in der IDB vorhandenen Hard- und Softwarebeschreibungen in einen konkreten Job um, der an das Batchsystem der Zielrechner submittiert wird.

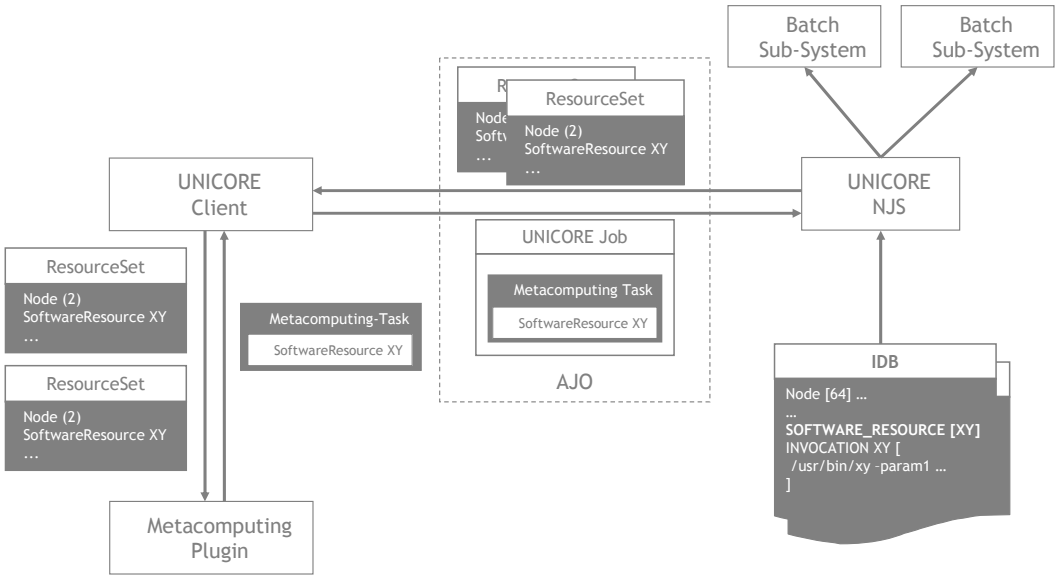

Abbildung 3.19.: Ausführung des Metacomputing Plugins in UNICORE

3.5.2. Verwendung

Das Metacomputing-Plugin besteht aus Java Klassen, die in einem signierten Jar-Archiv (Java Archivformat) zusammengefasst werden. Dieses Jar-Archiv muss vom Benutzer in ein bestimmtes Verzeichnis abgelegt werden, das vom UNICORE Klienten beim Starten durchsucht wird. Dadurch werden die Plugin Klassen zur Laufzeit dynamisch in den Klienten geladen. Wenn das Plugin geladen wurde, stehen dem Benutzer in der grafischen Benutzeroberfläche zusätzliche Komponenten zur Verfügung: ein neuer Metacomputing-Task und ein Konfigurationsdialog.

Wenn der Benutzer den neuen Metacomputing-Task auswählt, öffnet sich zuerst ein Dialog, mit dem er auswählen muss, bei welcher USite er seinen Job ausführen will. Danach öffnet sich eine spezielle grafische Oberfläche (siehe Bild 3.20) mit deren Hilfe der Benutzer Spezifikationen für seinen Metacomputing Job vornehmen kann. Dazu muss er als erstes die einzelnen Rechner auswählen, die er verwenden will. Als nächs-

tes muss er für jede Zielressource angeben, wo sich die Ausführungsdateien seiner PACX-MPI-Anwendungen befinden sowie die Anzahl der zu verwendenden Prozesse spezifizieren. Nachdem der Benutzer seinen Job submittiert hat, kann er mit Hilfe des Job Monitors den Status seines Jobs abfragen und sich die Standardausgaben sowie Fehlermeldungen (stdout/stderr) zurückliefern lassen. Falls gewünscht, kann man sich auch Ausgabedateien automatisch an bestimmte Speicherorte transferieren lassen.

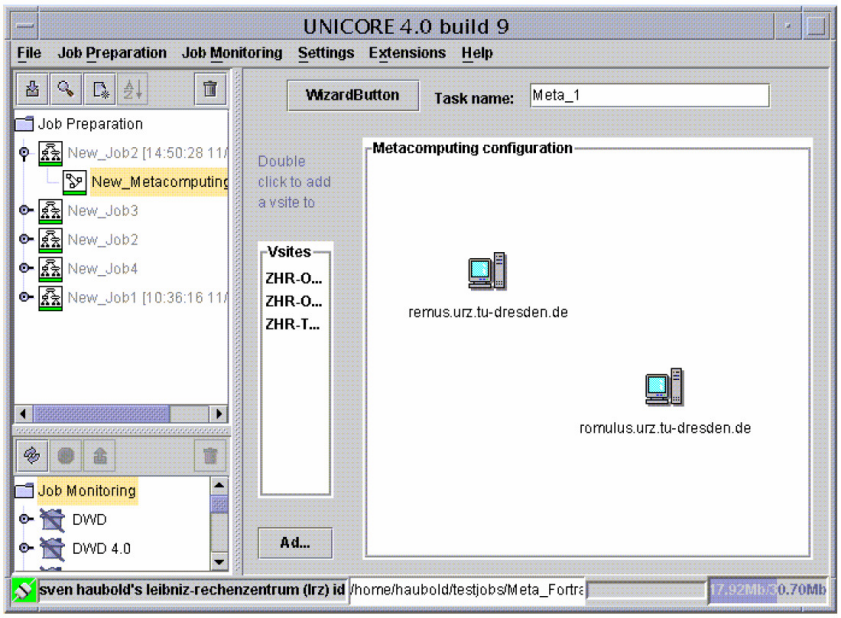

Abbildung 3.20.: Das UNICORE Metacomputing Plugin

Das Metacomputing-Plugin wurde von der TU Dresden erweitert, damit Unterstützung für die Erstellung und Auswertung von Tracedaten mit dem Vampir [13] Paket möglich ist. Vampir ist ein Programm für die Leistungsanalyse von MPI-parallelisierte Anwendungen. Mit Hilfe von Vampir kann sich ein Benutzer Tracing Informationen in einer Vielzahl von grafischen Ansichten anzeigen lassen, z. B. Zustandsdiagramme, Statistiken, Aktivitätsdarstellungen und Zeitlinien. Die dadurch gewonnenen Informationen können von Benutzern zur Optimierung seines parallelen Anwendungsprogramms eingesetzt werden. Das Plugin unterstützt den Benutzer bei der Auswahl von Parametern für die Erzeugung der Tracedaten mit dem Monitoring-Werkzeug Vampirtrace. Die Vampirtrace Konfigurationsfiles werden automatisch in die Arbeitsbereiche transferiert und die erzeugten Tracedaten zusammengefügt und an den Klienten zurückgeliefert. Das Starten von Vampir kann direkt innerhalb des UNICORE Klienten initialisiert werden. Der Vorteil dieser Lösung ist in einem verbesserten Arbeitsablauf für die Analyse von

Metacomputing Anwendungen zu sehen.

4. Untersuchung und Bewertung von Anwendungsszenarien des Grid Configuration Managers

„Es hängt vom Menschen ab, ob er die Technologie beherrscht oder sie ihn...“.
John Naisbitt

In diesem Kapitel wird der Einsatz des Grid Configuration Manager in verschiedenen Anwendungsszenarien gezeigt. Zunächst wird die erste erfolgreiche Umsetzung des GCM im Rahmen des europäischen Forschungsprojektes DAMIEN [38] vorgestellt.

Der Hauptteil dieses Kapitels widmet sich Untersuchungen der in Kapitel 3.4.3.2 vorgestellten Methoden zur automatischen Leistungsvorhersage mit dem GCM. In diesen Untersuchungen wird ein Programm zur Lösung von Gleichungssystemen, wie sie in ingenieurwissenschaftlichen Anwendungen häufig vorkommen, verwendet. Für die Bewertung der Funktionalitäten des GCM wurden verschiedene Szenarien entworfen. Diese und die damit erzielten Ergebnisse werden in Kap. 4.2 präsentiert.

4.1. Einsatz des GCM im DAMIEN Projekt

Im Rahmen des DAMIEN (Distributed Applications and Middleware for Industrial use of European Networks) Projektes [38] wurden eine Reihe von Werkzeugen für Grid Computing Umgebungen entwickelt. Diese Werkzeuge basierten auf zum Teil schon existierenden und weit verbreiteten Tools aus dem Bereich des Höchstleistungsrechnens. Alle diese Werkzeuge zusammen bilden eine Umgebung, die Anwendungsentwickler und -benutzer gleichermaßen unterstützen soll. Für die so genannte „Produktionsphase“ (Phase in der optimierter Programmcode für die Lösung von reellen Problemen zum Einsatz kommt) wurde eine erste Implementierung des Grid Configuration Managers entwickelt und eingesetzt.

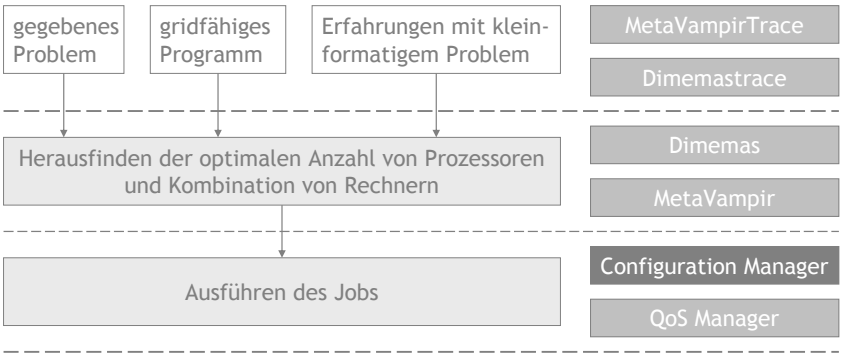

Abbildung 4.1.: Der Grid Configuration Manager im DAMIEN Projekt

Die Abbildung 4.1 zeigt einen strukturellen Überblick der Werkzeuge, wie sie in der Produktionsphase eingesetzt wurden. In diesem Szenario wird vorausgesetzt, dass der Endanwender eine Grid-fähige Version seines Programms besitzt. Außerdem wird davon ausgegangen, dass gewisse Erfahrungswerte und Tracedateien aus vorangegangen Läufen vorliegen. Für die Erstellung der Tracedateien kann z. B. die mit Dimemas vertriebene Tracing-Bibliothek mpidtrace oder MetaVampirtrace eingesetzt werden. Mit Hilfe von Dimemas und MetaVampir kann der Benutzer verschiedene Zielkonfigurationen simulieren und eine Entscheidung darüber treffen, welche Konfiguration für das getestete Problem am besten ist. Nachdem sich der Benutzer für eine Konfiguration entschieden hat, kann er mit Hilfe des Configuration Managers die Anwendung auf den Zielrechnern ausführen.

Im Rahmen von DAMIEN wurden nur einige Punkte des in Kapitel 3.4 beschriebenen Konzeptes verwirklicht [60], die im Folgenden aufgezählt werden:

- Generierung von Host- und Linkobjekten per Hand
- Generierung von Hostobjekten mit Hilfe einer Toolbox
- Öffnen/Speichern einer Konfiguration
- Export/Import einer Konfiguration
- Verschlüsselung von Passwörtern
- Erstellen von Konfigurationsdateien für PACX-MPI
- Verteilung der Konfigurationsdateien per ftp oder scp
- Unterstützung für die Beschreibung verschiedener Netzwerkverbindungen incl. IMPI-Protokoll
- Unterstützung für die Beschreibung von QoS-Parametern
- Startup-Mechanismus für PACX-MPI

- Starten und Überwachen der Anwendung per ssh

- Unterstützung des Zugangsmechanismus UNICORE

Mit der Realisierung des in Kapitel 3.4 beschriebenen Gesamtkonzeptes kann der oben beschriebene Ablauf zum Großteil automatisiert erfolgen, was zu einer deutlichen Reduzierung des Aufwandes für die Ermittlung der bestmöglichen Konfiguration und damit einer Zeiteinsparung für den Benutzer führt.

Der Grid Configuration Manager wurde bei verschiedenen Einsätzen im Rahmen des Projektes demonstriert. Zum einen wurde er in einem Szenario verwendet, bei dem unterschiedliche Rechner in Stuttgart/Deutschland, Rouen/Frankreich und Barcelona/Spanien verbunden wurden, um eine zwischen diesen Rechnern verteilte Simulationsanwendung auszuführen. Während der Vorbereitung dieser Demonstration wurde die Netzwerkverbindung zwischen den Rechnern ständig angepasst, um eine effiziente Anbindung der einzelnen Standorte an das neue Europäische Hochleistungsnetz Géant zu ermöglichen. Der Configuration Manager war dabei sehr nützlich, da damit das Starten der Anwendung mit modifizierten Konfigurationsdaten innerhalb von Sekunden möglich war. Damit konnte man schnelle Rückmeldungen an die Netzwerkfachleute über die Qualität der Verbindung liefern und somit Einfluss auf ihre Modifikationen nehmen. Die bei der Demonstration eingesetzte Anwendung stammt aus dem Bereich Strömungssimulation. URANUS [11] simuliert die Strömung um ein Raumfahrzeug bei der Wiedereintrittsphase. Der Einsatz des Configuration Managers war auch auf Grund einer technischen Eigenheit von URANUS besonders sinnvoll. URANUS erzeugt beim Kompilieren eine Verzeichnisstruktur, bei der die Ausführungsdateien für jede Zielarchitektur in einem gesonderten Verzeichnis abgelegt werden. Da bei der Demonstration unterschiedliche Architekturen (IBM RS6000SP und SGI Origin2000) verwendet wurden, konnte durch Verwendung des Configuration Managers die Startprozedur für die Anwendung auf den Rechnern sehr vereinfacht werden.

Des weiteren wurde der Grid Configuration Manager unter industriellen Bedingungen bei EADS (European Aeronautic Defense and Space Company) getestet. Als Anwendung kam dabei ein gekoppelter vibro-akkustischer Simulationscode zum Einsatz, der für die Lösung verschiedener Probleme verwendet werden kann (z. B. Simulationen für Geräuschreduzierung in Flugzeugen). Das Ziel von EADS war, diese Anwendung verteilt auf Ressourcen verschiedener Standorte durchzuführen, um möglichst realistische Problemgrößen simulieren zu können. Mit dem Grid Configuration Manager wurden Konfigurationen für das Starten der Anwendung erzeugt. Damit war es dann möglich, Berechnungen von Ingenieuren durchführen zu lassen, ohne dass diese mit der Komplexität der verteilten Grid Umgebung konfrontiert wurden.

