Studiengang: Softwaretechnik

Prüfer: Prof. Dr. Rothermel

Betreuer: Dipl. Inf. Christoph Ruffner
Steffen Rost (IBM Böblingen)

begonnen am: 9. März 2003

beendet am: 9. September 2003

CR-Klassifikation: C.2.4, C.4, D.2.8

Diplomarbeit Nr. 2084

Dynamische Einbindung unterschiedlicher Scheduling-Algorithmen in eine Grid Umgebung

Tobias Beichter

Institut für Informatik
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Grid Computing ermöglicht ein flexibles, sicheres und koordiniertes Nutzen von verteilten Ressourcen zwischen einer dynamischen Gruppe von Individuen und Institutionen. Ein Benutzer übergibt ein oder mehrere Anwendungen an den Scheduler, der die Anwendungen auf den potentiell in Frage kommenden und zur Verfügung stehenden Ressourcen ausführt. Das Resultat wird dem Benutzer nach Ende der Berechnung zurückgeliefert.

Das IBM Entwicklungslabor in Böblingen hat ein internes, auf dem Globus Toolkit basierendes Grid entwickelt. Der verwendete Scheduler ist jedoch sehr stark auf eine bestimmte Art von Simulationsjobs zugeschnitten. Somit können keine anderen Anwendungen mit abweichenden Ressourcenanforderungen durch das Grid ausgeführt werden.

Im Rahmen dieser Diplomarbeit soll auf der Basis des existierenden Grids ein Framework entwickelt werden, das eine Matchmaking-Funktionalität zur Verfügung stellt. Damit soll der zu implementierende Scheduler abhängig von den Ressourcenanforderungen der Anwendung und den zur Verfügung stehenden Ressourcen eine Auswahl durchführen. Zusätzlich soll der Scheduler durch unterschiedliche Scheduling-Policies in seinen Entscheidungen beeinflusst werden. Abschließend soll eine Evaluierung durchgeführt werden, die die erzielten Ergebnisse untersucht und bewertet.

Inhaltsverzeichnis

1 Einleitung **6**

1.1 Grid Computing 6

1.2 Themenstellung 6

1.3 Gliederung 7

2 Grundlagen **8**

2.1 Das Grid 8

2.1.1 Einführung 8

2.1.2 Definition 8

2.1.3 Funktionsweise 9

2.1.4 Virtuelle Organisation 9

2.1.5 Grid-Scheduler 10

2.2 Architektur eines Grids 10

2.2.1 Systemkomponenten 10

2.2.2 Beispiel einer Grid-Architektur 12

2.2.3 Anwendungen 14

2.3 Scheduling 15

2.3.1 Erstellung des Ausführungsplans 15

2.3.2 Schedulingarten 16

2.3.3 Lokaler Scheduler am Beispiel LoadLeveler 17

2.3.4 Grid Scheduler vs. lokale Scheduler 19

2.3.5 Phasen des Grid-Schedulingprozesses 19

2.3.6 Zentraler/Dezentraler Scheduler 20

2.3.7 Entscheidungsgrößen 22

2.4 Framework 23

3 Architektur im LabGrid **24**

3.1 Das Globus Toolkit 24

3.1.1 Beispiele für den Einsatz 24

3.1.2 Hauptkomponenten 24

3.2 Das Grid von IBM 27

3.2.1 Motivation 27

3.2.2 Architektur 27

3.2.3 Aktueller Stand 28

3.2.4 Beispiel für die Jobausführung 29

3.2.5 Zukunft 29

3.3 Der Scheduler 29
3.3.1 Komponenten 29
3.3.2 Architektur 31
3.4 Scheduling-Policy 32
3.5 XML-Jobbeschreibungssprache 36
3.6 Nachteile des Schedulers 36

4 Anforderungen an das Framework 38
4.1 Anforderungen der IBM 38
4.2 Nutzen der Lösung 39

5 Verwandte Arbeiten 40
5.1 Matchmaking 40
5.1.1 Condor ClassAds 40
5.1.2 Matchmaking in Globus 40
5.1.3 Matchmaking in Legion 40
5.1.4 Matchmaking in herkömmlichen Schedulern 40
5.2 Untersuchung existierender Scheduler 41
5.2.1 AppLeS 41
5.2.2 Nimrod-G 42
5.2.3 Condor-G 43
5.2.4 Weitere Grid-Scheduler 45
5.3 Scheduling-Policies 45
5.3.1 Interaktiver Modus 45
5.3.2 Batch-Modus 46
5.4 Jobbeschreibungssprachen 46
5.4.1 Resource Specification Language (RSL) 47
5.4.2 Job Submission Description Language (JSDL) 47

6 Architektur des Frameworks 50
6.1 Das Framework 50
6.1.1 Neue Komponenten 51
6.1.2 Funktionsweise 52
6.2 Optimierung des Frameworks 54
6.3 Schnittstellen 56
6.3.1 Anbindung von Informationssystemen 56
6.3.2 Anbindung von Scheduling-Policies 57
6.3.3 Anbindung einer Grid-Middleware 59
6.4 XML-Jobbeschreibungssprache 60

7 Prototyp **63**
7.1 Scheduling-Policy 63
7.1.1 Probleme der ursprünglichen Policy 63
7.1.2 Der neue Algorithmus 64
7.1.3 Vergleich der beiden Policies 68
7.2 GRAM-Adapter 69
7.3 MDS-Adapter 70

8 Evaluierung **71**
8.1 Evaluierung der Policies 71
8.1.1 Auslastung 72
8.1.2 Ausführungsdauer 74
8.1.3 Durchsatz und Wirkungsgrad 75
8.1.4 Durchführung 75
8.1.5 Ergebnisse 76
8.1.6 Diskussion 81
8.2 Integration von heterogenen Ressourcen 82
8.2.1 Durchführung 82
8.2.2 Ergebnisse 82
8.2.3 Diskussion 83

9 Zusammenfassung **84**
9.1 Lösung der Probleme, Bewertung der Ergebnisse 84
9.2 Neue Erkenntnisse 84

10 Ausblick **85**
10.1 Mögliche Erweiterungen des Frameworks 85
10.1.1 Mögliche Erweiterungen der Jobbeschreibungssprache . . . 85
10.1.2 Verwaltung mehrerer Informationssysteme 86
10.1.3 Automatische Auswahl der Policy 86
10.2 Open Grid Service Architecture (OGSA) 86
10.3 Anpassung des Frameworks für Globus Toolkit 3.x 86

11 Begriffslexikon **88**

Abbildungsverzeichnis

1 Architektur eines Grid-Schedulers 11
2 Sanduhren-Prinzip 12
3 Scheduling 17
4 Drei-Phasen Modell des Grid-Schedulings nach Schopf [Schopf, 2002] 20
5 zentraler und dezentraler Scheduler 21
6 hybrid Scheduler 21
7 Verwendung eines Frameworks 23
8 Die vier Globus Toolkit-Komponenten [Globus, 2003a] 25
9 Zusammenspiel der Globus Toolkit-Komponenten 26
10 Architektur des LabGrids 28
11 Der ursprüngliche Scheduler 32
12 AppLeS-Architektur 42
13 Nimrod/G-Architektur 43
14 Condor-G im Zusammenspiel mit Globus [Frey, 2001] 44
15 Prozess der Job-Übergabe 49
16 Das Framework 50
17 Das optimierte Framework 55
18 MatchMaker mit der Schnittstelle zum Informationssystem 58
19 PolicyManager mit der Schnittstelle zu den Policies 60
20 Schnittstelle zur Grid-Middleware mit dem GRAM-Adapter 61
21 Auslastung mit der alten Policy 64
22 Auslastung mit der neuen Policy 69
23 Die Callback-Lösung 72
24 Zeitpunkte während der Jobausführung 73
25 Auslastung der Ressourcen mit kurzen Jobs (aus der Sicht des Grid-Schedulers) 77
26 Auslastung der Ressourcen mit langen Jobs (aus der Sicht des Grid-Schedulers) 78
27 Auslastung des Grids mit kurzen Jobs 79
28 Auslastung des Grids mit langen Jobs 79

1 Einleitung

Dieses Kapitel stellt das Ziel von Grid Computing vor und beschreibt die Aufgabenstellung und die Gliederung der Diplomarbeit.

1.1 Grid Computing

Die explosionsartige Entwicklung des Internets beruht auf der Standardisierung der Kommunikation zwischen heterogenen Systemen. Einen ähnlichen Schritt wird vermutlich auch das Grid Computing machen. Die zunehmende Bandbreite zwischen heterogenen Systemen und die zunehmende Standardisierung, Ressourcen zu teilen, ermöglicht es, heterogene Systeme zu einem virtuellen Supercomputer (Grid) zu verbinden. Dieser liefert dann bei Bedarf die benötige Rechenleistung.

Die Einsatzmöglichkeiten für solch einen virtuellen Supercomputer sind vielfältig. Hauptsächlich wissenschaftliche und technische Berechnungen, welche in hohem Grade aus parallel ausführbaren Aufgaben bestehen, können das große Potential der Kapazität an parallelen Prozessoren nutzen. Beispiele für solche Anwendungen sind Simulationen zur Erforschung von Erdbeben [Futrelle, 2001], Simulationen von Crashtests [Netera Alliance, 2003], Anwendungen der Gentechnologie [Larson, 2003] oder die Berechnungen von Computeranimationen für Kinofilme [Netera Alliance, 2003]. Gleichzeitig kann nicht nur die Rechenleistung, sondern auch die anderen Ressourcen eines Rechners, wie z.B. der Speicherplatz, können gemeinsam genutzt werden.

Das „LabGrid“ von IBM unterstützt bis jetzt nur Simulationsjobs zur Entwicklung von Mikroprozessoren für die Großrechnerfamilie. Mit dem Einsatz des LabGrids ist es möglich, die nicht notwendigerweise dedizierten Rechnerressourcen der unterschiedlichen Organisationen einzubeziehen und das rechenzeitintensive Chip-Design und die Verifikationssimulation zu beschleunigen.

1.2 Themenstellung

Die Aufgabenstellung der Diplomarbeit lautet folgendermaßen (aus der Ausschreibung zitiert):
„... Das IBM Entwicklungslabor Böblingen hat ein internes, auf dem Globus Toolkit basierendes Grid entwickelt. Momentan ist das Scheduling jedoch sehr stark auf eine bestimmte Art von Simulationsjobs zugeschnitten und kann nicht ohne weiteres für Anwendungen verwendet werden, die andere Anforderungen mitbringen. Im Rahmen der Diplomarbeit soll, basierend auf dem existierenden Grid und dem verwendeten Scheduler, ein Prototyp entwickelt werden, welcher es ermöglicht, ein Matchmaking auf der Basis von Ressourcenanforderungen von Anwendungen und Informationen über die zur Verfügung stehenden Ressourcen durchzuführen. Mit dessen Hilfe sollen die potentiell in Frage kommenden Ressourcen (Systeme) ermittelt werden. Weiterhin sollen unterschiedliche Scheduling-Policies für die Auswahl einer konkreten Ressource unterstützt werden. Die Diplomarbeit soll sich auf folgende Bereiche konzentrieren:

- Untersuchung heute existierender Scheduler und Scheduling-Technologien

- Entwicklung eines Frameworks, welches eine Matchmaking-Funktionalität zur Verfügung stellt und welches es ermöglicht, unterschiedliche Scheduling-Policies einzubeziehen
- Implementierung mindestens einer konkreten Scheduling-Policy
- Test der Prototyp-Implementation (als Testbett kommt das Labor Grid zum Einsatz)

Die Ergebnisse der Diplomarbeit sind zu dokumentieren und zu präsentieren.“

1.3 Gliederung

Die Diplomarbeit verdeutlicht in Kapitel 2 die Notwendigkeit eines Grids und führt systematisch in die Grundlagen der Funktionsweise und in die Architektur eines Grids ein. Desweiteren werden die Grundlagen des Schedulings, d.h. der Aufteilung der Arbeit, behandelt.

In Kapitel 3 wird die Funktionsweise und die Architektur des Grids der IBM vorgestellt. Es werden einige Nachteile des aktuell verwendeten Schedulers angesprochen und in Kapitel 4 die Anforderungen für die Lösung der Probleme spezifiziert.

Kapitel 5 gibt einen Überblick über die Erfüllung der Anforderungen und betrachtet dazu bekannte, sich in der Architektur unterscheidende Matchmaking-Techniken und Grid-Scheduler, die für unterschiedliche Anforderungen und Einsatzgebiete erstellt wurden. Für die Entwicklung einer neuen Scheduling-Policy werden unterschiedliche Scheduling-Algorithmen vorgestellt.

Kapitel 6 befasst sich mit dem Entwurf der Architektur des Frameworks und diskutiert mögliche Alternativen.

Kapitel 7 erläutert den Prototyp, der zur Evaluierung des Frameworks erstellt wurde. Die Schnittstellen des Frameworks und die neu erstellte Scheduling-Policy werden beschrieben.

Kapitel 8 enthält die Evaluierung des Frameworks und untersucht, wie sich die neue Policy im Gegensatz zu der ursprünglichen Policy verhält. Die zu diesem Zweck erhobenen Messgrößen und die verschiedenen Testläufe werden dokumentiert. Um die Flexibilität des Frameworks zu untersuchen, wird eine heterogene Ressource in das Grid eingefügt. Die Ergebnisse werden präsentiert und diskutiert.

Kapitel 9 fasst die erzielten Ergebnisse zusammen, und Kapitel 10 gibt einen Ausblick auf zukünftige Arbeiten und mögliche Erweiterungen an diesem Framework.

2 Grundlagen

Zum besseren Verständnis der Diplomarbeit erläutert dieses Kapitel die Grundlagen und verwendeten Technologien von Grid Computing.

2.1 Das Grid

2.1.1 Einführung

Grid Computing ist Mitte der 90er Jahren aus dem Bedarf nach mehr Rechenkapazität aus den Bereichen Heterogene Computing und Metacomputing entstanden. Forschungseinrichtungen aus den verschiedensten Wissenschaftsbereichen stehen vor immer komplexeren Problemen. Zur Lösung dieser Probleme werden enorme Rechen- und Speicherkapazitäten benötigt. Weil ein einzelner Supercomputer die benötigten Anforderungen nicht mehr alleine bereitstellen kann, oder dieser nicht zur Verfügung steht, wurde damit begonnen, vorhandene und zum Teil nicht optimal genutzte Rechnersysteme (Ressourcen) miteinander zu verbinden, um damit einen virtuellen Supercomputer zu schaffen. Das Ziel dabei ist es, Ressourcen gemeinsam und koordiniert zur effektiveren und effizienteren Problemlösung einzusetzen.

Der Begriff „Grid“ ist von dem englischen Begriff „electrical power grid“ abgeleitet. Er stellt eine Verbindung zur Einführung des Stromnetzes um 1910 in den USA her. Zu diesem Zeitpunkt war die Stromerzeugung schon möglich, jedoch brauchte jeder Benutzer sein eigenes kleines Kraftwerk in Form eines Generators. Dieser Umstand hinderte die Verbreitung elektrischer Geräte. Erst mit der Einführung des Stromnetzes und der Standardisierung des Zugangs war es für jeden Benutzer möglich, zuverlässig und kostengünstig je nach Bedarf Strom zu bekommen. Das gleiche gilt auch für die Gas- und Wasserversorgung.

Diese Idee der Infrastruktur, um Ressourcen zu teilen, wird auch beim Grid verfolgt. Der Benutzer bekommt in diesem Fall keinen Strom, sondern die verschiedenen Leistungen der verfügbaren Ressourcen, z.B. Rechenleistung, Zugriff auf Daten oder auch die Möglichkeit Peripheriegeräte zu nutzen. Andere erfolgreiche Infrastrukturen sind das Eisenbahnnetz, das Telefonnetz und schließlich das Internet. Das Grid wird von vielen als die zweite Generation des Internets angesehen, da das Grid auf dem Internet aufbaut und weit mehr Möglichkeiten als nur den Zugang zu Informationen bietet [Foster, 1999].

2.1.2 Definition

In verschiedenen Veröffentlichungen aus unterschiedlichen Jahren zum Thema Grid Computing von Ian Foster und Carl Kesselman sind folgende Aussagen enthalten:

- 1998: „A computional grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities.“ [Foster, 1999]

- 2001: „The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.“ [Foster, 2001]
- 2002: „We view a Grid as an extensible set of Grid services that may be aggregated in various ways to meet the needs of Virtual Organisations.“ [Foster, 2002a]

Die erste Aussage zielt auf die sichere und zuverlässige Bereitstellung und den transparenten Zugang für Endbenutzer ab. Die Aussage von 2001 dagegen behandelt mehr das gemeinsame und koordinierte Nutzen von Ressourcen über Unternehmensgrenzen hinweg, und die letzte Definition bezieht den Trend zu Web Services mit ein.