4.2. Evaluierung des GCM unter Verwendung von parallelen Gleichungslösern für ingenieurwissenschaftliche Anwendungen

Die numerische Simulation technischer Probleme spielt eine immer größere Rolle in Industrie und Wissenschaft. Zeit- und kostenintensive Versuche können dadurch eingespart werden, dass mehr und mehr Computersimulationen durchgeführt werden, z. B. Crashtests in der Automobilindustrie. Viele dieser technischen und wissenschaftlichen Probleme werden durch große Systeme von Differential-, Integral- und Eigenwertgleichungen beschrieben. Die Lösung solcher großen, linearen Gleichungssysteme ist damit der Hauptbestandteil und gleichzeitig die zeit- und rechenintensivste Komponente der meisten ingenieurwissenschaftlichen Simulationen. Die Bezeichnung groß ist dabei als relativ anzusehen, da sie vom Verhältnis der Problemgröße zur Rechenleistung der zur Verfügung stehenden Rechner abhängt.

Für die Lösung solcher großen Gleichungssysteme wurden in den letzten Jahrzehnten eine Reihe von Gleichungslösern entwickelt. Um die Funktionalitäten des Grid Configuration Managers zu verifizieren, wird ein Programm verwendet, dass Implementierungen unterschiedlicher Gleichungslöser enthält [55].

Den Hintergrund der hier vorgestellten Tests bilden Überlegungen zum prognostizierbaren Verhalten des GCM bei Entscheidungen hinsichtlich der Auswahl der effizientesten Grid Konfiguration. Das Ziel ist die Bewertung der vom GCM ermittelten Vorschläge für die effizienteste Grid Konfiguration und nicht die Überprüfung der Genauigkeit des verwendeten Leistungsvorhersageprogrammes. Die vom GCM vorgeschlagene Grid Konfiguration sollte die folgenden Kriterien erfüllen:

(A) Der Ausführung einer Anwendung auf einer Maschine sollte immer der Vorzug vor der Ausführung in einer Grid Umgebung gegeben werden, da so die Kommunikation über Maschinengrenzen hinweg vermieden wird. D. h. falls die von der Anwendung benötigten Prozesse auf einer Maschine ausgeführt werden können, sollte die Konfiguration auch nur aus dieser einen Maschine bestehen.

(B) Die Kommunikation über Maschinengrenzen hinweg soll minimiert werden. Deshalb sollte die gewählte Grid Konfiguration aus der minimal notwendigen Anzahl von Maschinen bestehen.

(C) Die Grid Konfiguration sollte die Maschinen enthalten, die kommunikationstechnisch gesehen möglichst „nah beieinander“ liegen. Das bedeutet, dass die Maschinen ausgewählt werden, deren Verbindung(en) die bessere Bandbreite und/oder Latenz anbietet.

Damit die aufgeführten Punkte in einzelnen Tests verifiziert werden können, wurden verschiedene Szenarien entworfen, die in den folgenden Unterkapiteln detailliert dargestellt

werden. In Szenario 1 wird untersucht, ob der GCM die Grid Konfigurationen auswählt, die durch eine minimal mögliche Kommunikation zwischen Maschinen gekennzeichnet sind (Punkte A und B). Bei Szenario 2 werden die Kommunikationsparameter der Verbindungen in die Betrachtungen mit einbezogen (Punkt C).

Zusätzlich wird in einem weiteren Szenario die Anwendung des GCM als Entscheidungshilfe in der Programmentwicklung gezeigt wie es in Kap. 3.4.3.3 beschrieben wurde.

TRACE ERZEUGT AUF RECHNER	PROBLEMGRÖSSE	ANZAHL DER ITERATIONEN	ANZAHL DER PROZESSE	TRACE-DATEI
cacau.hww.de	$128 \times 128 \times 64$	25	16	16procs_Test1
cacau.hww.de	$256 \times 96 \times 48$	50	24	24procs_Test1
cacau.hww.de	$196 \times 128 \times 126$	25	48	48procs_Test1
shark.cs.uh.edu	$128 \times 128 \times 64$	25	16	16procs_Test1
shark.cs.uh.edu	$128 \times 128 \times 128$	25	32	32procs_Test1
shark.cs.uh.edu	$128 \times 128 \times 64$	50	16	qmr
shark.cs.uh.edu	$128 \times 128 \times 64$	50	16	qmr1
shark.cs.uh.edu	$128 \times 128 \times 64$	50	16	tfqmr
shark.cs.uh.edu	$128 \times 128 \times 64$	50	16	tfqmr1
shark.cs.uh.edu	$128 \times 128 \times 128$	50	32	qmr32
shark.cs.uh.edu	$128 \times 128 \times 128$	50	32	qmr132
shark.cs.uh.edu	$128 \times 128 \times 128$	50	32	tfqmr32
shark.cs.uh.edu	$128 \times 128 \times 128$	50	32	tfqmr132

Tabelle 4.1.: Übersicht der verwendeten Tracedateien

Für die Tests wurden zunächst Tracedaten für eine mittlere Problemgröße ($64 \times 64 \times 32$ Gitterpunkte pro Prozess) auf zwei verschiedenen Rechnern erzeugt, die die Grundlage für die Leistungsvorhersage mit Dimemas bilden sollen. Bei den verwendeten Rechnern handelt es sich um PC Cluster. *cacau.hww.de* besteht aus 200 Knoten, die jeweils mit zwei Intel Xeon EM64T Prozessoren (3.2GHz) bestückt sind und über 1 bzw. 2 GB Hauptspeicher verfügen. Die Knoten sind über Voltaire Infiniband und Gigabit Ethernet miteinander verbunden. *shark.cs.uh.edu* besitzt 24 Knoten, die jeweils über einen AMD Opteron DualCore-Prozessor (2,2 GHz) und 2 GB Hauptspeicher verfügen. Die Knoten sind über 4xInfiniBand und Gigabit Ethernet miteinander verbunden. Auf *cacau.hww.de* wurden die Traces unter Verwendung der MPI-Bibliothek VoltaireMPI mittels Einbindung der Bibliothek mpitrace [22] erzeugt. Auf auf *shark.cs.uh.edu* wurden die Tracedateien unter Verwendung der MPI-Bibliothek MVAPICH2 [48] und der Tracing Bibliothek Epilog [86] erzeugt. Die erstellten Tracedateien wurden mit entsprechenden Konvertern behandelt, um dem von Dimemas verwendeten Format zu entsprechen.

Tabelle 4.1 zeigt eine Zusammenstellung der für die Untersuchung verwendeten Tracedateien. Die Tracedateien wurde mit unterschiedlicher Anzahl von Prozessoren erzeugt, an die die Problemgröße jeweils angepasst wurde. Die Anzahl der Iterationen wurde bewusst klein gehalten, um die Größe der erzeugten Tracedatei zu minimieren. Die erzeugten Tracedateien hatten z. B. für die 32-Prozessor-Fälle eine durchschnittliche Größe von 11 MB.

Die für die Leistungsvorhersage benötigten Kommunikationsparameter (Latenz und Bandbreite) wurden mit PingPong Tests aus dem Intel MPI Benchmark Paket [52] ermittelt. In Tabelle 4.2 sind die im folgenden für die Simulation eingesetzten Parameter aufgeführt. Es werden zwei unterschiedliche Rechner simuliert, die jeweils aus 1-Prozessor-Knoten bestehen. In Szenario 3 werden auch 2-Prozessor-Knoten simuliert. Die verwendete Bandbreite für die Kommunikation zwischen den Knoten eines Rechners beträgt 750 MB/s bzw. 850 MB/s und die Latenz 4.7 µs bzw. 4.3 µs.

simulierter Rechner:	cacau.hww.de
Anzahl der verwendeten Prozessoren pro Knoten:	1
Bandbreite für Kommunikation zwischen den Knoten:	750 MB/s
Latenz für Kommunikation zwischen den Knoten:	4.7 µs
simulierter Rechner:	shark.cs.uh.edu
Anzahl der verwendeten Prozessoren pro Knoten:	1
Bandbreite für Kommunikation zwischen den Knoten:	850 MB/s
Latenz für Kommunikation zwischen den Knoten:	4.3 µs

Tabelle 4.2.: Verwendete Parameter für die Dimemas Simulation

4.2.1. Szenario 1 - Auswahl der Konfiguration mit minimalem Kommunikationsaufwand

In diesem Szenario soll nachgewiesen werden, dass der GCM die Punkte A und B der o. g. Liste erfüllt. Da hier eine Überprüfung der verwendeten Heuristiken und Entscheidungskriterien durchgeführt werden soll, werden folgende Vereinfachungen für die Durchführung der automatischen Leistungsvorhersage festgelegt:

- alle verwendeten Rechner besitzen den gleichen Typ von Prozessoren
- bei allen Rechnern sind die Kenndaten für die interne Kommunikation gleich
- die Kenndaten für die externe Kommunikation zwischen den Rechnern ist überall gleich.

Zunächst soll überprüft werden, ob der Grid Configuration Manager bei einer gegebenen Grid Konfiguration feststellt, dass die Ausführung der Anwendung auf genau

einer Maschine zu den besten Ergebnissen führen wird (Punkt A). Dazu wurden fünf verschiedene Testfälle entworfen. Die angebotene Zielkonfiguration besteht aus vier Maschinen. Die Aufteilung der auf diesen Maschinen vorhandenen Prozessoren wurde so gewählt, dass es für jeden Testfall möglich ist, nur eine Maschine zu benutzen. Für die Kommunikation zwischen den Rechnern wurde eine Bandbreite von 12 MB/s und eine Latenz von 4 ms zugrunde gelegt. Diese Werte entsprechen Verbindungsparametern wie sie z. B. zwischen zwei geografisch im Raum Deutschland liegenden Maschinen auftreten können. Die für die Simulation verwendeten Eingangsdaten der Tests sind in Tab. 4.3 zusammengefasst dargestellt.

Anzahl der zur Verfügung stehenden Rechner:	4				
Bandbreite zwischen den Rechnern:	12 MB/s				
Latenz zwischen den Rechnern:	4 ms				
	Test 1	Test 2	Test 3	Test 4	Test 5
Prozessoren auf Rechner 1:	16	6	6	16	6
Prozessoren auf Rechner 2:	6	24	6	6	32
Prozessoren auf Rechner 3:	6	6	48	6	6
Prozessoren auf Rechner 4:	6	6	6	6	6
verwendete Prozesse:	16	24	48	16	32
Tracedatei erzeugt auf:	cacau	cacau	cacau	shark	shark

Tabelle 4.3.: Eingangsdaten für die Vorhersagen in Szenario 1, Punkt A

Tabelle 4.4 enthält die Ergebnisse dieser ersten Tests (Szenario 1, Punkt A). Man kann erkennen, dass die bei der Simulation ermittelte Ausführungszeit für die Grid Konfiguration, bei der nur ein Rechner verwendet wird, jeweils am kürzesten ist. Die Ausführungszeiten für die Grid Konfigurationen, die aus zwei Rechnern bestehen, sind die zweit schnellsten. Die Ausführungszeiten für die Konfigurationen, die drei oder vier Maschinen verwenden, sind am langsamsten. Wenn die Verteilung der Prozesse gleich ist, wie z. B. in Test 1 (Konfiguration 2 und 3), sind auch die Ausführungszeiten gleich.

Es zeigt sich, dass das verwendete Verfahren formal zum korrekten Ergebnis führt, da die Ausführungszeiten am kürzesten sind, wenn nur eine Maschine in der Konfiguration verwendet wird. Der Grid Configuration Manager hat folgerichtig auch jeweils diese Konfigurationen als Vorschlag für die effizienteste Umgebung ausgewählt.

Für die Überprüfung, ob der GCM die Konfiguration auswählt, die eine möglichst geringe Anzahl von Maschinen verwendet (Punkt B), wurden die Testfälle so angepasst, dass die angebotenen Prozessoren eine unterschiedliche Verteilung der Anwendung erlauben. Es wird allerdings ausgeschlossen, dass die Anwendung nur auf einer Maschine

ausgeführt werden kann. Alle anderen Eingangsdaten entsprechen denen von Fall A. In Tab. 4.5 sind diese Daten zusammengefasst.

GETESTETE GRID-KONFIGURATION	ANZAHL DER PROZESSOREN AUF				SIMULIERTE AUSFÜHRUNGSZEIT (S)	
	RECHNER 1	RECHNER 2	RECHNER 3	RECHNER 4		
TEST 1: 16 PROZESSE „CACAU"-TRACE						
Konfiguration 1	16	0	0	0	1.175123	*
Konfiguration 2	0	6	6	4	7.415092	
Konfiguration 3	4	0	6	6	7.415092	
Konfiguration 4	10	0	0	6	5.138277	
TEST 2: 24 PROZESSE „CACAU"-TRACE						
Konfiguration 1	6	18	0	0	11.255319	
Konfiguration 2	0	24	0	0	1.682550	*
Konfiguration 3	6	6	6	6	13.907257	
Konfiguration 4	6	12	0	6	13.930373	
TEST 3: 48 PROZESSE „CACAU"-TRACE						
Konfiguration 1	6	6	36	0	8.092876	
Konfiguration 2	0	6	42	0	6.172948	
Konfiguration 3	0	0	48	0	1.299864	*
Konfiguration 4	6	6	3 0	6	8.811020	
TEST 4: 16 PROZESSE „SHARK"-TRACE						
Konfiguration 1	16	0	0	0	6.584126	*
Konfiguration 2	0	6	6	4	12.196775	
Konfiguration 3	4	0	6	6	12.197778	
Konfiguration 4	10	0	0	6	10.091078	
TEST 5: 32 PROZESSE „SHARK"-TRACE						
Konfiguration 1	6	26	0	0	8.712292	
Konfiguration 2	0	32	0	0	4.441051	*
Konfiguration 3	6	14	6	6	10.526176	
Konfiguration 4	6	20	0	6	10.520972	
* KENNZEICHNET DIE VOM GCM VORGESCHLAGENE ZIELKONFIGURATION						

Tabelle 4.4.: Ergebnisse Szenario 1, Punkt A

In Tab. 4.6 sind die Ergebnisse der zweiten Testreihe (Szenario 1, Punkt B) zusammengefasst dargestellt. Man kann erkennen, dass die simulierten Ausführungszeiten von Konfigurationen, die nur zwei Maschinen verwenden, kürzer sind, als die, die drei oder mehr Maschinen verwenden. Deshalb entsprechen auch hier die vom Grid Configuration Manager vorgeschlagenen Grid Konfigurationen den jeweils effizientesten Umgebungen für die Ausführung der Anwendung.