In dieser Diplomarbeit wird als eine **mögliche Definition** für ein Grid in Anlehnung an [Foster, 1999] folgende Definition verwendet:

„Ein Computional Grid ist eine Hard- und Softwareinfrastruktur, die einen zuverlässigen, beständigen, überall vorhandenen und kostengünstigen Zugang zu den leistungsfähigsten Computerressourcen bietet. Dabei ist die geographische Verteilung von Benutzern und Ressourcen für die Verwendung ohne Bedeutung. Ein Grid kann die verschiedensten Ressourcen zur Verfügung stellen. Neben Rechen- und Speicherkapazität, Bandbreite und Softwarelizenzen ist es auch in der Lage andere Komponenten, die für den gemeinsamen Zugriff freigegeben werden können, wie z.B. ein Elektronenmikroskop oder ein verteiltes Dateisystem, zur Verfügung zu stellen.“

2.1.3 Funktionsweise

Der Benutzer übergibt eine Anwendung und Informationen, unter welchen Bedingungen die Anwendung auszuführen ist, an das Grid. Dieses hat nun die Aufgabe, eine passende und freie Ressource im Grid zu finden und die Anwendung darauf auszuführen. Sobald die Ausführung beendet ist, werden die Ergebnisse an den Benutzer zurückgegeben.

2.1.4 Virtuelle Organisation

Grids sind nicht auf einzelne Abteilungen oder Organisationen beschränkt und können als sog. Virtuelle Organisationen (VO) organisationsübergreifend dynamisch erstellt werden. Dynamisch bedeutet in diesem Zusammenhang, dass Ressourcen zur Laufzeit hinzukommen oder nicht mehr zur Verfügung stehen. Damit der gemeinsame Zugriff koordiniert erfolgt, gibt es Nutzungsregeln. Diese regeln, wer wann welche Ressource zu welchen Bedingungen nutzen darf. Eine Gruppe von Personen oder Organisationen, die sich auf der Grundlage solcher Nutzungsbedingungen zusammenschließen, nennt man VO [Foster, 2001]. Sobald die Größe einer VO Abteilungs-, Organisations- oder Landesgrenzen überschreitet, gewinnen Sicherheitsaspekte wie Authentifizierung, Autorisierung und Geheimhaltung an

Bedeutung [Foster, 2001]. Auf das Thema Sicherheit wird in dieser Diplomarbeit nicht näher eingegangen, weil für die Arbeit nur Komponenten verwendet werden, die auf einer vorhandenen Sicherheitsschicht (siehe Kapitel 3.1.2) aufbauen. Somit ist die grundlegende Sicherheit gegeben und wird nicht weiter berücksichtigt.

2.1.5 Grid-Scheduler

Der Grid-Scheduler (siehe Abb. 1) hat die Aufgabe, für eine Anwendung eine geeignete Ressource zu finden und diese Anwendung dort auszuführen. Informationen über vorhandene Ressourcen bekommt er von einem Informationsdienst, bei dem sich die Ressourcen registriert haben.
Die Vorauswahl der Ressourcen ist die **Aufgabe des Matchmakings** und bezeichnet den Prozess, potentiell geeignete Ressourcen anhand deren Eigenschaften und den Anforderungen der Anwendung herauszufiltern.
Die Effektivität und Effizienz eines Grids hängt entscheidend von der Erstellung des Ausführungsplans ab. Er wird von dem Scheduler erstellt und regelt, wer wann welche Ressourcen zu welchen Bedingungen nutzen darf. Damit die Nutzung der Ressourcen des Grids koordiniert erfolgt, werden **Scheduling-Policies** benötigt. Diese verteilen nach bestimmten Regeln die Aufgaben auf die vom Matchmaking ausgewählten potentiellen Ressourcen.
Der Algorithmus für die Verteilung muss dabei Anforderungen erfüllen, die im Konflikt miteinander stehen:

- Ein Benutzer des Grids erwartet eine möglichst kurze Ausführungszeit seiner Anwendung.
- Der Betreiber der Ressourcen verlangt eine möglichst gleichmäßige Auslastung seiner Ressourcen. Ein Supercomputer wird anhand seiner durchschnittlichen Rechenleistung und nicht an der maximalen Rechenleistung zu einem bestimmten Zeitpunkt gemessen.
- Der Manager muss die gerechte Verteilung berücksichtigen und will somit eine gleichmäßige Auslastung und eine akzeptable Antwortzeit.

Das Ergebnis einer Verteilung ist der Ausführungsplan, der in Kapitel 2.3 näher betrachtet wird.

2.2 Architektur eines Grids

Dieses Kapitel stellt zunächst die beteiligten Systemkomponenten und dann eine mögliche Grid-Architektur vor.

2.2.1 Systemkomponenten

Ein Grid wird aus vorhandenen Systemkomponenten [Foster, 2001] aufgebaut, in dem sie untereinander vernetzt werden. Es können vier Klassen von Systemkomponenten unterschieden werden:

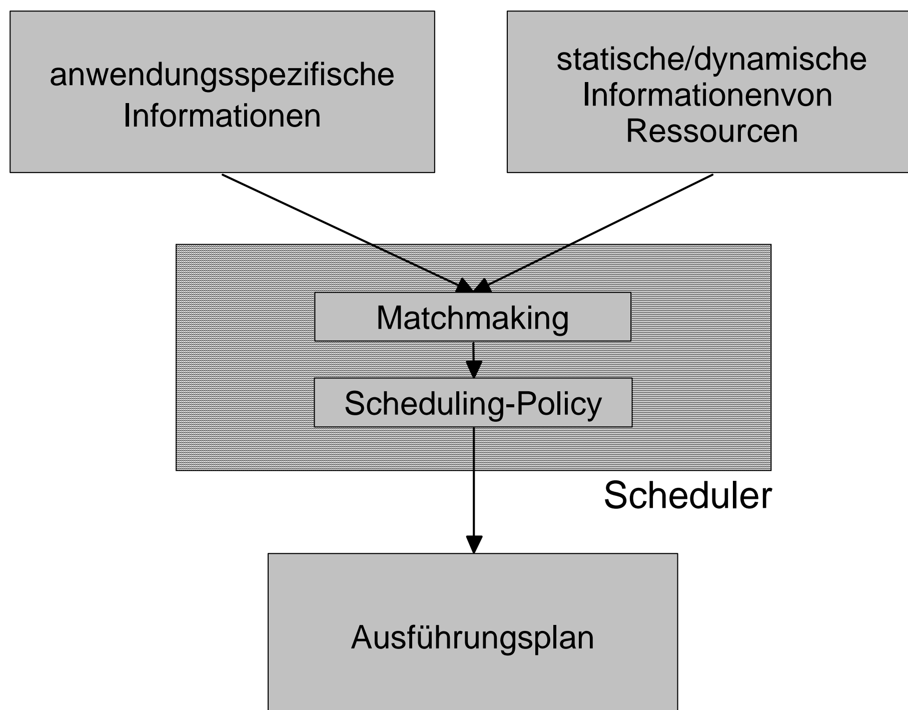


Abbildung 1: Architektur eines Grid-Schedulers

- Die kleinste Komponente in einem Grid ist das **Endsystem**, zu dem ein einzelner Rechner, ein Speichersystem, ein Sensor oder andere Geräte gehören. Die Eigenschaften, die diese Endsysteme auszeichnen sind, dass sie homogen, hochintegriert und in ihren physikalischen Ausmaßen kompakt sind. Die einzelnen Komponenten eines Endsystems sind dabei aufeinander abgestimmt, und die Software auf dem Endsystem hat die alleinige Kontrolle über die Komponenten. Damit wird eine einfache Verwaltung und eine hohe Performance gewährleistet.

- Als zweite Klasse können **Cluster** ausgemacht werden. Ein Cluster sind Workstations die durch ein Hochgeschwindigkeitsnetzwerk verbunden sind. Die einzelnen Workstations unterscheiden sich nur durch ihre Konfiguration. Somit besteht ein Cluster aus homogenen Systemen, die über eine administrative Einheit kontrolliert werden. Nach außen verhält sich ein Cluster wie eine leistungsstarke Workstation, nur dass ein höherer Grad an Parallelisierung und Verteilung hinzukommt.

- Werden Ressourcen innerhalb einer Organisation vernetzt, spricht man von einem **Intranet**. Dieses besteht im Unterschied zu Clustern meist aus heterogenen und räumlich weiter entfernten Ressourcen, die zum Teil getrennt verwaltet werden. Hier ist auch erstmals der Grad erreicht, wo ein Benutzer nicht mehr den Überblick über sämtliche Ressourcen und deren Zustand hat.

- Die letzte Klasse ist das **Internet**, welches aus einer Vielzahl an heterogenen Netzen besteht. Im Unterschied zum Intranet sind unterschiedliche Organi-

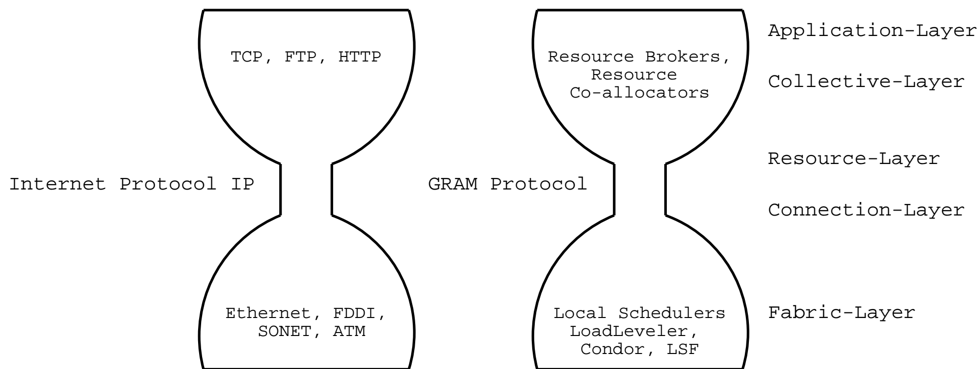


Abbildung 2: Sanduhren-Prinzip

sationen und Nationen eingebunden, und es gibt keine zentrale Verwaltung über die Ressourcen.