Anzahl der zur Verfügung stehenden Rechner:	4				
Bandbreite zwischen den Rechnern:	12 MB/s				
Latenz zwischen den Rechnern:	4 ms				
	Test 1	Test 2	Test 3	Test 4	Test 5
Prozessoren auf Rechner 1:	10	10	10	10	10
Prozessoren auf Rechner 2:	6	12	24	6	12
Prozessoren auf Rechner 3:	6	14	26	6	22
Prozessoren auf Rechner 4:	4	2	12	4	4
Anwendungsprozesse:	16	24	48	16	32
Trace erzeugt auf:	cacau	cacau	cacau	shark	shark

Tabelle 4.5.: Eingangsdaten für die Vorhersagen in Szenario 1, Punkt B

4.2.2. Szenario 2 - Variation der Kommunikationsparameter zwischen den Rechnern

In diesem Szenario wird untersucht, welchen Einfluss die Qualität der Kommunikation zwischen den Rechnern auf die Vorhersagen des GCM hat. Dazu wurden Testfälle entworfen, bei denen unterschiedliche Variationen der Kommunikationskenngrößen Bandbreite und Latenz zwischen den Rechnern untersucht werden. Als Ergebnis sollte der Grid Configuration Manager diejenige Konfiguration auswählen, bei der die beteiligten Rechner über die beste Kommunikation untereinander verfügen (Punkt C).

Es wurden keine Tests gemacht, bei denen gleichzeitig die Bandbreite und die Latenz variiert wurden, da die Ergebnisse nicht verifizierbar sind. Es gibt in der Regel keine Möglichkeit, die Kommunikationsparameter zwischen zwei Rechnern an feste Werte anzupassen und Messungen durchzuführen, um diese mit den in der Vorhersage ermittelten Ausführungszeiten zu vergleichen. Für das hier gezeigte Szenario hätten sich keine neuen Erkenntnisse ergeben.

Es gibt noch eine weitere Beschränkung, die bei den Untersuchungen dieses Szenarios beachtet werden musste. Bei der derzeit zur Verfügung stehenden Version von Dimemas

muss der Benutzer bei Verwendung der dedizierten Verbindungen angeben, welche Nachrichten über diese Verbindungen geschickt werden sollen. Dazu muss er eine Liste von MPI-Nachrichten-Tags bei der Konfiguration angeben.

GETESTETE GRID-KONFIGURATION	ANZAHL DER PROZESSOREN AUF RECHNER 1	RECHNER 2	RECHNER 3	RECHNER 4	SIMULIERTE AUSFÜHRUNGSZEIT (S)	
TEST 1: 16 PROZESSE „CACAU"-TRACE						
Konfiguration 1	10	6	0	0	5.135505	*
Konfiguration 2	0	6	6	4	7.415092	
Konfiguration 3	6	0	6	4	7.413611	
Konfiguration 4	10	2	0	4	7.420782	
TEST 2: 24 PROZESSE „CACAU"-TRACE						
Konfiguration 1	10	12	2	0	14.357081	
Konfiguration 2	0	12	12	0	9.59947	*
Konfiguration 3	8	0	14	2	14.266122	
Konfiguration 4	10	12	0	2	13.81217	
TEST 3: 48 PROZESSE „CACAU"-TRACE						
Konfiguration 1	10	24	14	0	9.683969	
Konfiguration 2	0	24	24	0	6.095392	*
Konfiguration 3	10	0	26	12	10.021826	
Konfiguration 4	10	24	2	12	10.093805	
TEST 4: 16 PROZESSE „SHARK"-TRACE						
Konfiguration 1	10	6	0	0	10.088311	*
Konfiguration 2	0	6	6	4	12.196775	
Konfiguration 3	6	0	6	4	12.228386	
Konfiguration 4	10	2	0	4	12.235293	
TEST 5: 32 PROZESSE „SHARK"-TRACE						
Konfiguration 1	10	12	10	0	10.697889	
Konfiguration 2	0	12	20	0	7.997231	*
Konfiguration 3	6	0	22	4	11.229926	
Konfiguration 4	10	12	6	4	10.473372	
* KENNZEICHNET DIE VOM GCM VORGESCHLAGENE ZIELKONFIGURATION						

Tabelle 4.6.: Ergebnisse Szenario 1, Fall B

Es gibt keine Möglichkeit anzugeben, dass alle Nachrichten, die zwischen zwei Rechnern

ausgetauscht werden, diese Verbindung benutzen sollen. Da aber genau dieses Feature von besonderem Interesse für die Vorhersagen in Grid Umgebungen ist, wurde in den Grid Configuration Manager eine zusätzliche Konfigurationsmöglichkeit eingebaut, um diese Beschränkung zumindest teilweise zu umgehen. Der Benutzer kann im GCM bei der Beschreibung von dedizierten Verbindungen zusätzliche Werte für Bandbreite und Latenz dieser Verbindung angeben. Der GCM erkennt dann automatisch, dass er speziell angepasste Konfigurationsdateien für Dimemas erstellen soll. Da bei dieser Vorgehensweise auf den Einsatz der dedizierten Verbindungen für die Simulation verzichtet wird, kann diese Methode allerdings nur Vorhersagen für Umgebungen erstellen die genau aus zwei Rechnern bestehen. Bei der Vorhersage wird von der in Dimemas enthaltenen Beschreibung einer Umgebung mittels WAN Gebrauch gemacht, um die Bandbreite einfließen zu lassen. Die Latenz zwischen den Rechnern wird aus der Beschreibung der einzelnen Rechner gewonnen.

Anzahl der zur Verfügung stehenden Rechner:	4				
	Test1	Test 2	Test 3	Test 4	Test 5
Prozessoren auf Rechner 1:	8	12	24	8	16
Prozessoren auf Rechner 2:	8	12	24	8	16
Prozessoren auf Rechner 3:	8	12	24	8	16
Prozessoren auf Rechner 4:	8	12	24	8	16
Anwendungsprozesse:	16	24	48	16	32
Trace erzeugt auf:	cacau	cacau	cacau	shark	shark
Bandbreite zwischen Rechner 1 und 2:	100 MB/s				
Bandbreite zwischen Rechner 2 und 3:	25 MB/s				
Bandbreite zwischen Rechner 3 und 4:	30 MB/s				
Bandbreite zwischen Rechner 4 und 1:	95 MB/s				
Latenz zwischen allen Rechnern:	4 ms				

Tabelle 4.7.: Eingangsdaten für die Vorhersagen in Szenario 2, Bandbreite variiert

Die Eingangskonfigurationen, die für die Tests in diesem Szenario entworfen wurden, verwenden jeweils vier Rechner. Zunächst wird eine Variation der Bandbreiten zwischen den Rechnern vorgenommen, die Latenz zwischen den Rechnern beträgt immer 4 ms (siehe Tab. 4.7). Die Bandbreiten zwischen den Rechnern wurden so gewählt, dass sie einer bestimmten Verteilung entsprechen. Drei Rechner liegen relativ „nah“ beieinander, was durch eine hohe Bandbreite zwischen den Rechnern ausgedrückt wird. Die Verbindungen zu Rechner 3 sind durch eine niedrigere Bandbreite gekennzeichnet, was z. B. einem geografisch weiter entfernten Rechner entspricht. Die Verteilung der ange-

botenen Prozessoren wurde so gewählt, dass die Anwendung jeweils zwei Maschinen für die Ausführung verwenden muss, um die eben beschriebene Beschränkung des verwendeten Modells zu erfüllen.

Die Ergebnisse dieser Tests sind in Tab. 4.9 zu sehen. Man kann erkennen, dass die simulierten Ausführungszeiten, bei denen Rechner verwendet werden, die die Verbindung mit der höchsten Bandbreite benutzen (zwischen Rechner 1 und 2) am kürzesten sind. Die Rechner, die durch die zweitschnellste Verbindung gekennzeichnet sind, erreichen auch die zweitschnellste Ausführungszeit u. s. w. Weiterhin ist zu sehen, dass die Simulationsresultate auch relative kleine Unterschiede in den Parametern widerspiegeln, wie man jeweils an den Konfigurationen 2 und 3 erkennen kann. In diesen Konfigurationen wird eine Bandbreite von 25 bzw. 30 MB/s verwendet.

Anzahl der zur Verfügung stehenden Rechner:	4				
	Test1	Test 2	Test 3	Test 4	Test 5
Prozessoren auf Rechner 1:	8	12	24	8	16
Prozessoren auf Rechner 2:	8	12	24	8	16
Prozessoren auf Rechner 3:	8	12	24	8	16
Prozessoren auf Rechner 4:	8	12	24	8	16
Anwendungsprozesse:	16	24	48	16	32
Trace erzeugt auf:	cacau	cacau	cacau	shark	shark
Latenz zwischen Rechner 1 und 2:	4 ms				
Latenz zwischen Rechner 2 und 3:	5 ms				
Latenz zwischen Rechner 3 und 4:	0.6 ms				
Latenz zwischen Rechner 4 und 1:	0.8 ms				
Bandbreite zwischen allen Rechnern:	12 MB/s				

Tabelle 4.8.: Eingangsdaten für die Vorhersagen in Szenario 2, Latenz variiert

In Tabelle 4.8 sind die Eingangsdaten der nächsten Tests zu sehen. Die Latenz zwischen den Rechnern wird variiert und die Bandbreite konstant gewählt. Auch hier wird versucht, durch entsprechende Werte der Latenz eine Umgebung nachzubilden, bei der ein Rechner geografisch weiter entfernt zu sein scheint (Rechner 2), was durch eine höhere Latenz der Verbindung zu diesem Rechner ausgedrückt wird (4 bzw. 5 ms).

Die Ergebnisse dieser Tests (siehe Tab. 4.10) zeigen, dass die simulierten Ausführungszeiten die Unterschiede in der Verbindungsqualität widerspiegeln. Die Grid Konfiguration, deren Verbindung durch die geringste Latenz gekennzeichnet ist, erreicht die beste Ausführungszeit. Weiterhin ist auch hier die Staffelung der gewählten Verbindungspara-

meter an den Ergebnissen zu erkennen. Der Grid Configuration Manager ist somit in der Lage zu erkennen, wenn Rechner durch eine gute Netzwerkverbindung gekennzeichnet sind und trifft eine richtige Entscheidung hinsichtlich der effizientesten Grid Konfiguration.

Getestete Grid-konfiguration	Anzahl der Prozessoren auf				Simulierte Ausführungszeit (s)
	Rechner 1	Rechner 2	Rechner 3	Rechner 4	
Test 1: 16 Prozesse „cacau"-Trace					
Konfiguration 1	8	8	0	0	3.702515 *
Konfiguration 2	0	8	8	0	3.795772
Konfiguration 3	0	0	8	8	3.705274
Konfiguration 4	8	0	0	8	3.702516
Test 2: 24 Prozesse „cacau"-Trace					
Konfiguration 1	12	12	0	0	6.60162 *
Konfiguration 2	0	12	12	0	7.361563
Konfiguration 3	0	0	12	12	7.052505
Konfiguration 4	12	0	0	12	6.601623
Test 3: 48 Prozesse „cacau"-Trace					
Konfiguration 1	24	24	0	0	4.934447 *
Konfiguration 2	0	24	24	0	4.93591
Konfiguration 3	0	0	24	24	4.934503
Konfiguration 4	24	0	0	24	4.934448
Test 4: 16 Prozesse „shark"-Trace					
Konfiguration 1	8	8	0	0	9.086711 *
Konfiguration 2	0	8	8	0	9.109551
Konfiguration 3	0	0	8	8	9.087665
Konfiguration 4	8	0	0	8	9.08673
Test 5: 32 Prozesse „shark"-Trace					
Konfiguration 1	16	16	0	0	6.937863 *
Konfiguration 2	0	16	16	0	6.958964
Konfiguration 3	0	0	16	16	6.939202
Konfiguration 4	16	0	0	16	6.937885
* kennzeichnet die vom GCM vorgeschlagene Zielkonfiguration					

Tabelle 4.9.: Ergebnisse Szenario 2, Variation der Bandbreite zwischen den Rechnern

GETESTETE GRID-KONFIGURATION	ANZAHL DER PROZESSOREN AUF RECHNER 1	RECHNER 2	RECHNER 3	RECHNER 4	SIMULIERTE AUSFÜHRUNGSZEIT (S)	
TEST 1: 16 PROZESSE „CACAU"-TRACE						
Konfiguration 1	8	8	0	0	4.465478	
Konfiguration 2	0	8	8	0	4.973478	
Konfiguration 3	0	0	8	8	2.83186	*
Konfiguration 4	8	0	0	8	2.92286	
TEST 2: 24 PROZESSE „CACAU"-TRACE						
Konfiguration 1	12	12	0	0	9.59947	
Konfiguration 2	0	12	12	0	10.480678	
Konfiguration 3	0	0	12	12	6.612363	*
Konfiguration 4	12	0	0	12	6.787963	
TEST 3: 48 PROZESSE „CACAU"-TRACE						
Konfiguration 1	24	24	0	0	6.095392	
Konfiguration 2	0	24	24	0	6.651931	
Konfiguration 3	0	0	24	24	4.514996	*
Konfiguration 4	24	0	0	24	4.605396	
TEST 4: 16 PROZESSE „SHARK"-TRACE						
Konfiguration 1	8	8	0	0	9.593537	
Konfiguration 2	0	8	8	0	10.123205	
Konfiguration 3	0	0	8	8	7.863483	*
Konfiguration 4	8	0	0	8	7.960203	
TEST 5: 32 PROZESSE „SHARK"-TRACE						
Konfiguration 1	16	16	0	0	7.414597	
Konfiguration 2	0	16	16	0	7.948321	
Konfiguration 3	0	0	16	16	5.660996	*
Konfiguration 4	16	0	0	16	5.759201	
* KENNZEICHNET DIE VOM GCM VORGESCHLAGENE ZIELKONFIGURATION						

Tabelle 4.10.: Ergebnisse Szenario 2, Variation der Latenz zwischen den Rechnern

4.2.3. Szenario 3 - Testen verschiedener Implementierungen einer Anwendung

Wie in Kap. 3.4.3.3 beschrieben, existieren für bestimmte Anwendungen, die für Grid Umgebungen entworfen oder angepasst wurden, mehrere Implementierungen, die in der Regel durch unterschiedliche Kommunikationsmuster gekennzeichnet sind. In Szenario 3 wird der Grid Configuration Manager dazu verwendet, aus verschiedenen Implementierungen diejenige Alternative herausfinden, die in einer feststehenden Grid Konfiguration am effizientesten ausgeführt werden kann. Der GCM wertet dazu die Ausführungszeiten der durch verschiedene Tracedateien repräsentierten Versionen einer Anwendung mittels Simulation aus.

Die Entscheidungsbasis für dieses Szenario bilden Implementierungen von Gleichungslösern, wie sie z. B. in vielen ingenieurwissenschaftlichen Anwendungen vorkommen. Um die Ergebnisse der Untersuchungen in diesem Kapitel besser zu verstehen, werden an dieser Stelle die den hier verwendeten Implementierungen zugrunde liegenden Algorithmen kurz erläutert.

Es werden je zwei Implementierungen des so genannten QMR- und TFQMR-Verfahrens miteinander verglichen. Diese Verfahren gehören alle zur Gruppe der iterativen Lösungsverfahren. Bei diesen Verfahren wird versucht, ausgehend von einer Anfangsnäherung, eine Folge von Vektoren zu erzeugen, die schrittweise gegen die gesuchte Lösung des Gleichungssystems konvergiert, d. h. sie liefern eine Näherungslösung.

Die Grundlage für das QMR-Verfahren [34] bildet der so genannte unsymmetrische Lanczos-Algorithmus. Die Näherungslösung wird über eine Quasi-Minimierung der Residuumsnorm bestimmt, daher auch der Name QMR. Der verwendete Algorithmus enthält in jeder Iteration sowohl ein Matrix-Vektor-Produkt mit der Koeffizientenmatrix des zu lösenden Gleichungssystems als auch ein Matrix-Transponiert-Vektor-Produkt. Da dieses Vorgehen kommunikationstechnisch aufwendig ist, wurde eine transponierungsfreie Modifikation dieses Verfahrens entwickelt, das TFQMR-Verfahren (Transpose Free QMR) [33]. TFQMR berechnet in jeder Iteration zwei Matrix-Vektor-Produkte mit der Koeffizientenmatrix. Während die beiden Matrix-Vektor-Produkte in QMR unabhängig von einander berechnet werden können, sind die Matrix-Vektor-Produkte bei der Methode TFQMR voneinander abhängig.

Die hier für das QMR-Verfahren verwendete Implementierung (`QMR`) ist eine in [15] vorgestellte Version, in welcher der Lanczos-Algorithmus so umgestellt wurde, dass nur noch ein globaler Synchronisationspunkt vorhanden ist. In der zweiten hier verwendeten Implementierung ($\mathtt{QMR}_1$) wird im Gegensatz zur ersten Variante der Parameter für die Wichtung $\omega_i = 1$ gesetzt. Außerdem gibt es in der Version $\mathtt{QMR}_1$ drei Synchronisationspunkt.

Auch für das TFQMR-Verfahren werden zwei unterschiedlich implementierte Versionen

verwendet. Der Algorithmus für die Version TFQMR ist aus [33] entnommen. Der Algorithmus für $\mathtt{TFQMR}_1$ wird in [14] vorgestellt und verwendet im Unterschied zu TFQMR die euklidische Norm als Basis für die Ermittlung des minimalen Residuums.

Bei der parallelen Implementierung dieser Verfahren gilt die besondere Aufmerksamkeit allen Operationen, die Kommunikation erforderlich machen. Es ist ein Dilemma der parallelen Programmierung, dass einerseits die (sequentiellen) Algorithmen bei der exakten parallelen Implementierung Kommunikation erfordern, man aber andererseits Datenaustausch zwischen den Prozessoren so weit wie möglich vermeiden möchte, da dieser relativ teuer ist [37]. Dieser Faktor spielt im Hinblick auf die Ausführung der Anwendung in verteilten Grid Umgebungen eine noch größere Rolle. Tabelle 4.11 zeigt einen Vergleich hinsichtlich des rechentechnischen Aufwandes der verwendeten Implementierungen. Die Daten dieser Tabelle sind aus [37] entnommen. Einen Einfluss auf das Kommunikationsaufkommen hat die in der 1. Zeile aufgeführte Anzahl der Matrix-Vektor Multiplikationen. Sie beinhalten normalerweise eine Kommunikation zwischen benachbarten Prozessen. Die in der 2. Zeile aufgeführte Anzahl der notwendigen Matrix-Transponiert-Vektor Multiplikationen enthalten ebenfalls Kommunikation mit den Nachbarprozessen. Kommunikation zwischen allen Prozessoren erfordern die in der 4. Zeile aufgeführten globalen Synchronisationspunkte, die aufgrund der in den Algorithmen auftretenden Zusammenfassung von Operationen für die Berechnung der Skalarprodukte und Beträge von Vektoren notwendig sind. Die in der Tabelle aufgeführten Unterschiede in den einzelnen Implementierungen sollten sich in den Vorhersagen des Grid Configuration Managers widerspiegeln.

	QMR	QMR_1	TFQMR	$TFQMR_1$
Matrix-Vektor Multiplikationen	1	1	3	3
Matrix-Transponiert-Vektor Multiplikationen	1	1	0	0
Skalarprodukte	4	4	4	4
Synchronisationspunkte	1	3	4	4

Tabelle 4.11.: Vergleich der verwendeten Implementierungen von Gleichungslösern

In den folgenden Tests wurden je zwei Implementierungen miteinander verglichen. Die Grid Konfigurationen für die Vorhersage bestehen jeweils aus zwei Rechnern. Die Anwendung verwendet je nach Test 16 bzw. 32 Prozessoren. Als Parameter für die Beschreibung der Rechner wurden die in der Einleitung dieses Kapitels vorgestellten Daten des Rechners *shark.cs.uh.edu* benutzt. Für die 16-Prozessor-Tests wird in der Simulation von 1-Prozessor-Knoten ausgegangen, die 32-Prozessor-Tests verwenden 2-Prozessor-Knoten. Eine Zusammenfassung der gewählten Eingangskonfigurationen für die Vorhersagen mit dem GCM ist in Tab. 4.12 zu sehen.

Zur Verifizierung der Simulationsresultate wurden die Ausführungszeiten der reellen

Anzahl der zur Verfügung stehenden Rechner:	2			
Bandbreite zwischen den Rechnern:	80 MB/s			
Latenz zwischen den Rechnern:	0.2 ms			
	Test1	Test 2	Test 3	Test 4
Prozessoren auf Rechner 1:	8	8	16	16
Prozessoren auf Rechner 2:	8	8	16	16
Anwendungsprozesse:	16	16	32	32
verwendete Tracedateien:	qmr.trf	tfqmr.trf	qmr32.trf	tfqmr32.trf
	qmr1.trf	tfqmr1.trf	qmr132.trf	tfqmr132.trf
Prozessoren pro Knoten	1	1	2	2

Tabelle 4.12.: Eingangsdaten für die Vorhersagen in Szenario 3

Anwendung gemessen. Dazu wurde die Anwendung zunächst mit der Bibliothek PACX-MPI übersetzt. Für die Messungen wurde eine Testumgebung gewählt, die eine einfache Nachvollziehbarkeit der Messergebnisse und damit auch eine Bewertung der Simulationsergebnisse ermöglicht. Deshalb wurden die Messungen der Ausführungszeiten für dieses Szenario auf einem einzelnen Rechner (*shark.cs.uh.edu*) durchgeführt. Für die Nachbildung einer verteilten Grid Umgebung verwendet die Anwendung zwei Teilbereiche der Maschine, die jeweils einen einzelnen Rechner repräsentieren. Für jeden Testfall wurden fünf Messungen gemacht.

In den Tabellen 4.13 und 4.14 sind die bei der Simulation ermittelten und die gemessenen Ausführungszeiten gegenübergestellt. Tab. 4.13 zeigt die Resultate für die Ausführung mit 16 Prozessoren, Tab. 4.14 für 32 Prozessoren. Man kann zunächst erkennen, dass die Streuweite der gemessenen Werte den für solche Umgebungen üblichen Werten entspricht. Die bei der Simulation ermittelten Werte liegen für die Tests 1 und 2 (Ausführung auf 16 Prozessoren) im Bereich der gemessenen Werte. Eine Ausnahme bildet der Test, der die Implementierung $\texttt{QMR}_1$ verwendet. Für die Tests 3 und 4 (Ausführung auf 32 Prozessoren) weichen sie von den gemessenen Werten ab.

Die gemessenen Ausführungszeiten für `QMR` und $\texttt{QMR}_1$ spiegeln die in den Implementierungen vorhandenen kommunikationstechnischen Unterschiede wider. Der Hauptunterschied zwischen beiden Implementierungen ist die unterschiedliche Anzahl von Synchronisationspunkten, die eine Kommunikation zwischen allen Prozessen erforderlich machen. Die Implementierung `QMR` ergab die bessere Ausführungszeit. Dieses Verhalten wird auch in der Simulation vorhergesagt und der Vorschlag des Grid Configuration Manager ist korrekt.

Die gemessenen Ausführungszeiten für `TFQMR` und $\texttt{TFQMR}_1$ liegen sehr nah beieinander.

Dieses Verhalten lässt sich schon anhand der in Tab.4.11 aufgeführten Merkmale vermuten, da keine sehr großen Unterschiede in den Implementierungen vorhanden sind. In der Tendenz ist festzustellen, dass die gemessenen Ausführungszeiten für die Implementierung $\texttt{TFQMR}_1$ etwas schneller sind als die für TFQMR. Die Vorhersage, die der Grid Configuration Manager auf Grund der in der Simulation ermittelten Ausführungszeit, macht ist auch in diesem Fall korrekt.

verwendete Trace-Datei	Anzahl der Prozessoren auf Rechner 1	Anzahl der Prozessoren auf Rechner 2	gemessene Ausführungszeiten (s)	simulierte Ausführungszeit (s)
Test 1: qmr, 16 Prozesse				
qmr.trf	8	8	1.802327	1.851796 *
	8	8	1.876425	
	8	8	1.757125	
	8	8	1.655712	
	8	8	1.823848	
qmr1.trf	8	8	2.858442	5.622984
	8	8	2.791418	
	8	8	2.430137	
	8	8	2.918516	
	8	8	2.949373	
Test 2: tfqmr, 16 Prozesse				
tfqmr.trf	8	8	3.766639	4.259648
	8	8	3.711796	
	8	8	3.137587	
	8	8	4.048421	
	8	8	3.490557	
tfqmr1.trf	8	8	3.481374	3.500381 *
	8	8	3.639168	
	8	8	3.493191	
	8	8	3.415210	
	8	8	3.481454	
* kennzeichnet die vom GCM zur Verwendung vorgeschlagene Implementierung				

Tabelle 4.13.: Ergebnisse Szenario 3 - Untersuchung verschiedener Verfahren, 16 Prozesse

VERWENDETE TRACE-DATEI	ANZAHL DER PROZESSOREN AUF RECHNER 1	RECHNER 2	GEMESSENE AUSFÜHRUNGSZEITEN (S)	SIMULIERTE AUSFÜHRUNGSZEIT (S)
TEST 1: QMR, 32 PROZESSE				
qmr32.trf	16	16	1.524234	5.666172 *
	16	16	1.343385	
	16	16	1.427566	
	16	16	1.336827	
	16	16	1.474707	
qmr132.trf	16	16	2.444896	7.90087
	16	16	2.386638	
	16	16	2.282217	
	16	16	2.285177	
	16	16	2.353846	
TEST 2: TFQMR, 32 PROZESSE				
tfqmr32.trf	16	16	2.699631	8.67678
	16	16	2.824925	
	16	16	2.831748	
	16	16	2.826975	
	16	16	2.809060	
tfqmr132.trf	16	16	2.673511	5.0411 *
	16	16	2.609544	
	16	16	2.644937	
	16	16	2.621897	
	16	16	2.788405	
* KENNZEICHNET DIE VOM GCM ZUR VERWENDUNG VORGESCHLAGENE IMPLEMENTIERUNG				

Tabelle 4.14.: Ergebnisse Szenario 3 - Untersuchung verschiedener Verfahren, 32 Prozesse

4.2.4. Fazit

Die Ergebnisse der in diesem Kapitel durchgeführten Untersuchungen zeigen, dass das für den Grid Configuration Manager entwickelte Verfahren zur Vorhersage einer effizienten Grid Konfiguration zu korrekten Ergebnissen führt. In allen Tests hat der GCM jeweils die richtige Konfiguration bzw. die beste Implementierung als Vorschlag für die effizienteste Umgebung ausgewählt.

Es soll hier aber noch einmal darauf hingewiesen werden, dass die Entscheidungen

des Grid Configuration Managers in hohem Maße vom verwendeten Programm für die Simulation der Ausführungszeiten abhängen. Auch die Wahl der in die Simulationen einfließenden Parameter spielt eine wichtige Rolle, die in der Regel einen erfahrenen Benutzer voraussetzt. Es wäre sicherlich sinnvoll, für die Vorhersage der Ausführungszeiten auch andere Simulationsprogramme einzusetzen, damit sich z. B. Beschränkungen, wie in Szenario 2, vermeiden lassen.

5. Zusammenfassung und Ausblick

Grids sind Hochleistungsnetze, die die gesamte Kapazität aller angeschlossenen Rechner nutzen und das Austauschen von Daten und Rechenkapazitäten zwischen einzelnen Computern ermöglichen. Zurzeit werden Grid Technologien im Bereich des Hochleistungsrechnens bei der Simulation komplexer ingenieurwissenschaftlicher Vorgänge wie z. B. Strömung in einer Wasserturbine eingesetzt. Die Verteilung der dafür verwendeten parallelen Anwendungen auf verschiedene Rechner kann viele Schwierigkeiten für den Benutzer und die Anwendung verursachen. Die meisten dieser Probleme rühren von der Tatsache her, dass „das Grid“ nicht existiert, sondern aus einer Vielzahl heterogener „Grid-Inseln“ besteht. Dabei ist die Heterogenität der vorhandenen Grid Software Umgebungen das größte Problem mit dem Benutzer umgehen müssen, wenn sie parallele, verteilte Anwendungen effizient im Grid ausführen wollen.