Grid Computing arbeitet mit sehr viel höheren Datenmengen und benötigt Datenraten, für die das Internet zum jetzigen Zeitpunkt noch nicht ausgebaut ist, um es flächendeckend, wie z.B. das Stromnetz, bereitstellen zu können.

2.2.2 Beispiel einer Grid-Architektur

Das Globus-Projekt (siehe Kapitel 3.1) setzt eine mögliche Grid-Architektur um, die hier kurz vorgestellt wird.

Um die Architektur so flexibel und so unkompliziert wie möglich zu machen, wird bei Globus ein Schichtenmodell eingesetzt. Dabei sind die Schnittstellen zwischen den einzelnen Schichten „durchsichtig“. Das bedeutet, dass höhere Schichten die Funktionalität und Funktionsweise niedrigerer Schichten entdecken und direkt nutzen können. Durch das geschickte Design können die Schnittstellen schlank und universell gehalten werden. Die Mächtigkeit führt in diesem Fall nicht zwangsweise zu komplizierteren Schnittstellen. Dies erlaubt höheren Schichten eine höhere Performance und einen höheren Grad an Funktionalität, weil für sie die Funktionsweise der niedrigeren Schichten bekannt ist. Bei den Netzwerkprotokollen des Internets wird dagegen das Verheimlichungsprinzip angewendet. Höhere Schichten kennen nur die Funktionalität aber nicht die Funktionsweise der darunterliegenden Schicht. Das kann zu Einbußen in der Performance führen, wenn z.B. eine Fehlerkorrektur doppelt auf unterschiedlichen Schichten gemacht wird.

Die Schichtenstruktur [Foster, 2001][Globus, 2003a], hier im Vergleich mit den Schichten des Internets, kann mit dem sog. Sanduhrenmodell in Abbildung 2 veranschaulicht werden:

Im Internet wird die Interoperabilität zwischen heterogenen Systemen durch das IP-Protokoll gewährleistet. Es kapselt die verschiedenen netzwerkspezifischen Eigenschaften und bietet den höheren Protokollen, wie z.B. TCP, eine Schnittstelle an, um Datenpakete an einen beliebigen Rechner im Netzwerk zu verschicken. In dem Modell des Grids übernimmt die Verbindung zu den einzelnen lokalen Resourcemanager die Connectivity- und die Resource-Schicht. Dabei gilt genauso

wie bei dem IP-Protokoll, die Schnittstelle schlank und die Anzahl der Protokolle möglichst gering zu halten. Dies wird im Sanduhrenmodell mit dem schlanken Hals verdeutlicht und dient als Basis für die Implementierung höherer und niedrigerer Schichten. Damit ist die Interoperabilität gewährleistet.

1. **Fabric-Schicht**
   Die unterste Ebene bildet die Fabric-Schicht. Zu ihr gehören alle im Grid zur Verfügung stehenden Ressourcen. Auf diese Schicht erfolgt der Zugriff über die Grid-Protokolle.

   Ressourcen können von unterschiedlicher Natur sein und stellen daher auch spezifische Funktionen zur Verfügung. Für den Zugriff durch höhere Schichten wird eine standardisierte Schnittstelle zur Abfrage der Fähigkeiten und des Zustands einer Ressource, sowie deren Steuerung benötigt. Mögliche Beispiele für spezifische Funktionen sind:

   - Rechenressourcen benötigen Mechanismen, um die Ausführung von Programmen zu steuern.
   - Speicherressourcen benötigen Mechanismen, um den Zugriff und das Ablegen von Daten zu steuern.
   - Netzwerkressourcen benötigen Mechanismen, um den Netzwerkverkehr zu steuern.

2. **Connectivity-Schicht**
   Die Connectivity-Schicht legt die Kommunikations- und Authentifizierungsprotokolle für Grid-spezifische Netzwerktransaktionen fest. Dabei wird das Kommunikationsprotokoll für den Austausch von Daten zwischen den einzelnen Ressourcen der Fabric-Schicht benötigt. Die Authentifizierungsprotokolle bieten kryptographisch sichere Mechanismen, um die Authentizität der Benutzer und der verwendeten Ressourcen zu überprüfen.

3. **Resource-Schicht**
   Die Resource-Schicht definiert Protokolle zur Verwaltung einzelner Ressourcen der Fabric-Schicht. Es kann zwischen zwei Protokollen unterschieden werden:

   - Informations-Protokolle zur Ermittlung der Struktur und des Zustands einer Ressource
   - Management-Protokolle, um den Zugriff auf Ressourcen auszuhandeln

4. **Collective-Schicht**
   Die Collective-Schicht bietet im Gegensatz zur Resource-Schicht Protokolle, Dienste, Application Programming Interfaces (API) und Software Development Kits (SDK) an. Diese werden zur Interaktion zwischen mehreren Ressourcen benötigt. Weil die Komponenten der Collective-Schicht auf dem schmalen Hals des Sanduhrenmodells der Connectivity- und Resource-Schicht aufsetzen, ist es möglich eine Vielzahl an Diensten durch die Nutzung der Protokolle, Dienste, APIs und SDKs zu implementieren, ohne neue Anforderungen an die Ressourcen stellen zu müssen [Foster, 2001]. Mögliche Dienste sind:

- Directory Services zum Finden und Abfrage von Ressourcen
- Co-allocation, Scheduling und Brokering Services zur Reservierung von Ressourcen, Planung und Verteilung von Tätigkeiten auf reservierte Ressourcen
- Monitoring und Diagnostics Services zur Überwachung der Ressourcen auf Fehler, Überlastung usw.

5. **Application-Schicht**
   Als letzte Schicht enthält die Application-Schicht alle Anwendungen, die innerhalb einer virtuelle Organisation verwendet werden. Dabei verwenden die Anwendungen die APIs der darunterliegenden Schichten.

2.2.3 Anwendungen

Es folgt eine Aufzählung von möglichen Anwendungen, für die ein Grid geeignet wäre [Foster, 1999]. Weitere Anwendungen sind denkbar und werden mit der Verbreitung des Grids entstehen, wie es auch bei Internetanwendungen geschieht.

- **Verteiltes Supercomputing**
  Dazu zählen Anwendungen, die eine sehr hohe Rechenleistung oder Speicherkapazität benötigen. SETI@HOME [Korpela, 2003] oder das ZetaGrid [Zeta, 2003] sind Beispiele für solche Anwendungen und werden auch als „Cycle Scavenging Systeme“ bezeichnet. Bei SETI@HOME werden Frequenzmuster aus dem Weltall analysiert, und beim ZetaGrid werden Nullstellen der Riemannschen-Zeta-Funktion berechnet. Die Berechnungen der Anwendungen geschieht auf mehreren Mainframes und Tausenden von PCs, solange diese nicht für ihren eigentlichen Zweck benötigt werden. Damit wird eine sehr hohe Rechenleistung genutzt.

  Je besser ein Problem unterteilbar ist, desto besser lassen sich die Teilprobleme unabhängig voneinander bearbeiten. Damit ein Problem effizient verteilt berechnet werden kann, muss der Overhead für die Kommunikation zwischen den Teilaufgaben wegen der globalen Verteilung der Ressourcen möglichst gering sein. Die Kommunikation kann beispielsweise über das Message Passing Interface (MPI) erfolgen. Für das senden und empfangen von Nachrichten werden standardisierte Unterprogramme der MPI-Library aufgerufen.

- **Aufgaben mit hohem Datendurchsatz**
  Kryptographische Probleme benötigen einen hohen Datendurchsatz und können auf vielen Systemen durch unabhängige Teilaufgaben gelöst werden.

- **Rechenleistung je nach Bedarf (Computing on Demand)**
  Manche Anwendungen benötigen nur zu bestimmten Zeitpunkten hohe Rechenleistungen. In diesen Fällen lohnt sich die Anschaffung von entsprechenden Systemen nicht. Zu solchen Anwendungen gehört z.B. das Rendern von computer-animierten Filmen oder CAD-Modellen oder auch das Routing eines Leiterplatten- oder Prozessorlayouts.