Im Rahmen dieser Arbeit wurde das Konzept und die Implementierung eines Grid Configuration Managers (GCM) vorgestellt, der die Komplexität der Grid Umgebungen und die damit verbundenen Probleme vor dem Benutzer verbergen soll. Das wichtigste Ziel des GCM ist die Vereinfachung des Managements von Grid Umgebungen für Endanwender und Entwickler. Deshalb wurde eine Abstraktion der Schritte, die für die Ausführung von verteilten, parallelen Anwendungen benötigt werden, entwickelt und in die Implementierung als grafische Oberfläche eingearbeitet. Damit ist der GCM in der Lage, Benutzer bei der Erstellung von Grid Konfiguration als Ausführungsumgebung für die verteilten Jobs zu unterstützen. Des Weiteren kann der GCM eine Vorhersage von Leistungsdaten (Laufzeitverhalten) einer Anwendung auf Basis von Tracedaten mit Hilfe von Dimemas machen. Unter Beachtung der vom Benutzer vorgeschlagenen Rechnerressourcen, den auf jedem Rechner verfügbaren Knoten und der Kommunikationscharakteristik des Rechners bzw. zwischen den einzelnen Rechnern versucht der GCM herauszufinden, wie die beste Konfiguration für die Ausführung sein sollte.

Das Starten der Jobs erfolgt zentral innerhalb eines speziell entworfenen Grid Terminals. Das Grid Terminal gibt dem Benutzer auch die Möglichkeit, Befehle für Pre- und Post-Processing Aufgaben auszuführen. Für den Dateitransfer von Ein- und Ausgabedaten wurde eine eigene Komponente entwickelt, die auch in andere Software Projekte integriert werden kann. Die Überwachung der laufenden Jobs erfolgt ebenfalls im Grid Terminal. Die Ausgabe-Meldungen der Jobs sind dort zusammengefasst dargestellt, um die Handhabung zu erleichtern. Der GCM bietet eine spezielle Unterstützung für verteilte Anwendungen, die auf Basis der Kommunikationsbibliothek PACX-MPI entwickelt

wurden. Dafür werden die benötigten Konfigurationsdateien automatisch erstellt und auf den beteiligten Rechnern konsistent gehalten. Außerdem wird die Start-Prozedur von PACX-MPI mit einem eigenen Startup-Server unterstützt.

Der Grid Configuration Manager funktioniert innerhalb einer heterogenen Grid Umgebung. Zurzeit werden auf Globus, UNICORE und ssh basierende Umgebungen unterstützt. Es wurde eine Abstraktionsebene entwickelt, damit dem Benutzer diese Umgebungen mit der gleichen Ansicht präsentiert werden können, so dass die Unterschiede in den einzelnen Umgebungen vor ihm verborgen bleiben.

Detaillierte Untersuchungen zur Bewertung der Vorhersagen des Grid Configuration Managers ergaben, dass die im Rahmen dieser Arbeit entwickelte Vorgehensweise für die Ermittlung der effizientesten Grid Konfiguration verwertbare Ergebnisse erzeugt und damit die Festlegung der Ausführungsumgebung für den Benutzer erleichtert. Der Grid Configuration Manager wurde im Rahmen des DAMIEN Projektes in verschiedenen Szenarien mit ingenieurwissenschaftlichen Anwendungen eingesetzt. Dabei hat sich gezeigt, dass die Handhabung von verteilten Anwendungen durch die Verwendung des GCM signifikant vereinfacht wird.

Eine Fortführung dieser Arbeit kann mehrere Richtungen umfassen: Zum einen sollte man die **Integration weiterer Grid Umgebungen** fortsetzen. Wichtigstes Ziel sollte hierbei die Anpassung an serviceorientierte Grid Lösungen sein, wie sie z. B. mit dem Globus Toolkit Version 4 verwirklicht wurden. Das ermöglicht die Integration neuer Dienste zum Auffinden von Ressourcen, was eine sinnvolle Erweiterung des Grid Configuration Managers darstellt.

Eines der Hauptprobleme für die Ausführung von verteilten, parallelen MPI-Anwendungen ist die Abstimmung beim Start der Anwendung. Die zu verwendenden Ressourcen müssen zu einem bestimmten Zeitpunkt für den Benutzer zur Verfügung stehen. Das ist nur mit der **Entwicklung von ausgeklügelten Scheduling Mechanismen** zu erreichen, die zum jetzigen Zeitpunkt für Grid Umgebungen noch nicht existieren.

Die **Heuristiken für die Leistungsvorhersage sollten erweitert werden**, um eine Verbesserung der Vorhersage zu erreichen. Es könnten z. B. zusätzliche Angaben über die Kommunikationsmuster einer Anwendung einfließen. Eine sinnvolle Erweiterung für den Einsatz des GCM in der Softwareentwicklung ist der Einbau von Parametrisierungsmöglichkeiten für die Simulationsparameter. Des Weiteren ist vorstellbar, die Ermittlung der Parameter für Dimemas zu verfeinern und z. B. unter Ausnutzung von Grid- und Onlinediensten wie dem NetWork Weather Service [87] zu automatisieren, damit die Genauigkeit der Abschätzung weiter erhöht wird.

Ein **Ausbau des Metadaten Repositorys**, um die in den Metadaten enthaltene Datensätze in die Vorhersagen einzubeziehen, kann zu einer weiteren Verbesserung der Vorhersage führen. Zur Erhöhung der Benutzerfreundlichkeit des Repositorys können Erweiterungen wie z. B. eine Suchfunktion etc. beitragen.

Literaturverzeichnis

[1] ALTMANN, J. ; BÖHM, A. : Algorithm-based fault-tolerant programming in scientific computation on multiprocessors. In: *Proceedings of the 3rd Euromicro Workshop on Parallel and Distributed Processing (PDP '95)*, IEEE Computer Society, 1995, S. 374–382

[2] ARNOLD, K. ; GOSLING, J. : *The Java Programming Language*. Addison-Wesley, 1997. – ISBN 0–32134–980–6

[3] AYDT, R. A.: *The Pablo Self-Defining Data Format*. University of Illinois, USA: Pablo Research Group, 1992

[4] BADIA, R. M. ; ESCALÉ, F. ; GABRIEL, E. ; GIMENÉZ, J. ; KELLER, R. ; LABARTA, J. ; MÜLLER, M. S.: Performance Prediction in a Grid environment. In: *European Across Grids Conference*, 2003, S. 257–264

[5] BADIA, R. M. ; ESCALÉ, F. ; GIMÉNEZ, J. ; LABARTA, J. : Final implementation of DAMIEN performance prediction tool / European Center for Parallelism of Barcelona (CEPBA). 2003 (D5.3). – Forschungsbericht. – DAMIEN Projekt, IST-2000-25406

[6] BADIA, R. M. ; GIMENÉZ, J. ; LABARTA, J. : Detail Specification of DAMIEN Performance Prediction Tool / European Center for Parallelism of Barcelona (CEPBA). 2001 (D5.1). – Forschungsbericht. – DAMIEN Projekt, IST-2000-25406

[7] BADIA, R. M. ; GIRONA, S. ; LABARTA, J. : Performance Prediction Functional Specifications / European Center for Parallelism of Barcelona (CEPBA). 2001 (D2.3). – Forschungsbericht. – DAMIEN Projekt, IST-2000-25406

[8] BADIA, R. M. ; LABARTA, J. ; GIMÉNEZ, J. ; ESCALÉ, F. : DIMEMAS: Predicting MPI applications behavior in Grid environments. In: *Workshop on Grid Applications and Programming Tools Held in conjunction with GGF8, June 25 2003, Seattle, USA Proceedings*, 2003, S. 52–62

[9] BARRETT, D. J. ; SILVERMAN, R. ; BYRNES, R. G.: *SSH, The Secure Shell (The Definitive Guide)*. 2nd edition. O'Reilly, 2005. – ISBN 0–596–00895–3

[10] BERMAN, F. ; FOX, G. ; HEY, T. : The Grid: Past, Present and Future. In: *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003, S. 9–51

[11] BÖNISCH, T. B. ; RÜHLE, R. : Efficient Flow Simulation with Structured Multiblock Meshes on Current Supercomputers. In: *ERCOFTAC Bulletin No. 50: Parallel Computing in CFD*, 2001

[12] BOUTEILLER, A. ; HERAULT, T. ; KRAWEZIK, G. ; LEMARINIER, P. ; CAPPELLO, F. : *MPICH-V: a Multiprotocol Fault Tolerant MPI*. 2005. – To appear in International Journal of High Performance Computing and Applications

[13] BRUNST, H. ; WINKLER, M. ; NAGEL, W. E. ; HOPPE, H.-C. : Performance Optimization for Large Scale Computing: The scalable VAMPIR approach. In: *Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, René S. Renner and C. J. Kenneth Tan (Eds.): International Conference on Computational Science – ICCS 2001* Bd. 2074:2. Springer, May 2001, S. 751–760

[14] BÜCKER, H. M.: Entwurf paralleler iterativer Verfahren mit kurzen Rekursionen für große dünnbesetzte lineare Gleichungssysteme / Forschungszentrum Jülich. 1997 (JÜL-3478). – Forschungsbericht

[15] BÜCKER, H. M. ; SAUREN, M. : A parallel version of the unsymmetric Lanczos algorithm and its application to QMR. 1996 (KFA-ZAM-IB-9605). – Forschungsbericht

[16] BURNS, G. ; DAOUD, R. ; VAIGL, J. : LAM: An Open Cluster Environment for MPI. In: *Proceedings of Supercomputing Symposium* (1994), S. 379–386

[17] BUYYA, R. ; ABRAMSON, D. ; GIDDY, J. : Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid. In: *The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, IEEE Computer Society Press, 2000, S. 283–289

[18] BUYYA, R. ; ABRAMSON, D. ; GIDDY, J. ; STOCKINGER, H. : A Case for Economy Grid Architecture for Service Oriented Grid Computing. In: *Special Issue on Grid Computing Environments, Concurrency and Computation: Practice and Experience (CCPE) Journal* 14 (2002), November - December, S. 1507–1542

[19] CERN: *Gridcafe*. `http://gridcafe.web.cern.ch/gridcafe/`, Abruf: 17.05.2006

[20] CHETTY, M. ; BUYYA, R. : Weaving Computational Grids: How analogous are they with electrical grids? In: *IEEE Computing in Science & Engineering* 4 (2002), July-August, Nr. 4, S. 61–71

[21] ESSER, F. : *Java2 Designmuster und Zertifizierungswissen*. Galileo Press, 2001. – ISBN 3–934358–66–7

[22] EUROPEAN CENTER FOR PARALLELISM OF BARCELONA (CEPBA) (Hrsg.): *MPItrace tool, version 1.1, Instrumentation of generic MPI applications, User's Guide*. Universitat Politècnica de Catalunya, Spanien: European Center for Parallelism of Barcelona (CEPBA), November 2000

[23] FOSTER, I. ; KISHIMOTO, H. ; SAVVA, A. ; BERRY, D. ; DJAOUI, A. ; GRIMSHAW, A. ; HORN, B. ; MACIEL, F. ; SIEBENLIST, F. ; SUBRAMANIAM, R. ; TREADWELL, J. ; REICH, J. V.: The Open Grid Services Architecture, Version 1.0 / Global Grid Forum. 2005 (GFD-I.030). – Forschungsbericht

[24] FOSTER, I. : What is the grid? A three point checklist. In: *GRIDtoday* 1 (2002), July 22, Nr. 6. `http://www.gridtoday.com/02/0722/100136.html`. – 28.03.2006

[25] FOSTER, I. ; GEISLER, J. ; GROPP, W. ; KARONIS, N. ; LUSK, E. ; THIRUVATHUKAL, G. ; TÜCKE, S. : Wide-Area Implementation of the Message Passing Standard. In: *Parallel Computing* 24 (1998)

[26] FOSTER, I. ; IAMNITCHI, A. : On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In: *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003, S. 118–128

[27] FOSTER, I. ; KESSELMAN, C. : The Globus Project: A Status Report. In: *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, 1998, S. 4–18

[28] FOSTER, I. ; KESSELMAN, C. ; TSUDIK, G. ; TÜCKE, S. : A security architecture for computational grids. In: *ACM Conference on Computers and Communications Security*, ACM Press, 1998, S. 83–91

[29] FOSTER, I. ; KESSELMAN, C. ; TÜCKE, S. : The Nexus approach to integrating multithreading and communication. In: *Journal of Parallel and Distributed Computing* 37 (1996), S. 70–82

[30] FOSTER, I. ; KESSELMAN, C. ; TÜCKE, S. : The Anatomy of the Grid: Enabling Scalable Virtual Organizations. In: *International Journal of Supercomputer Applications* 15 (2001)

[31] FOSTER, I. ; KESSELMANN, C. : *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann San Francisco/California, 1998. – ISBN 1–55860–475–8

[32] FOX, G. ; PIERCE, M. ; GANNON, D. ; THOMAS, M. : Overview on Grid Computing Environments / Global Grid Forum. 2002 (GFD-I.9). – Forschungsbericht

[33] FREUND, R. W.: A Transpose Free Quasi-Minimal Residual algorithm for non-hermitian linear Systems. In: *SIAM Journal of Scientific Computing* 14 (1993), Nr. 2, S. 470–482

[34] FREUND, R. W. ; NACHTIGAL, N. M.: An implementation of the QMR method based on coupled two-term recurrences. In: *SIAM Journal on Scientific Computing* 15 (1994), March, Nr. 2, S. 313–337

[35] FREY, J. ; TANNENBAUM, T. ; FOSTER, I. ; LIVNY, M. ; TÜCKE, S. : Condor-G: A Computation Management Agent for Multi-Institutional Grids. In: *Cluster Computing* 5 (2002), S. 237–246

[36] FUJITSU LTD: *Arcon – Client interface for UNICORE*. `http://unicore.sourceforge.net/`, Abruf: 03.02.2006

[37] GABRIEL, E. : *Untersuchung effizienter paralleler Gleichungslöser auf die Verwendbarkeit in einem parallelen Strömungsprogramm*. Stuttgart, Rechenzentrum der Universität Stuttgart, Anwendungen der Informatik im Maschinenwesen, Diplomarbeit, April 1998. – 65 S.