- **Datenintensive Aufgaben**
  Zu den datenintensiven Aufgaben gehören Anwendungen, die riesige Datenmengen aus unterschiedlichen Datenquellen beziehen. Sie werden zur Erzielung von neuen Erkenntnissen eingesetzt z.B. in der Medizin bei der Analyse von Krankheitsbildern. Das CERN (European Organization for Nuclear Research) [CERN, 2003] entwickelt Lösungen, um damit riesige Datenmengen im Bereich von Petabytes, die bei Experimenten mit ihrem Teilchenbeschleuniger anfallen, zu untersuchen.

- **Kollaboratives Arbeiten mit Computern**
  CAVE5D [CAVE5d, 1996] ist eine Anwendung zur Visualisierung von Gewässerdaten. Es ermöglicht, gemeinsam physikalische Modelle von unterschiedlichen Standorten aus weiter zu entwickeln.

Neben diesen Anwendungen gibt es auch Anwendungen, für die das Grid nicht geeignet ist. Zu diesen zählen Programme mit interaktiven Benutzerschnittstellen, die eine sehr schnelle Reaktionszeit vom System fordern. Diese kann aufgrund der globalen Verteilung der Ressourcen und den Verzögerungszeiten der Netzwerke nicht erreicht werden. Ein Textverarbeitungsprogramm ist ein mögliches Beispiel.

2.3 Scheduling

Einer der entscheidensten Faktoren für die Auslastung und Performance des Grids ist das Scheduling. Die Aufgabe des Schedulings besteht darin, festzulegen, wann welche Aufgabe auf welchen freien Ressourcen ausgeführt wird.

Eine Aufgabe wird auch **Job** genannt und kann aus einzelnen unabhängigen Teilaufgaben, den **Tasks**, bestehen. Der Scheduler erstellt als Ergebnis den Ausführungsplan.

2.3.1 Erstellung des Ausführungsplans

Es gibt unterschiedliche Zeitpunkte für die Erstellung des Ausführungsplans. Folgende zwei Arten können hier unterschieden werden:

- **Offline**
  Wenn alle Informationen, wie Ankunftszeit, Ausführungsdauer der Jobs im voraus bekannt sind, kann der Ausführungsplan offline erstellt werden. Das bedeutet, dass der komplette Ausführungsplan vor der Ausführung der Jobs berechnet wird. Weil dieser Ausführungsplan mindestens soviele Informationen wie ein Ausführungsplan, der online erstellt wurde, hat, ist er meistens besser, weil die Berechnungszeit des Ausführungsplans im Verhältnis zu der Laufzeit der Jobs vernachlässigbar ist.

- **Online**
  Falls nicht alle Informationen im voraus bekannt sind, muss der Ausführungsplan online erstellt werden. Dabei kann die Erstellung nach jedem Eintreffen eines Jobs oder nach dem Eintreffen einer bestimmten Anzahl von Jobs erfolgen:

- **interaktiver Modus**
  Sofort bei der Ankunft wird ein Job einer Ressource zugeordnet. Typische Scheduling-Policies für den interaktiven Modus sind OLB (oppertunistic load balancing), MCT (minimum completion time) und MET (minimum execution time) und werden in Kapitel 5.3 näher betrachtet.
- **Batch-Modus**
  Im Batch-Modus wird erst eine bestimmte Anzahl oder alternativ eine bestimmte Zeit lang Jobs gesammelt, und dann periodisch den Ressourcen zugeordnet. Zu den bekannten Algorithmen gehört der Min-Min- und Max-Min-Algorithmus. Diese werden ebenfalls in Kapitel 5.3 vorgestellt.

Eine Möglichkeit für eine Scheduling-Policy wäre, dass die Jobs per Round-Robin reihum auf die Ressourcen verteilt werden. Aufwendigere Implementierungen von Schedulern berücksichtigen ein oder mehrere Eigenschaften der Ressourcen und Jobs, um die Performance der auszuführenden Jobs und die Auslastung des Grids zu erhöhen. Einige Schedulerimplementierungen geben mit der Hilfe von Prioritäten bestimmten Jobs den Vorzug.

Weiterentwickelte Scheduler überwachen die Jobausführung. Falls ein Job auf Grund eines Systemfehlers auf einer Ressource nicht komplett ausgeführt werden kann, wird dieser auf einer geeigneten Ressource neu gestartet. Den Vorgang der Optimierung des Ausführungsplans nennt man **Rescheduling**. In diesem Zusammenhang gibt es das **Checkpointing** [Litzkow, 1997], das es ermöglicht, den aktuellen Stand der Ausführung eines Jobs festzuhalten und den Job zu einem späteren Zeitpunkt fortzusetzen. Diese Technik wird in dem Grid-Scheduler Condor-G [Frey, 2001] angewendet. Falls der Job auf einer anderen Ressource fortgesetzt werden soll, weil er z.B. nicht schnell genug ausgeführt, oder die Ressource von einem lokalen Benutzer benötigt wird [Frey, 2001], muss er übertragen werden. Diesen Vorgang nennt man **Migration**. Für den Fall, dass der Job in eine Endlosschleife läuft und eine festgelegte Zeit überschreitet, darf er nicht erneut gestartet werden, weil es sonst zu einer unnötigen Belastung des Grids führt, und sich dieser Vorgang mit hoher Wahrscheinlichkeit endlos wiederholen würde.

2.3.2 Schedulingarten

Es können zwei Arten von Scheduling unterschieden werden. Zum einen ist es das **lokale Scheduling**, das die Zuweisung auf der Seite der Ressource durchführt. Bei einem Cluster als Ressource verteilt der lokale Scheduler die Jobs auf die einzelnen Workstations innerhalb des Clusters. Im Gegensatz zum lokalen Scheduler arbeitet der **Grid-Scheduler** in der Hierarchie oberhalb den lokalen Schedulern (siehe Abbildung 3). Der Grid-Scheduler verteilt Jobs an die lokalen Scheduler oder an andere Grid-Scheduler. Mit Grid-Schedulern ist man in der Lage, andere Grids mit Jobs zu bedienen und somit eine Hierarchie aufzubauen.

Der große Unterschied zwischen einem lokalen und einem Grid-Scheduler ist, dass der Grid-Scheduler nur eine globale Sicht und keine Kontrolle über die Ressourcen hat. Weiterhin verwendet er Informationen von Informationsdiensten. Diese

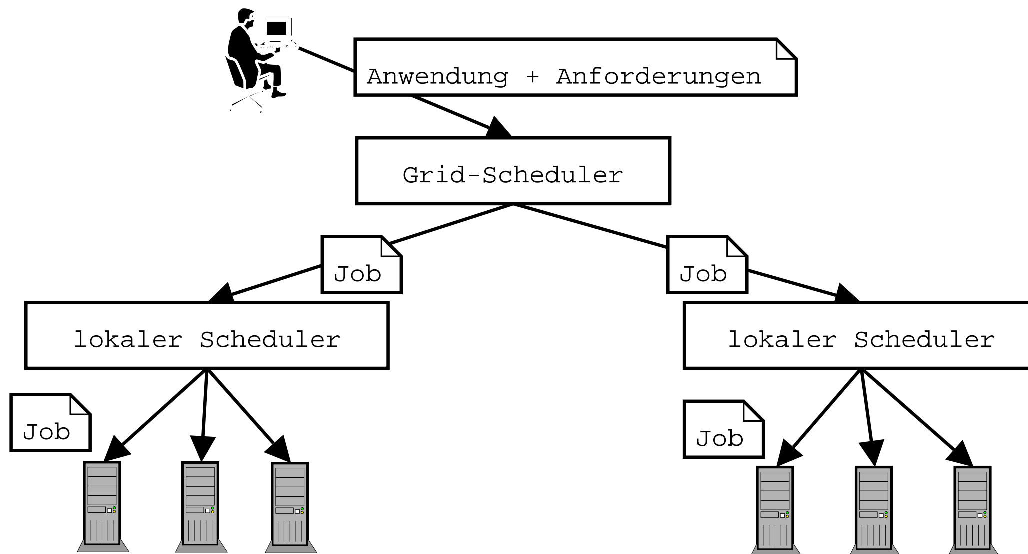


Abbildung 3: Scheduling

Informationen entsprechen nicht unbedingt dem aktuellen Zustand der Ressource, und der Scheduler erstellt damit einen nicht optimalen Ausführungsplan.

Der Unterschied zwischen einem lokalen und einem Grid-Scheduler wird in Kapitel 2.3.4 genauer betrachtet.

2.3.3 Lokaler Scheduler am Beispiel LoadLeveler

LoadLeveler [Kannan, 2001] ist eine Clustersoftware bzw. ein lokaler Scheduler von IBM. Er hat die Aufgabe serielle und parallele Tasks auf einem Cluster von Workstations auszuführen (siehe Abbildung 3). Workstations können in dem Cluster Desktops oder dedizierte Server sein. Dem Scheduler werden die auszuführenden Jobs übergeben, und dieser verteilt die Tasks auf die einzelnen Rechner im Cluster. Der Scheduler versucht die Tasks auf die zur Verfügung stehenden Rechner so zu verteilen, dass die höchstmöglichste Effizienz des Clusters erreicht wird. LoadLeveler besitzt drei interne Scheduler [Kannan, 2001], die je nach Bedarf verwendet werden. Externe Scheduler können über die API angebunden und verwendet werden.

Sobald ein Job an den Scheduler übergeben wird, wird er anhand seiner Priorität in eine der FIFO-Warteschlangen abgelegt. Die Priorität eines Jobs kann implizit in einer Konfigurationsdatei des LoadLeveler gesetzt werden oder explizit durch den Benutzer bei der Übergabe des Jobs. Ein Job ist an der Reihe, wenn er die höchste Priorität hat. Die Priorität eines Jobs kann als eine Formel in einer Konfigurationsdatei definiert werden. Die Formel kann z.B. die Priorität eines Benutzers oder die Zeit, seit der Job in der Warteschlange ist, berücksichtigen. Ist ein Job an der Reihe, müssen für die Ausführung noch die Ressourcenanforderungen der Tasks erfüllt sein.

Falls nicht genügend Knoten verfügbar sind, werden die verfügbaren Knoten reserviert und gewartet, bis genügend Knoten reserviert sind, um die Tasks schließlich ausführen zu können. Sind mehr Knoten, als benötigt werden, verfügbar, werden je nach durchschnittlicher Auslastung der Knoten innerhalb der letzten Minute, die Knoten mit der geringsten Auslastung gewählt. Diese Auswahl kann durch eine weitere Formel in der Konfigurationsdatei beeinflusst werden. Somit ist die Auswahl unabhängig von den folgenden Scheduling-Algorithmen, die lediglich den Zeitpunkt der Ausführung bestimmen.

Man unterscheidet bei LoadLeveler folgende Scheduler:

- **LL_DEFAULT-Scheduler**
  Bei seriellen Tasks betrachtet der LoadLeveler den ersten Job in der Warteschlange. Falls die Ressourcenanforderungen erfüllt sind, wird er ausgeführt. Müssen parallele Tasks ausgeführt werden, allokiert der Scheduler die entsprechende Anzahl an Knoten und führt die Tasks aus. In dem Fall, dass nicht genügend Knoten zur Verfügung stehen, reserviert der Scheduler die zur Verfügung stehenden Knoten. Jedesmal, wenn ein Task beendet ist, wird ein Knoten verfügbar und wird reserviert. Dieser Vorgang wiederholt sich solange, bis genügend Knoten reserviert sind, um den Job ausführen zu können.

  Dieser Algorithmus hat zwei Nachteile. Zum einen nimmt die Auslastung des Clusters progressiv ab, sobald der Scheduler beginnt, nach und nach Knoten zu reservieren. Das zweite Problem ist, dass ein Task einen Knoten nur eine bestimmte Zeit reservieren darf. Falls der Job es nicht schafft in dieser Zeit genügend Knoten zu reservieren, wird der entsprechende Knoten wieder freigegeben.

- **BACKFILL-Scheduler**
  Der BACKFILL-Scheduler bietet eine Lösung für diese Nachteile. Dafür benötigt er die Angabe, wie lange ein Job maximal eine Ressource belegt. BACKFILL nimmt den nächsten Job aus der Warteschlange und führt diesen aus oder reserviert die entsprechende Anzahl der Knoten in der Zukunft. Falls es nötig ist, Knoten zu reservieren, kann er die Zeit, bis diese Knoten in der Zukunft reserviert sind, für die nachfolgenden Jobs aus der Warteschlange nutzen. Dabei darf sich die Ausführung der vorgezogenen Jobs nicht mit der Reservierung anderer Jobs überlappen. Somit ist BACKFILL viel effizienter als der LL_DEFAULT-Scheduler.

- **GANG-Scheduler**
  Der GANG-Scheduler bietet die Möglichkeit, im Gegensatz zu LL_Default und BACKFILL, mehrere Jobs gleichzeitig auf einer Gruppe von Knoten oder auf einem einzelnen Knoten auszuführen. Dies wird durch die Unterteilung der Zeit in Zeitschlitze erreicht. Nach jedem Zeitschlitz wechseln alle Prozessoren gleichzeitig den Kontext, und die nächsten Jobs können ausgeführt werden. Ein Vorteil davon ist, dass mit der Ausführung der Jobs frühzeitig begonnen wird, und diese sich nicht unnötig lange in der Warteschlange befinden. Der Benutzer hat damit die Möglichkeit, Jobs auf Grund fehlerhafter Ausgaben frühzeitig abzubrechen, oder der Job schlägt fehl, und es kann frühzeitig auf diese Situation reagiert werden.

Der GANG-Scheduler bietet noch die Möglichkeiten an, bestimmten Jobs ein Ausführungsrecht zu geben. Damit kann ein Job mit einer höheren Priorität ausgeführt werden, indem Jobs mit einer niedrigeren Priorität gestoppt werden, welche auf Grund ihrer langen Laufzeit Knoten blockiert haben. Später können die gestoppten Jobs mit der Hilfe von Checkpointing fortgesetzt werden.

2.3.4 Grid Scheduler vs. lokale Scheduler

Ein Grid-Scheduler, auch Meta-Scheduler genannt, hat die Aufgabe, Jobs auf die zur Verfügung stehenden Ressourcen zu verteilen. Dies kann z.B. ein Cluster sein, welcher von einem lokalen Scheduler bedient wird (siehe Abbildung 3).

Die Idee, modifizierte Scheduler von Massive Parallel Processor-Systemen (MPP-System; Mehrprozessorrechner) zu verwenden, funktioniert nicht. Obwohl es bei Anwendungen für MPP-Systemen auch auf eine koordinierte Verteilung der Ressourcen ankommt, gibt es einige Unterschiede zu lokalen Schedulern, die das Scheduling erschweren [Foster, 1999]:

- Grid-Scheduler haben keine Kontrolle über die Ressourcen.
- Die Resourcen befinden sich in unterschiedlichen Administrationsbereichen.
- Die Menge und Art der Ressourcen ändert sich zur Laufzeit.
- Das Verhalten der Ressourcen wird von anderen lokal ausgeführten Anwendungen beeinflusst.
- Durch die globale Verteilung und die heterogenen Ressourcen ist die verfügbare Rechenleistung und Bandbreite für die Kommunikation sehr unterschiedlich.

2.3.5 Phasen des Grid-Schedulingprozesses

Jennifer M. Schopf [Schopf, 2002] hat die Aufgaben eines Grid-Schedulers in Abbildung 4 in drei Phasen mit insgesamt elf Schritten unterteilt (siehe Abbildung 4).

1. Phase: **Ressourcen finden**
   Es wird damit begonnen, alle Grid-Ressourcen zu finden, zu denen der Benutzer überhaupt einen Zugang hat. Diese Menge an Ressourcen wird nach den minimalen Ressourcenanforderungen (z.B. minimaler Bedarf an Hauptspeicher) des zu verarbeitenden Jobs gefiltert. Die Ressourcenanforderungen werden dazu vom Benutzer an den Grid-Scheduler übergeben.

2. Phase: **Ressource auswählen**
   Damit die bestmöglichste Entscheidung getroffen werden kann, welcher Ressource der Job übergeben wird, wird aus der Menge der Ressourcen aus der ersten Phase dynamische Informationen, wie z.B. die Auslastung, abgefragt. Anhand dieser Informationen wird die entsprechende Ressource ausgewählt.