[38] GABRIEL, E. ; KELLER, R. ; LINDNER, P. ; MÜLLER, M. S. ; RESCH, M. M.: Software Development in the Grid: The DAMIEN tool-set. In: *Lecture Notes in Computer Science, in P. M. A. Sloot, D. Abramson, A. V. Bogdano, J. J. Dongarra, A. Y. Zomaya and Y. E. Gorbachev (Eds.): Computational Science - ICCS 2003* 2659 (2003), S. 235–244

[39] GABRIEL, E. ; RESCH, M. ; BEISEL, T. ; KELLER, R. : Distributed Computing in a heterogeneous computing environment. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 1998 (Lecture Notes in Computer Science), S. 180 – 187

[40] GABRIEL, E. ; RESCH, M. ; RÜHLE, R. : Implementing MPI with Optimized Algorithms for Metacomputing. In: *Anthony Skjellum, Purushotham V. Bangalore and Yoginder S. Dandass (Eds.): Proceedings of the Third MPI Developer's and User's Conference*. Starkville, Mississippi : MPI Software Technology Press, 1999, S. 31–42

[41] GIRONA, S. ; LABARTA, J. : Sensitivity of Performance Prediction of Message Passing Programs. In: *The Journal of Supercomputing* 17 (2000), November, Nr. 3, S. 291–298

[42] GIRONA, S. ; LABARTA, J. ; BADIA, R. M.: Validation of Dimemas communication model for MPI collective communications. In: *Jack Dongarra, Peter Kacsuk and N. Podhorszki (Eds.): Recent Advances in Parallel Virtual Machine and Message Passing Interface* Bd. 1908, Springer, 2000 (Lecture Notes In Computer Science), S. 39–46. – Proceedings of the 7th European PVM/MPI Users' Group Meeting

[43] GLOBAL GRID FORUM: *GGF*. `http://www.gridforum.org/`, Abruf: 17.05.2006

[44] GLOBUS: *Resource Specification Language*. Version: 2002. `http://www.globus.org/toolkit/docs/3.0/gram/rsl-schema.html`, Abruf: 17.05.2006

[45] GRID TECHNOLOGY RESEARCH CENTER, AIST: *Grid MPI Homepage*. `http://www.gridmpi.org/`, Abruf: 17.05.2006

[46] GRIMSHAW, A. S. ; WULF, W. A. ; LEGION TEAM the: The Legion vision of a worldwide virtual computer. In: *Communications of the ACM* 40 (1997), S. 39–45

[47] HOWES, T. A. ; SMITH, M. C.: A Scalable, Deployable Directory Service Framework for the Internet / Center for Information Technology Integration, University of Michigan. 1995 (CITI Technical Report 95-7). – Forschungsbericht

[48] HUANG, W. ; SANTHANARAMAN, G. ; JIN, H.-W. ; GAO, Q. ; PANDA, D. K.: Design and Implementation of High Performance MVAPICH2: MPI2 over InfiniBand. In:

Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), IEEE Computer Society Press, 2006, S. 43–48

[49] IBM: *Using and Administering IBM LoadLeveler, Release 3.0, SC23-3989*. USA: IBM Corporation, 1996

[50] IMAMURA, T. ; HASEGAWA, Y. ; YAMAGISHI, H. ; TAKEMIYA, H. : TME - a Distributed Resource Handling Tool. In: *Proceedings of the International Conference on Scientific and Engineering Computation (IC-SEC) 2002*, Imperial College Press, 2002

[51] IMAMURA, T. ; TSUJITA, Y. ; KOIDE, H. ; TAKEMIYA, H. : An Architecture of Stampi: MPI library on a cluster of parallel computers. In: *J. Dongarra, P. Kacsuk and N. Podhorszki (Eds.): Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science* 1908 (2000), S. 200–207

[52] INTEL GMBH (Hrsg.): *Intel ® MPI Benchmarks Users Guide and Methodolgy Description*. Brühl, Germany: Intel GmbH, 2004

[53] JCRAFT, INC.: *Jsch project Homepage*. `http://www.jcraft.com/jsch/index.html`, Abruf: 30.05.2006

[54] KARONIS, N. T. ; TOONEN, B. ; FOSTER, I. : MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. In: *Journal of Parallel and Distributed Computing* 63 (2003), S. 551–563

[55] KELLER, R. ; GABRIEL, E. ; KRAMMER, B. ; MÜLLER, M. S. ; RESCH, M. M.: Towards efficient execution of MPI applications on the Grid: porting and optimization issues. In: *Journal of Grid Computing* 1 (2003), Nr. 2, S. 133–149

[56] LASZEWSKI, G. von ; FOSTER, I. ; GAWOR, J. ; LANE, P. : A Java Commodity Grid Kit. In: *Concurrency and Computation: Practice and Experience* 13 (2001), Nr. 8–9, S. 643–662

[57] LASZEWSKI, G. von ; WAGSTROM, P. : Gestalt of the Grid. In: *Salim Hariri and Manish Parashar (Eds.): Tools and Environments for Parallel and Distributed Computing*. Wiley, 2004, S. 149–187

[58] LESYNG, B. ; BALA, P. ; ERWIN, D. : EUROGRID - European computational grid testbed. In: *Journal of parallel and distributed computing* 63 (2003), Nr. 5, S. 590 – 596

[59] LINDNER, P. ; ALMOND, J. : An Integrated Global Service for File Transfer and Management in a Network (FTM). In: ARABNIA, H. R. (Hrsg.): *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA '04)*. Las Vegas, USA : CSREA Press, June 2004, S. 1063–1067

[60] LINDNER, P. ; KELLER, R. : Final Implementation of the Communication Library / High Performance Computing Center Stuttgart (HLRS). 2003 (D7.3). – Forschungsbericht. – DAMIEN Projekt, IST-2000-25406

[61] LINDNER, P. ; RESCH, M. M.: GRIDWELTEN - User requirements and environments for Grid Computing. In: *DFN Mitteilungen* 26 (2003), S. 13–15

[62] LORCH, M. ; KAFURA, D. : Symphony - A Java based Composition and Manipulation Framework for Computational Grids. In: *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, 2002, S. 136

[63] LOWEKAMP, B. ; TIERNEY, B. ; COTTRELL, L. ; HUGHES-JONES, R. ; KIELMANN, T. ; SWANY, M. : A Hierarchy of Network Performance Characteristics for Grid Applications and Services / GGF. 2004 (GFD-R-P.023). – Forschungsbericht. – GGF Network Measurements Working Group

[64] LOY, M. ; ECKSTEIN, R. ; WOOD, D. : *Java Swing*. 2. Auflage. O'Reilly, 2002. – ISBN 0–59600–408–7

[65] MCCONNELL, B. : *Beyond Contact: A Guide to SETI and Communicating with Alien Civilizations*. O'Reilly, 2001

[66] MENDAY, R. ; WIEDER, P. : GRIP: The Evolution of UNICORE towards a Service-Oriented Grid. In: *Proceedings of the Cracow Grid Workshop*. Cracow, Poland : Academic Computer Centre CYFRONET AGH, 2003

[67] NOVOTNY, J. : The Grid Portal Development Kit. In: *IEEE Concurrency and Computation: Practice and Experience* 14 (2002), S. 1129–1144

[68] OPENBSD PROJECT: *OpenSSH project Homepage*. `http://www.openssh.org/`, Abruf: 17.05.2006

[69] ORGANISATION FOR THE ADVANCEMENT OF STRUCTURED INFORMATION STANDARDS: *OASIS*. `http://www.oasis-open.org/`, Abruf: 17.05.2006

[70] PLATFORM COMPUTING INC.: *Load Sharing Facility*. `http://www.platform.com/Products/Platform.LSF.Family/Platform.LSF/`, Abruf: 03.02.2006

[71] RATERING, R. : *Plugin Programmer's Guide UNICOREpro Client Version 1.0*. Brühl: Pallas GmbH, 2003. `http://unicore.sourceforge.net/docs/plugin_programmers_guide.pdf`, Abruf: 15.03.2006

[72] RESCH, M. ; BEISEL, T. ; BÖNISCH, T. B. ; LOFTIS, B. ; REDDY, R. : Performance Issues of Intercontinental Computing. In: *Proceedings of the Cray User Group Conference*. San José/California, May 5-9 1997

[73] RIEDEL, M. ; SANDER, V. ; WIEDER, P. ; SHAN, J. : Web Services Agreement based Resource Negotiation in UNICORE. In: ARABNIA, H. R. (Hrsg.): *2005 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2005, S. 31–37

[74] RIPS, S. : Adaptive Load Distribution in Strong Dynamic Heterogeneous Systems. In: *Proceedings of the Second International Conference on Massively Parallel Computing Systems (MPCS '96)*. Ischia, Italy : IEEE Computer Society Press, May 6-9 1996

[75] SSH COMMUNICATIONS SECURITY: *SSH Communications Security Homepage*. `http://www.ssh.com/`, Abruf: 17.05.2006

[76] STOCKINGER, H. ; DONNO, F. ; LAURE, E. ; MUZAFFAR, S. ; KUNSZT, P. ; ANDRONICO, G. ; MILLAR, P. : Grid data management in action: Experience in running and supporting data management services in the EU DataGrid project. In: *ECONF* C0303241 (2003)

[77] SURRIDGE, M. : A Rough Guide to Grid Security. In: *e-Science Technical Report Series (UKeS-2002-05)* (2002)

[78] SUZUKI, Y. ; SAIA, K. ; MATSUMOTO, N. ; HAZAMA, O. : Visualization Systems on the Information-Technology-Based Laboratory. In: *IEEE Computer Graphics and Applications* 23 (2003), Mar/Apr, Nr. 2, S. 32–39

[79] TAKEMIYA, H. ; YAMAGISHI, N. : *Starpc: A Library for Communication among Tools on a Parallel Computer Cluster - User's and Developer's Guide to Starpc*. JAERI-Data/Code 2000-005. Japan: JAERI, 2000

[80] TANENBAUM, A. S. ; STEEN, M. van: *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2001. – ISBN 0–1308–8893–1

[81] THAIN, D. ; TANNENBAUM, T. ; LIVNY, M. : Condor and the Grid. In: *Fran Berman, Geoffrey Fox and Tony Hey (Eds.): Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002, S. 299–335

[82] THOMPSON, M. R. ; JACKSON, K. R.: Security issues of Grid resource management. In: *Jarek Nabrzyski, Jennifer M. Schopf and Jan Weglarz (Eds.): Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2004, S. 53–59

[83] TONELLOTTO, N. ; YAHYAPOUR, R. ; WIEDER, P. : A Proposal for a Generic Grid Scheduling Architecture / CoreGRID Technical Report. 2006 (TR-0015). – Forschungsbericht. – Institute on Resource Management and Scheduling

[84] ERWIN, D. (Hrsg.): *Joint Project Report for the BMBF Project UNICORE Plus, Grant Number: 01 IR 001 A-D, Duration: January 2000 to December 2002*. 2003

[85] VERIDIAN SYSTEMS: *Portable Batch System*. `http://www.openpbs.org/`, Abruf: 17.05.2006

[86] WOLF, F. ; MOHR, B. : EPILOG Binary Trace-Data Format / Forschungszentrum Jülich. 2004 (FZJ-ZAM-IB-2004-06). – Forschungsbericht

[87] WOLSKI, R. : Experiences with Predicting Resource Performance On-line in Computational Grid Settings. In: *SIGMETRICS Performance Evaluation Review* 30 (2003), March, Nr. 4, S. 41–49

[88] XSTREAM DEVELOPMENT TEAM: *XStream project Homepage*. `http://xstream.codehaus.org/index.html`, Abruf: 03.02.2006

Anhang A.

Verwendung des Grid Configuration Managers

A.1. Installation und Starten

Um den Grid Configuration Manager verwenden zu können, muss eine Java Runtime Umgebung (JDK oder JRE) vorhanden sein. Die aktuelle Version des GCM benötigt Java Version 1.5 oder höher. Die Klassen des GCM wurden in einer Java Archivdatei zusammengefasst, um die Installation zu erleichtern. Es gibt eine selbst entpackende Installationsdatei mit dem Namen `gcm-versionXX-install.jar`, die den Benutzer automatisch durch die Installation führt. Die Installation wird durch Doppelklick auf die Datei oder mit dem Befehl `java -jar gcm-versionXX-install.jar` gestartet. Während der Installation kann der Benutzer das Installationsverzeichnis festlegen. Je nach Betriebssystem stehen danach unterschiedliche Möglichkeiten zum Starten des Grid Configuration Managers zur Verfügung:

- Für Windows-Systeme wird ein Eintrag im Startmenü erzeugt bzw. der Benutzer muss das im Installationsverzeichnis befindliche Skript `bin\config_manager.bat` starten
- Für Unix-Systeme muss der Benutzer das im Installationsverzeichnis befindliche Skript `bin\config_manager` starten

Beim ersten Starten erzeugt der GCM ein Unterverzeichnis `.gcm` im Home-Verzeichnis des Benutzers. In diesem Verzeichnis werden standardmäßig die Logdateien und die Konfigurationsdatei (`User.props`) für den GCM abgelegt.

Nach dem Starten steht dem Benutzer die in Abb. A.1 dargestellte Oberfläche zur Verfügung. Zunächst muss er eine neue Konfiguration in den dafür vorgesehenen Oberflächen („PACX Configuration/Dimemas Configuration“) erstellen und Angaben zur Anwendung machen („Applikation Specification“). Danach kann er (automatisierte) Leistungsvorhersagen mit Dimemas vornehmen und die Anwendung starten.

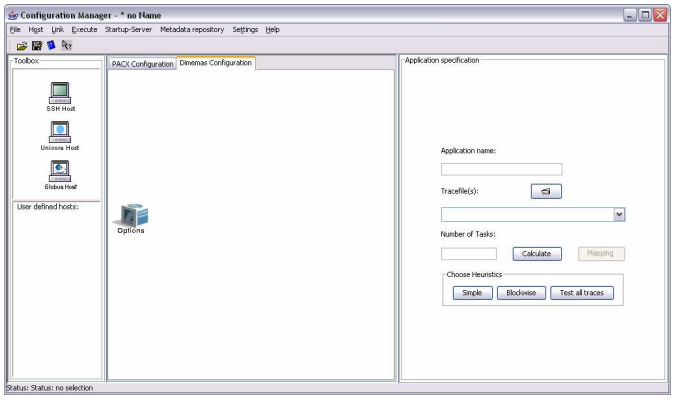

Abbildung A.1.: Der Grid Configuration Manager nach dem Starten

A.2. Erstellen und Speichern einer Grid Konfiguration

Dieses Kapitel bezieht sich sowohl auf die Erstellung von Grid Konfigurationen für die Ausführung von Anwendungen („PACX Configuration“) als auch für die „Dimemas Configuration“ zur Leistungsvorhersage. Zum Erstellen einer neuen Konfiguration muss der Benutzer zunächst die so genannten *Hosts* erstellen, die die einzelnen Rechner repräsentieren. Dazu kann er entweder im Menü „Host“ den entsprechenden Eintrag „New ... Host“ benutzen oder aus der „Toolbox“ durch Doppelklick auf den entsprechenden Host ein neues Hostobjekt erzeugen. Es gibt Hostobjekte für ssh, UNICORE oder Globus.

Nach Auswahl des Hosts mit der rechten Maustaste kann man Menüpunkte für die Bearbeitung der Hostobjekte erzeugen. „Configure Host“ startet eine Eingabedialog für Daten, die den ausgewählten Host betreffen (siehe Abb. A.2).