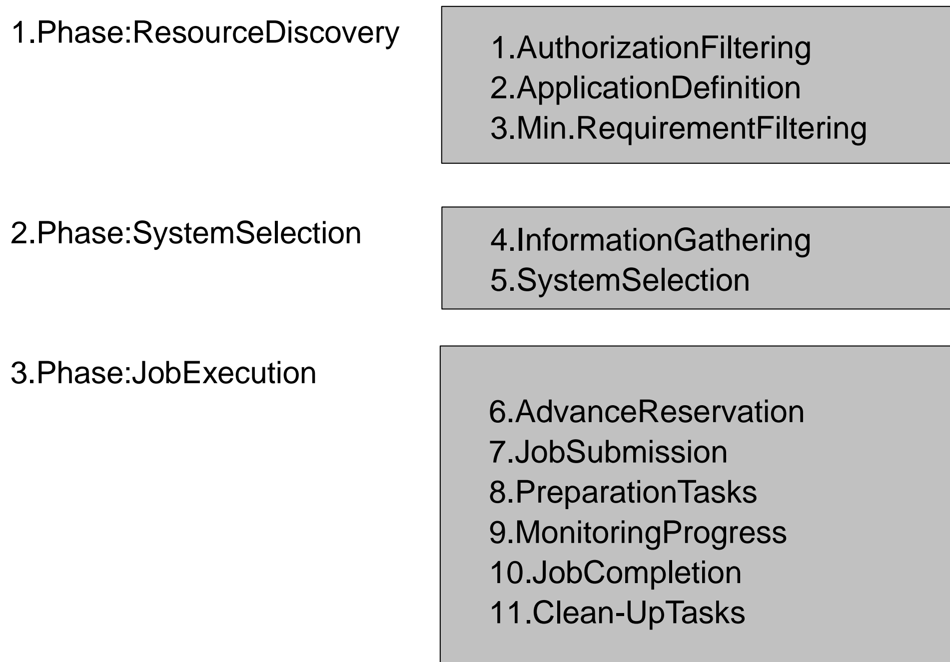


Abbildung 4: Drei-Phasen Modell des Grid-Schedulings nach Schopf [Schopf, 2002]

3. Phase: **Job ausführen**
   Unter Umständen müssen für die Ausführung des Jobs Ressourcen im voraus reserviert werden. Dafür muss die Möglichkeit der Reservierung einer Ressource angeboten werden. Danach kann der Job der ausgewählten Ressource übergeben werden. Das Problem hierbei ist es, dass jeder lokale Scheduler eine andere Aufrufsyntax mit anderen Parametern benötigt, weil sich die Hersteller noch auf keinen Standard geeinigt haben. Vor der eigentlichen Ausführung kann es sein, dass man neben der Reservierung noch Daten auf die Ressource kopieren muss (Staging). Während der Ausführung überwacht der Scheduler den Job und führt ggf. ein Rescheduling durch, wodurch wieder mit der zweiten Phase begonnen wird. Sobald der Job beendet ist, wird der Benutzer benachrichtigt. Die Ergebnisse werden ihm zur Verfügung gestellt, und die Ressourcen werden wieder in ihren Ausgangszustand zurückgesetzt.

2.3.6 Zentraler/Dezentraler Scheduler

Bei Schedulern können folgende drei - in Abbildung 5 und 6 dargestellte - Ansätze unterschieden werden [Berman, 1997][Hamscher, 2000][Subramani, 2002]:

- **Zentraler Scheduler**
  Der meist verbreitete Ansatz eines Schedulers ist der zentrale Ansatz. Bei ihm verteilt der Scheduler sämtliche Jobs auf die ihm zur Verfügung stehenden Ressourcen. Der Vorteil dabei ist, dass der zentrale Scheduler die bestmögliche Kombination bestimmen kann, auf welcher Ressource welcher

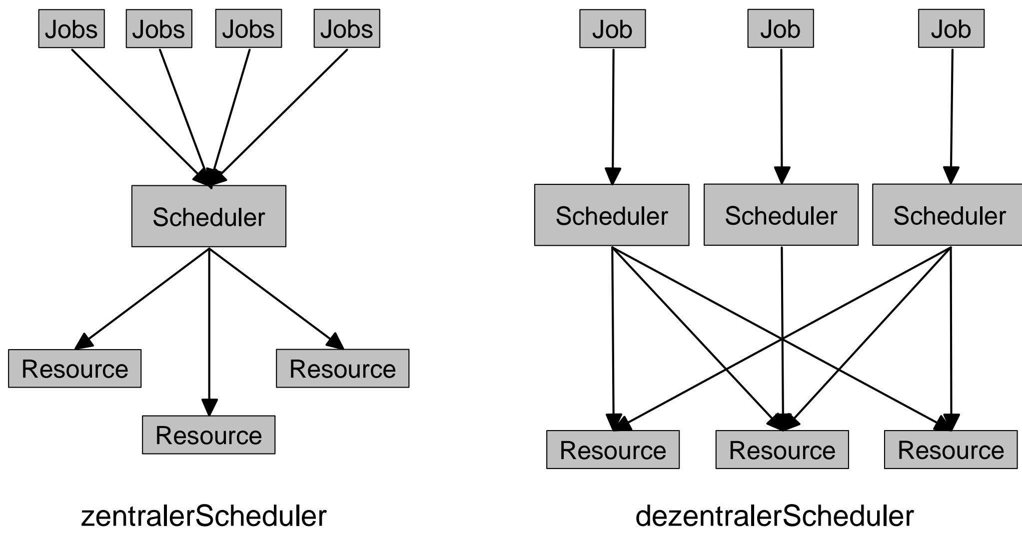


Abbildung 5: zentraler und dezentraler Scheduler

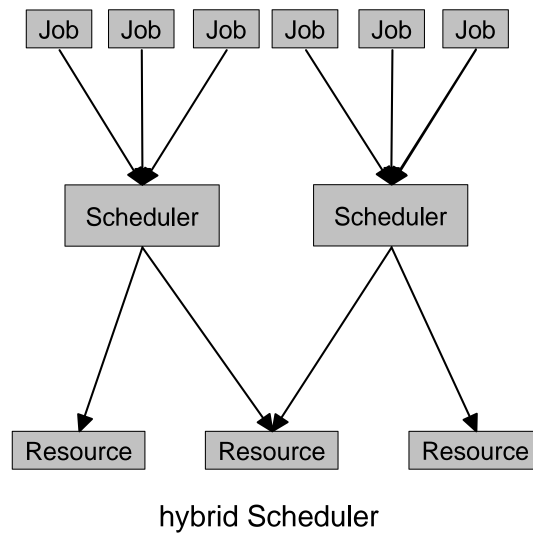


Abbildung 6: hybrid Scheduler

Job zu welchem Zeitpunkt ausgeführt wird. Dadurch ist das Ergebnis eines zentralen Schedulers immer besser als das eines dezentralen. Der Nachteil dabei ist jedoch, dass ein zentraler Scheduler einen Single-Point-Of-Failure darstellt und nicht skaliert. Zu den Vertretern des zentralen Ansatzes gehören z.B. Condor-G und Nimrod-G aus Kapitel 5.2.2 und 5.2.3.

- **Dezentraler Scheduler**
  Beim dezentralen Ansatz ist der Scheduler genau für einen Job zuständig. Er hat die Aufgabe, für diesen einen Job die bestmögliche Ressource zu finden. Der Nachteil dabei ist, dass er nichts über schon laufende Jobs oder Jobs, welche zur gleichen Zeit von anderen Schedulern verteilt werden, weiß. Ein dezentraler Ansatz skaliert dagegen hervorragend und stellt keinen Single-Point-Of-Failure dar. AppLeS [Berman, 1997] ist ein Beispiel für einen dezentralen Ansatz und wird in Kapitel 5.2.1 beschrieben. AppLeS hat die Architektur des dezentralen Schedulers aus Abbildung 5. In [Hamscher, 2000] werden weitere dezentrale Ansätze untersucht.

- **Hybrid Scheduler**
  Ein Hybrid Scheduler kombiniert den zentralen mit dem dezentralen Ansatz. Es werden mehrere zentrale Scheduler parallel betrieben. Ein Scheduler wäre in diesem Fall für eine Teilmenge an Jobs zuständig. Damit kann das Problem der Skalierbarkeit bei zentralen Schedulern auf Kosten einer schlechteren Verteilung beim Scheduling erreicht werden. Dabei ist das Scheduling immer noch besser, als bei einem reinen dezentralen Scheduler, der nur für genau einen Job zuständig ist [Subramani, 2002]. Das ist ein möglicher Ansatz für eine Architektur. Weiterhin ist es denkbar, dass beispielsweise Grid-Scheduler in der Lage sind, untereinander Nachrichten oder Jobs auszutauschen.

2.3.7 Entscheidungsgrößen

Das Scheduling kann von verschieden Entscheidungsgrößen beeinflusst werden. Zu denen in Abbildung 1 dargestellten Größen gehören Informationen der Jobs und der Ressourcen. Je mehr Informationen berücksichtigt werden und je genauer die Informationen sind, desto effektiver kann ein Scheduler arbeiten.