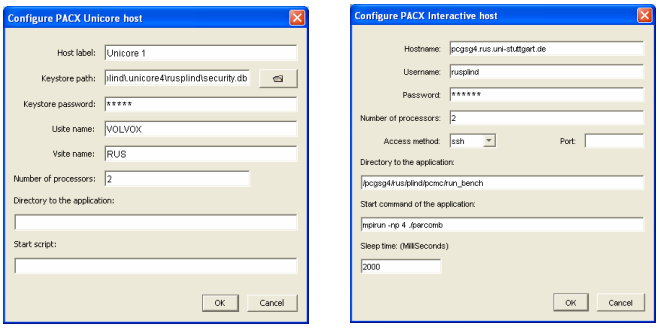

Abbildung A.2.: Dialog für die Konfiguration eines Hostobjektes

Für die Erstellung von Linkobjekten, die die Verbindung zwischen den Rechnern repräsentieren, wird der Benutzer zuerst gefragt, welche Hosts (Rechner) durch diesen Link verbunden sind. Diese muss er nacheinander anklicken, um den Link zu erzeugen.

Nach Auswahl des Links mit der rechten Maustaste kann man Menüpunkte für die Bearbeitung der Linkobjekte erzeugen. Im Konfigurationsdialog für Links (Abb. A.3) kann der Benutzer angeben, wie viele Verbindungen zwischen den Hosts bestehen. Jede dieser Verbindungen kann dann individuell konfiguriert werden.

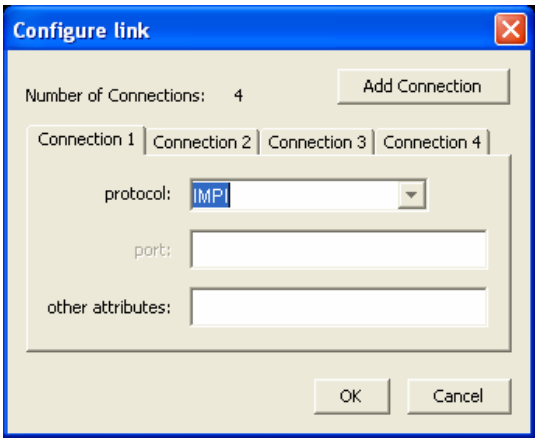

Abbildung A.3.: Dialog für die Konfiguration von Linkobjekten

Im Menü „File“ kann der Benutzer das Speichern seiner Konfiguration starten. Dafür wird ein Dialog angezeigt, mit dem der Benutzer den Speicherort für die Konfiguration festlegen und einen Dateinamen angeben kann. Er kann die aktuelle Konfiguration auch exportieren. Im Unterschied zum normalen Speichern werden damit keine Passwörter gesichert.

Für das Öffnen und Importieren stehen ebenfalls entsprechende Funktionen bereit. Zusätzlich existiert die Möglichkeit, die aktuelle Konfiguration als Bilddatei im jpg-Format zu sichern, um sie z. B. für Präsentationen zu verwenden. „New Configuration“ löscht die aktuellen Konfiguration ohne vorherige Nachfrage.

In Tabelle A.1 sind die Befehle der Menüs des GCM für die Erstellung von Konfigurationen noch einmal zusammengefasst.

MENÜPUNKT	TASTENKÜRZEL	VERWENDUNG
New configuration	(Ctrl+N)	öffnet ein leeres Konfigurationsfenster
Open configuration	(Ctrl+O)	öffnet eine gespeicherte Konfiguration
Save configuration		Speichert die aktuelle Konfiguration im XML-Format
Save as ...		Speichert die aktuelle Konfiguration unter einem neuen Dateinamen
Export configuration		Exportiert die neue Konfiguration (keine Speicherung der Passwörter)
Import configuration		Importiert eine Konfiguration
Print to jpg-image		Druckt die aktuelle Konfiguration in eine jpg-Datei
Exit		Beendet GCM. Falls der Benutzer Änderungen an der aktuellen Konfiguration vorgenommen hat, wird gefragt, ob diese gespeichert werden sollen.
New <Typ> host		Erstellt ein neues Host Objekt des ausgewählten Typs
Delete host		Löscht ein ausgewähltes Host Objekt
Configure host		Öffnet den Dialog für die Konfiguration des ausgewählten Host Objektes
New link		Erstellt ein neues *Link* Objekt
Delete host		Löscht ein ausgewähltes *Link* Objekt
Configure host		Öffnet den Dialog für die Konfiguration des ausgewählten *Link* Objektes

Tabelle A.1.: Menüpunkte des Grid Configuration Managers für Grid Konfigurationen

A.3. Leistungsvorhersage mit Dimemas

Zunächst muss vom Benutzer sichergestellt werden, dass die „Dimemas Configuration“ als aktuelle Konfiguration ausgewählt wurde. Nach der Erstellung einer Dimemas Grid Konfiguration, muss der Benutzer in der „Application Specification“ angeben, wo die für die Simulation benötigten Tracedateien zu finden sind. Außerdem muss angegeben werden, wie viele Prozesse von der Anwendung benötigt werden. Dazu kann der Benutzer den „Calculate“-Knopf drücken, der eine automatische Berechnung der Prozesse aus der aktuellen Tracedatei vornimmt. Danach muss der Benutzer im „Mapping“-Dialog angeben, wie die benötigten Prozesse auf die Hosts in der aktuellen Dimemas Konfiguration verteilt sind.

Nach dem Betätigen der Schaltfläche „Start Dimemas“ im Menü „Execute“ werden automatisch die Konfigurationsdateien für den Dimemas Simulator erstellt und das Programm gestartet. Die Ergebnisse oder Fehlermeldungen der Simulation erscheinen in gesonderten Ausgabefenstern.

A.4. Automatische Leistungsvorhersage

Für die automatische Leistungsvorhersage muss der Benutzer eine Dimemas Grid Konfiguration erstellen, ohne Angaben für die Verteilung der Prozesse im „Mapping"-Dialog zu machen, da diese Verteilung automatisch vorgenommen wird. Der Benutzer muss auswählen, auf welcher Methode die Vorhersage basieren soll:

- *Simple* erstellt Testkonfigurationen basierend auf einfacher Verteilung.
- *Blockwise* erstellt Testkonfigurationen basierend auf blockweiser Verteilung. Der Benutzer muss eine Blockgröße angeben.
- *Test all traces* verwendet für die Auswertung alle vom Benutzer angegebenen Tracedateien.

Die Methode kann vom Benutzer durch Drücken der entsprechenden Schaltfläche ausgewählt werden. Der Grid Configuration Manager testet dann verschiedene Konfigurationen und erstellt basierend auf der Auswertung einen Vorschlag für die effizienteste Ausführungsumgebung im „PACX Configuration" Fenster.

A.5. Starten und Überwachen der Anwendung, Dateitransfer

Nach dem Betätigen der Schaltfläche „Grid Terminal" im Menü „Execute" öffnet sich das Grid Terminal, mit dem der Benutzer Befehle auf entfernten Rechnern ausführen und die Rückmeldungen anschauen kann. Auf der linken Seite des Grid Terminals kann der Benutzer aus einer Liste der sich in der aktuellen Konfiguration befindlichen Hosts, den- oder diejenigen auswählen an den er Befehle verschicken soll. Falls keine Hosts direkt auswählt werden, versendet das Grid Terminal das Kommando an alle Hosts, die sich in der aktuellen Konfiguration befinden. Bei Auswahl der Schaltfläche „Execute Pretty Command" werden die in der PACX Konfiguration bei jedem Host angegebenen Startkommandos verwendet. Wenn Kommandos in einem extra Feld eingegeben werden, können diese mit Auswahl der „Execute" Schaltfläche verschickt werden.

Nach dem Betätigen der Schaltfläche „File Transfer" im Menü „Execute" öffnet sich ein Dialog mit einer Liste aller aktuell konfigurierten Maschinen, aus der der Benutzer die Maschine auswählen kann, zu der ein Dateitransfer stattfinden soll. Falls notwendig, kann der Benutzer auch zusätzliche Maschinen definieren. Nachdem die Verbindung zur entfernten Maschine hergestellt wurde, öffnet sich ein Datei-Browser, der auf der einen Seite die lokalen Dateiverzeichnisse und auf der anderen Seite die des entfernten Rechners darstellt. Jetzt kann der Benutzer Dateien transferieren oder Operationen (`mkdir, delete, rename`) auf dem entfernten Rechner ausführen. Die Abb. A.5 zeigt die grafische Oberfläche der Dateitransferkomponente.

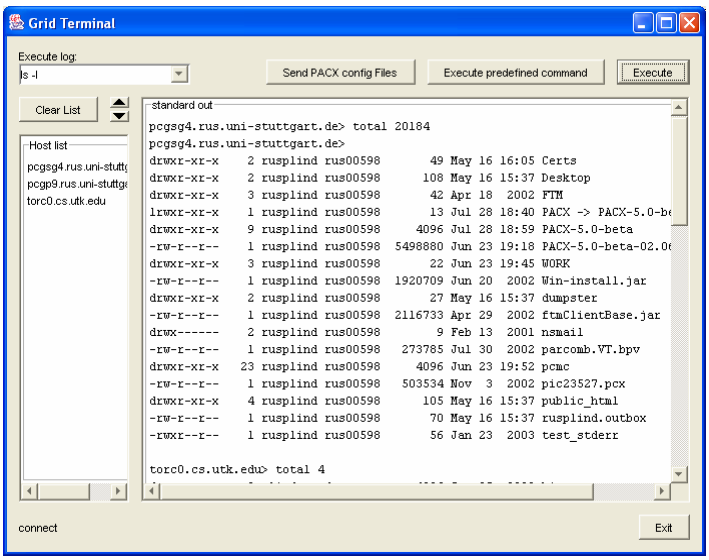

Abbildung A.4.: Das Grid Terminal

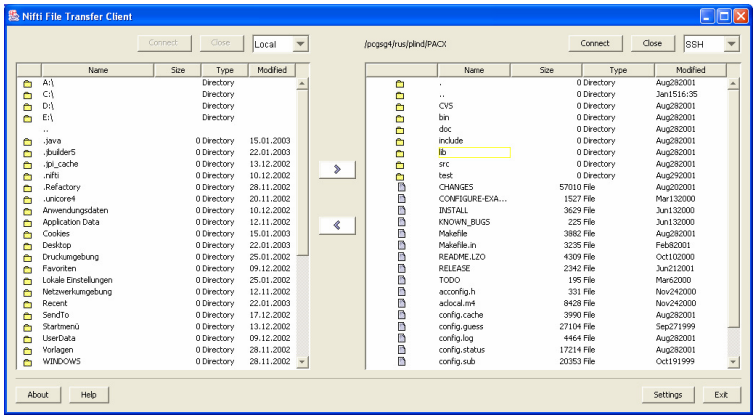

Abbildung A.5.: Dateitransferkomponente von GCM

A.6. Spezifische Unterstützung von PACX-MPI Anwendungen

Zunächst muss vom Benutzer sichergestellt werden, dass die „PACX Configuration“ als aktuelle Konfiguration ausgewählt wurde. Nach dem Betätigen der Schaltfläche „Send configuration (Ctrl+S)“ im Menü „Execute“ werden die `hostfiles` und, falls notwen-

MENÜPUNKT	TASTENKÜRZEL	VERWENDUNG
Add ...		Fügt der aktuellen Konfiguration einen Startup-Server hinzu
Configure ...		Startet einen Eingabedialog für die Konfiguration des Startup-Servers
Start ...		Startet den Startup-Server
Delete ...		Löscht den Startup-Server aus der aktuellen Konfiguration

Tabelle A.2.: Befehle des Startup-Server Menüs

dig, auch die `netfiles` für PACX-MPI generiert. Nach Erstellung werden die Dateien automatisch an alle an der aktuellen PACX Konfiguration beteiligten Maschinen versendet. Deshalb sollte dieser Schritt erst durchgeführt werden, wenn die Konfiguration abgeschlossen ist.

Für das Starten und die Konfiguration des Startup-Servers für PACX-MPI muss man einen Startup-Server Host über das Menü „Startup-Server“ - „Add Startup-Server“ generieren. In einem speziellen Dialog kann dieser dann konfiguriert werden. Der Benutzer muss entscheiden, welchen Startup-Server er benutzen will. Die Vorgabeeinstellung verwendet den integrierten Startup-Server. Der Benutzer kann aber auch selbst einen Startup-Server konfigurieren. Falls er diese Option verwenden will, muss er das Häkchen bei „Use default startup-server“ entfernen und angeben wie und wo er seinen Startup-Server starten will.

Nach der Konfiguration muss der Startup-Server mit einem speziellen Menüpunkt gestartet werden. Wenn der Benutzer dann mit der rechten Maus das Startup-Server-Symbol anklickt und den Menüpunkt „Log window“ auswählt, öffnet sich ein Fenster mit Status-Informationen des Startup-Servers. In Tabelle A.2 sind die Menüpunkte für die Handhabung des Startup-Servers noch einmal zusammengefasst.

A.7. Einstellungen und Hilfe

Der Benutzer kann einen Dialog für Einstellungen im „Settings“ Menü unter „User defaults“ starten. Des Weiteren kann er mit dem Menüpunkt „Save User hosts“ seine im Toolbox-Bereich abgelegten, vorkonfigurierten Hosts in einer eigenen Datei abspeichern.

Um den Benutzer bei der Verwendung des GCM zu unterstützen, wurde ein integriertes Hilfesystem entwickelt. Es enthält eine Volltextsuche, und eine Context-Sensitive Hilfe. Das Hilfesystem enthält zusätzlich auch eine Dokumentation zu PACX-MPI. Es wird über den Menüpunkt „Help“-“Help Topics“ gestartet.

Glossar

Application Programming Interface (API) ist eine Schnittstelle, die von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.

Asynchronous Transfer Mode (ATM) ist eine Technik, bei der der Datenverkehr in kleine Pakete, Zellen genannt, mit fester Länge (53 Byte) codiert und über asynchrones Zeitmultiplexing übertragen wird. Die Zellen-Technik hat im Vergleich zu Übertragungstechniken mit variabler Paketgröße (zum Beispiel Ethernet) den Vorteil, dass die Zellen effizienter weitergeleitet werden können.

Authentifizierung (Beweisen einer Identität) bezeichnet den Vorgang der Identifikation eines Benutzers von Rechnerressourcen.

Autorisierung regelt die Frage, welche Berechtigungen einem konkreten Benutzer zugestanden werden. Das erfolgt gewöhnlich nach erfolgreicher Authentifizierung.

Bandbreite einer Netzverbindung ist die Menge der gesendeten Daten je Zeiteinheit. Für Parallelrechner wird sie in Megabyte pro Sekunde angegeben. Für schnelle Netze erfolgt die Angabe oft in Megabit pro Sekunde.

Batchsystem ist ein Verwaltungssystem für lokale Ressourcen auf einem Rechner. Der Benutzer startet nicht einfach sein Programm, sondern er teilt dem Batch-System mit, was er tun möchte und welche Ressourcen dabei verbraucht werden. Diese Aufträge werden zunächst in einer Warteschlange (engl.: queue) zwischengespeichert. Es werden dann nur so viele Rechnungen gestartet, wie in den Hauptspeicher passen. Wenn ein Job beendet ist, wird automatisch der nächste aus der Warteschlange begonnen.

Certification Authority (CA) (dt.: Zertifizierungsstelle) ist eine Organisation, die Zertifikate heraus gibt, verwaltet und einzieht. Solche Organisationen verstehen sich als vertrauenswürdige Dritte, werden also von beiden Partnern einer Kommunikationskette anerkannt.

Cluster bezeichnet eine Anzahl von vernetzten Rechner, die von außen in vielen Fällen als ein Rechner gesehen werden können. In der Regel sind die einzelnen Elemente eines Clusters untereinander über ein schnelles Netzwerk verbunden. Ziel des

Clustering besteht meistens in der Erhöhung der Rechengeschwindigkeit oder der Verfügbarkeit gegenüber einem einzelnen Computer. Die einzelnen in einem Cluster befindlichen Rechner werden als Knoten bezeichnet.

Design-Pattern (dt.: Entwurfsmuster) bezeichnet im Software-Engineering eine besondere Objektstruktur oder Interaktions-Beziehung zwischen Objekten, die bei ähnlichen Problemen so immer wiederkehrt.

e-Business ist die integrierte Ausführung aller automatisierbaren Geschäftsprozesse eines Unternehmens mit Hilfe von Informations- und Kommunikationstechnologie.

File Transfer Protocol (ftp) ist ein Netzprotokoll für die Dateiübertragung in TCP/IP-Netzwerken.

Firewall (dt.: Feuerwand) ist ein Schutzmechanismus, der verhindern soll, dass unerlaubte Zugriffe auf ein Netz oder einen Rechner erfolgen. Meist erfolgt das durch einen Filter, der zwischen erlaubten und unerlaubten Zugriffen unterscheidet.

HyperText Transfer Protocol (HTTP) ist ein Protokoll zur Übertragung von Daten über ein Netzwerk. Es wird hauptsächlich eingesetzt, um Webseiten und andere Daten aus dem World Wide Web (WWW) in einen Webbrowser zu laden.

HyperText Transfer Protocol Secure (HTTPS) ist als eine zusätzliche Schicht zwischen HTTP und TCP definiert. Es dient zur Verschlüsselung und zur Authentifizierung der Kommunikation zwischen Web-Server und Browser.

Java ist eine objektorientierte, plattformunabhängige Programmiersprache, die von SunMicrosystems entwickelt wurde. Üblicherweise benötigen Java-Programme zur Ausführung eine spezielle Umgebung (Java Virtual Machine).

JavaBeans JavaBeans ist ein von der Firma Sun entwickeltes auf Java basierendes Komponentenmodell zur Softwareentwicklung, d. h. eine Architektur für die Definition und Wiederverwendung von Softwarekomponenten. Einzelne Beans dienen als Basisbausteine aus denen man durch Zusammensetzen größere Softwareteile erstellen kann.

Klassendiagramm ist in der Informatik eine graphische Darstellung von Klassen sowie der Beziehungen zwischen diesen Klassen.

Latenz ist eine Angabe über die beanspruchte Zeit einer Datenübertragung zwischen mehreren Computern in einem Netzwerk. Die Latenz bzw. Netzwerklatenz wird in der Regel in Millisekunden (ms) angegeben und ist ein direktes Maß für die Leistungsfähigkeit eines Netzwerks. Je größer der Betrag der Latenz desto größer ist die Verzögerung der gesendeten Daten.

Lightweight Directory Access Protocol (LDAP) ist ein Netzprotokoll, das die Abfrage und die Modifikation von Informationen eines Verzeichnisdienstes (eine im Netzwerk verteilte hierarchische Datenbank) erlaubt.

Message Passing ist ein Programmiermodell für parallele Anwendungen, bei dem die beteiligten Prozesse jeweils ihren eigenen privaten Adressbereich haben und Daten über explizite Messages (Nachrichten) austauschen. Die Synchronisation ist implizit im Akt des Message-Austausches enthalten.

Message-Passing Interface (MPI) ist die standardisierte Definition für eine Programmierschnittstelle des Message-Passing. MPI wurde von einem Forum von Rechnerherstellern, Forschungseinrichtungen und Software-Herstellern definiert und ist kein offizieller Standard. Es wird jedoch von der Industrie als Standard anerkannt.

Metadaten oder Metadateninformationen bezeichnen im Allgemeinen Daten, die Informationen über andere Daten enthalten. Bei den beschriebenen Daten handelt es sich oft um größere Datensammlungen (Dokumente) wie Bücher, Datenbanken oder Dateien. Eine allgemein gültige Unterscheidung zwischen Metadaten und normalen Daten existiert allerdings nicht. So werden auch Angaben zur Beschreibung eines Objektes (z. B. Personennamen) als Metadaten bezeichnet.

Model-View-Controller (MVC) (dt.: Modell-Präsentation-Steuerung) bezeichnet ein Architekturmuster zur Aufteilung von Softwaresystemen in die drei Einheiten Datenmodell (engl. Model), Präsentation (engl. View) und Programmsteuerung (engl. Controller). Ziel des Modells ist ein flexibles Programmdesign, um u. a. eine spätere Änderung oder Erweiterung einfach zu halten und die Wiederverwendbarkeit der einzelnen Komponenten zu ermöglichen. Außerdem sorgt das Modell bei großen Anwendungen für eine gewisse Übersicht und Ordnung durch Reduzierung der Komplexität.

Peer-to-Peer (P2P) computing (dt.: „Seinesgleichen“ zu „Seinesgleichen“) ist die direkte Vernetzung einzelner Rechner verschiedenster Anwender im Internet ohne einen zentralen Rechner.

Performance-Analyse bedeutet, herauszufinden, in welchem Teil des Programms die meiste Rechenzeit verbraucht wird. Performace-Analyse ist sehr wichtig, wenn man ein Programm beschleunigen will. Nach der Performance-Analyse können gezielt die Programmstellen optimiert werden, die die meiste Rechenzeit benötigen.

Public Key Infrastruktur (PKI) (dt.: Öffentlicher-Schlüssel-Infrastruktur) bezeichnet in der Kryptologie ein System, welches es ermöglicht, digitale Zertifikate auszustellen, zu verteilen und zu prüfen. Die innerhalb einer PKI ausgestellten Zertifikate

sind meist auf Personen oder Rechner festgelegt und werden zur Absicherung computergestützter Kommunikation verwendet.

QoS (Quality of Service) bezeichnet allgemein die Dienstgüte von Übertragungskanälen. Die Dienstgüte setzt sich aus einer Reihe von Parametern wie Verlustrate, Verfüg-barkeit, Durchsatz und Latenz (Verzögerung) zusammen.

Remote Method Invocation (RMI) (dt.: entfernter Methodenaufruf) ist der Aufruf einer Methode eines entfernten Objekts. Dabei sieht der Aufruf für das aufrufende Objekt (bzw. dessen Programmierer) genauso aus, wie ein lokaler Aufruf, es müssen jedoch besondere Ausnahmen abgefangen werden, die zum Beispiel einen Verbindungsabbruch signalisieren können.

Remote Procedure Call (RPC) ist ein Mechanismus, der das Aufrufen von auf entfernten Computern befindlichen Prozeduren erlaubt. RPC wurde in den 1980er von SUN Microsystems entwickelt und von dem Internet-Gremium IETF (Internet Engineering Task Force) standardisiert.

Resource Broker ist ein zentraler Dienst innerhalb von Grid Umgebungen, der zu jeder Rechenanforderung einen möglichst günstigen Ort für deren Ausführung bestimmt. Die Kriterien für die Auswahl können beispielsweise möglichst schnelle Ausführung und möglichst enig Transprot von Daten sein.

Round-Robin ist ein in der Informatik verwendete(s) Strategie/Verfahren, um z. B. Ressourcen bestimmten Prozessen zuzuordnen.

Scheduling ist die geplante Verteilung von Ressourcen auf zu bearbeitende Aufgaben.

Secure Shell (ssh) ist eine Anwendung und ein sicheres Kommunikationsprotokoll für den Fernzugriff auf einen Rechner.

Secure Sockets Layer (SSL) ist ein Verschlüsselungsprotokoll für Datenübertragungen im Internet.

Single-Sign-On (SSO) (dt.: Einmalanmeldung) Dabei muss sich der Anwender nur bei einer Instanz authentifizieren; diese vermittelt dann den Zugriff auf die einzelnen benötigten Daten bzw. Anwendungen. Der Anwender muss sich also auch nur ein Passwort merken bzw. ein Authentifizierungsverfahren durchlaufen. Die SSO-Instanz speichert und verwaltet die Passworte der einzelnen Anwendungen kontrolliert.

Swing ist eine API zum Programmieren von grafischen Benutzeroberflächen. Swing wurde von Sun Microsystems für die Programmiersprache Java entwickelt. Seit Java-Version 1.2 (1998) ist es Bestandteil der Java-Runtime. Swing gehört zu den Java Foundation Classes (JFC), die eine Sammlung von Bibliotheken zur Programmierung von grafischen Benutzerschnittstellen bereitstellen.

Symmetric Multi-Processing (SMP) (dt.: Symmetrische Parallelverarbeitung) ist eine Rechnerarchitektur, bei der mehrere Prozessoren auf den gleichen Speicher zugreifen. Der Speicher enthält dabei eine Instanz des Betriebssystems und je eine Instanz der verwendeten Anwendungen und Daten. Da das Betriebssystem die Verarbeitung in Tasks aufteilt und diese den jeweils freien Prozessoren zuteilt, führt SMP zu einer Reduzierung der Verarbeitungszeit.

TCP/IP (Transmission Control Protocol/Internetprotocol) wird als Synonym für die Familie der Internetprotokolle verwendet. Das ist eine Sammlung von rund 500 Netzprotokollen, die die Basis für die Netzkommunikation im Internet bilden.

Tracing ist die detaillierte Aufnahme der einzelnen Schritte, die ein Programm während seiner Laufzeit durchläuft. Die damit erzeugten Tracedaten werden z. B. in der Performance-Analyse verwendet.

UPL (UNICORE Protocol Layer) definiert die Struktur zum Austausch von Daten zwischen UNICORE-Server und -Klient über eine SSL-Verbindung.

Virtual Private Network (VPN) (dt.: Virtuelles Privates Netzwerk) ist ein Computernetz, das zum Transport privater Daten ein öffentliches Netz (zum Beispiel das Internet) nutzt. Teilnehmer eines VPN können Daten wie in einem internen LAN austauschen. Die einzelnen Teilnehmer selbst müssen hierzu nicht direkt verbunden sein. Die Verbindung über das öffentliche Netz wird üblicherweise verschlüsselt. Der Begriff „Private“impliziert jedoch nicht, wie vielfach angenommen, dass es sich um eine verschlüsselte Übertragung handelt. Eine Verbindung der Netze wird über einen Tunnel zwischen VPN-Client und VPN-Server (Concentrator) ermöglicht. Meist wird der Tunnel dabei gesichert, aber auch ein ungesicherter Klartexttunnel ist ein VPN.

Web Services Ein Web Service ist ein Teil einer auf verteilten Systemen arbeitenden Software, welche über Internet Standard Web Protokolle wie z. B. HTTP kommuniziert und XML zur Beschreibung der gesendeten und empfangenen Messages verwendet.

X.509 ist ein Standard für eine PKI und derzeit der wichtigste Standard für digitale Zertifikate. Er setzt ein striktes hierarchisches System von vertrauenswürdigen Zertifizierungsstellen (CA) voraus, die Zertifikate erteilen können.

XML (eXtensible Markup Language) (dt.: erweiterbare Auszeichnungs-Sprache) ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur.

Index

Arcon Bibliothek, 60
Automatische Leistungsvorhersage, 63
Avaki, 31

Computational Grid, 2, 6
Credential delegation, 5

DAMIEN Projekt, 81
Dimemas, 48

e-Business, 1
e-Science, 1
Epilog, 85

FTM, 72

Gleichungslöser, 84
Global Grid Forum, 37
Globus, 19, 61
Grid, 1, 2
Grid Architektur, 3
Grid Configuration Manager, 44, 55
Grid Konfiguration, 55, 57
Grid Middleware, 5
Grid Ressourcen, 2
Grid Terminal, 72

Host, 57

Intel MPI Benchmark, 86
ITBL, 27

Java, 56, 113
JSch, 60

Kosten, 11

Leistungsvorhersage, 61

Metacomputing-Plugin, 76
MPI, 45
MPICH-G2, 21
Mpitrace, 85

NIFTI, 73

OASIS, 40
OGSA, 3

PACX-MPI, 45, 46
Peer-to-Peer Computing, 3

QMR-Verfahren, 96
Quality of Service, 3

Scheduling, 9, 104
Sicherheit, 5
Ssh, 18, 60
Stampi, 30
Starpc, 27

TFQMR-Verfahren, 96
TME, 27
Toolbox, 59

UNICORE, 23, 60

Virtuelle Organisation, 3
VoltaireMPI, 85

Lebenslauf

Persönliche Daten

Name	Peggy Lindner
geboren	am 04. 09. 1974 in Zschopau
Familienstand	verheiratet

Schulbildung

09/1981–07/1991	Polytechnische Oberschule in Waldkirchen/Forchheim
09/1991–06/1993	Gymnasium Brand-Erbisdorf Abschluss: Abitur

Studium

10/1994–04/2000	Technische Universität Bergakademie Freiberg Diplom-Ingenieur: Geotechnik/Bergbau Vertiefungsrichtung: Spezialtiefbau

Beruf

02/2000–01/2001	Matthaei + Schotte Ingenieurgesellschaft mbH in Stuttgart, Entwicklung Software für die Bemessung im Verbundbau
seit 02/2001	wissenschaftliche Mitarbeiterin am Höchstleistungsrechenzentrum Stuttgart (HLRS)

Stuttgart, 20. Juni 2006