- **Anwendungsspezifische Informationen**
  - benötigte Systemressourcen
  - Struktur der Jobs z.B. parallele Tasks
  - Performancemodell des Jobs
- **statische/dynamische Informationen von Ressourcen**
  - Konfiguration
  - Kostenmodelle
  - Auslastung
  - Features z.B. Reservierung

2.4 Framework

In dieser Diplomarbeit soll ein Framework für einen Grid-Scheduler entwickelt

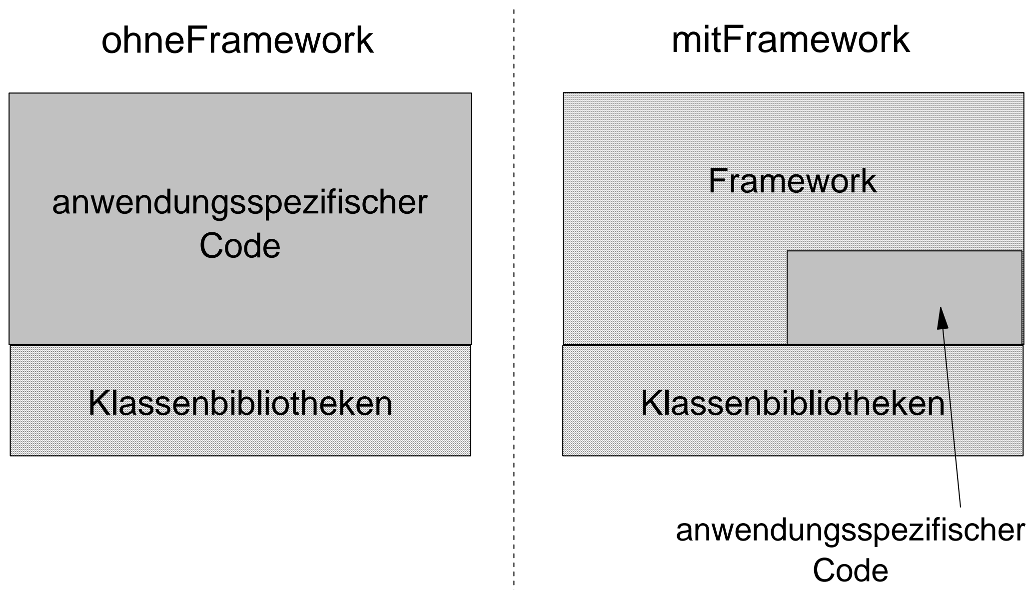


Abbildung 7: Verwendung eines Frameworks

Ein Framework ist in sich geschlossen und lauffähig und gibt die Architektur für eine Klasse von Anwendungen vor. Für eine konkrete Anwendung, in diesem Fall für den Grid-Scheduler, müssen bestimmte Schnittstellen implementiert werden, die für die Ausführung des Frameworks benötigt werden.

Der Unterschied zu einer Klassenbibliothek ist, dass ein Framework eine bestimmte Architektur und ein eigenes Design vorgibt. Ein Entwurfsmuster gibt zwar auch die Architektur vor, besitzt aber keine Implementierung, die für den konkreten Anwendungsfall erweitert wird [Wydler, 2001]. Diese Erweiterung ist in Abb. 7 veranschaulicht.

Zwei Möglichkeiten für Schnittstellen in Java sind mit folgenden Komponenten möglich:

- **Interface**: Schnittstellen, die von Klassen implementiert werden werden
- **Abstrakte Klasse**: Schnittstellen, die durch Ableitung von (abstrakten) Klassen und Überschreibung von Methoden implementiert werden

Der Vorteil eines Frameworks ist, dass es eine generische Lösung für bestimmte Problemfälle darstellt. Dadurch kann ein Framework wiederverwendet werden und besitzt somit die Vorteile, die durch die Wiederverwendung im Laufe der Zeit entstehen. Dazu gehören einfache Wartbarkeit, weniger Fehler und einen geringeren Implementierungsaufwand für neue Problemfälle.

3 Architektur im LabGrid

3.1 Das Globus Toolkit

Das Globus Projekt [Globus, 2003a] ist ein Forschungsprojekt unter der Führung des Argonne National Laboratory, der University of Southern California und der University of Chicago. Es hat das Ziel, Grid-Infrastrukturen und damit verbundene Sachverhalte sowie die Entwicklung von Grid-Anwendungen zu erforschen. Globus arbeitet dabei sehr eng mit Grid-Projekten aus Instustrie und Wissenschaft zusammen. Dabei geht es einerseits darum, standardisierte Grid-Protokolle für die Interoperabilität und andererseits APIs und SDKs für die Portabilität zu entwickeln. Globus stellt einfache Basisdienste zur Verfügung, die es erleichtern, Grids und gridbasierte Anwendungen zu erstellen. Die Kombination der Tools ergibt jedoch noch keine vertikal integrierte Lösung. Dafür fehlt beispielsweise der Scheduler. Zusammengefasst werden diese Tools als Globus Toolkit (GTK) [Foster, 1998][Ferreira, 2002][Globus, 2003a] bezeichnet. Viele Institutionen setzen für ihr eigenes Grid das Globus Toolkit ein.

In Globus Toolkit 3.0 wird die neue, auf WebServices basierende Architektur OGSA (siehe Kapitel 10.2), umgesetzt. In dieser Diplomarbeit wird das Globus Toolkit 2.0 verwendet, welches noch von der IBM verwendet wird und sich von der Architektur und dem API von der Version 3.0 gravierend unterscheidet.

3.1.1 Beispiele für den Einsatz

Zu den bekannten Beispielen für den Einsatz des GTKs gehören:

- Earth System Grid II
  ist ein Projekt des US Department of Energy zur Erforschung eines globalen Modells des Erdsystems [Earth System Grid, 2003].
- TeraGrid
  ist das amerikanische Wissenschaftsnetz, welches verschiedene Forschungseinrichtungen verbindet. Das Ziel ist es, das größte und die schnellste verteilte Infrastruktur für die wissenschaftliche Forschung zu erstellen [Catlett, 2002].
- DataGrid
  ist ein Projekt der Europäischen Union. Es bietet Rechenkapazität und die Forschung an verteilten Datenbanken an [Donno, 2002].

3.1.2 Hauptkomponenten

Das GTK ist ein Open Source Toolkit, das Referenzimplementierungen von Grid-Protokollen und APIs zur Entwicklung von Grids und dafür geeignete Anwendungen bereitstellt. Es besteht aus den folgenden vier Hauptkomponenten [Globus, 2003a] (siehe Abbildung 8):

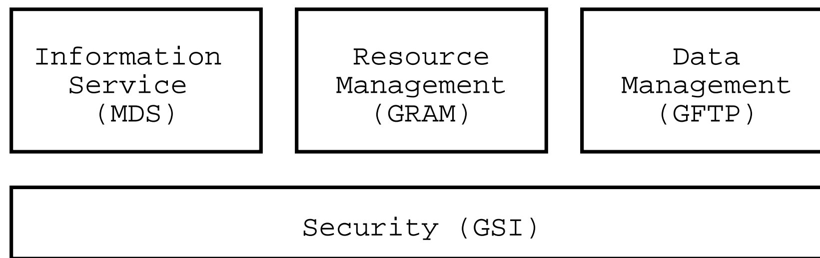


Abbildung 8: Die vier Globus Toolkit-Komponenten [Globus, 2003a]

- **Security** (Connectivity-Schicht):
  Die Grid Security Infrastructure (GSI) ist für die Authentifizierung und die Geheimhaltung zuständig. Sie dient als Grundlage für die drei folgenden Komponenten.

- **Information Service** (Resource-Schicht):
  Der Monitoring and Discovery Service (MDS) bietet Zugriff auf statische und dynamische Informationen der Ressourcen. Statische Informationen sind z.B. Prozessortyp und Betriebssystem. Dynamische Informationen sind Auslastung und freier Speicherplatz. Mit dem LDAP-Protokoll kann das MDS nach statischen und dynamischen Informationen abgefragt werden.
  Das MDS besteht aus drei Komponenten:

  - **Information Provider**
    Ein Information Provider fragt die Information von der lokalen Ressource, z.B. bei einem Cluster von einem Job-Scheduler, ab. Er wandelt die Informationen in das LDIF-Format (LDAP Data Interchange Format) um und übergibt sie dem GRIS.
  - **Grid Resource Information Service (GRIS)**
    Der GRIS ist für die lokalen Informationen verantwortlich, die es von den registrierten Information Providern erhält. Die Informationen sind dann auf dem aktuellen Stand und werden für eine bestimmte Zeit zwischengespeichert. Ist diese Zeitspanne überschritten, werden die Informationen aus dem GRIS entfernt. Der GRIS beantwortet nicht direkt Anfragen, sondern registriert sich bei dem GIIS, um die Informationen bereit zu stellen.
  - **Grid Index Information Service (GIIS)**
    Das GIIS bekommt seine Informationen von den GRIS oder von anderen GIIS. Damit kann eine Hierarchie ähnlich wie bei dem Domain Name System (DNS) zur Abfrage von Informationen aufgebaut werden.

- **Resource Management** (Resource-Schicht):
  Das Grid Resource Allocation Management (GRAM) ist für die Reservierung von Ressourcen, die Ausführung und die Verwaltung von Anwendungen auf entfernten Ressourcen zuständig.

- **Data Management** (Resource-Schicht):
  Das Grid File Transfer Protocol (GFTP) basiert auf dem FTP-Protokoll und

bietet einen sichere, zuverlässige und effiziente Datenübertragung zwischen einzelnen Knoten im Grid an.

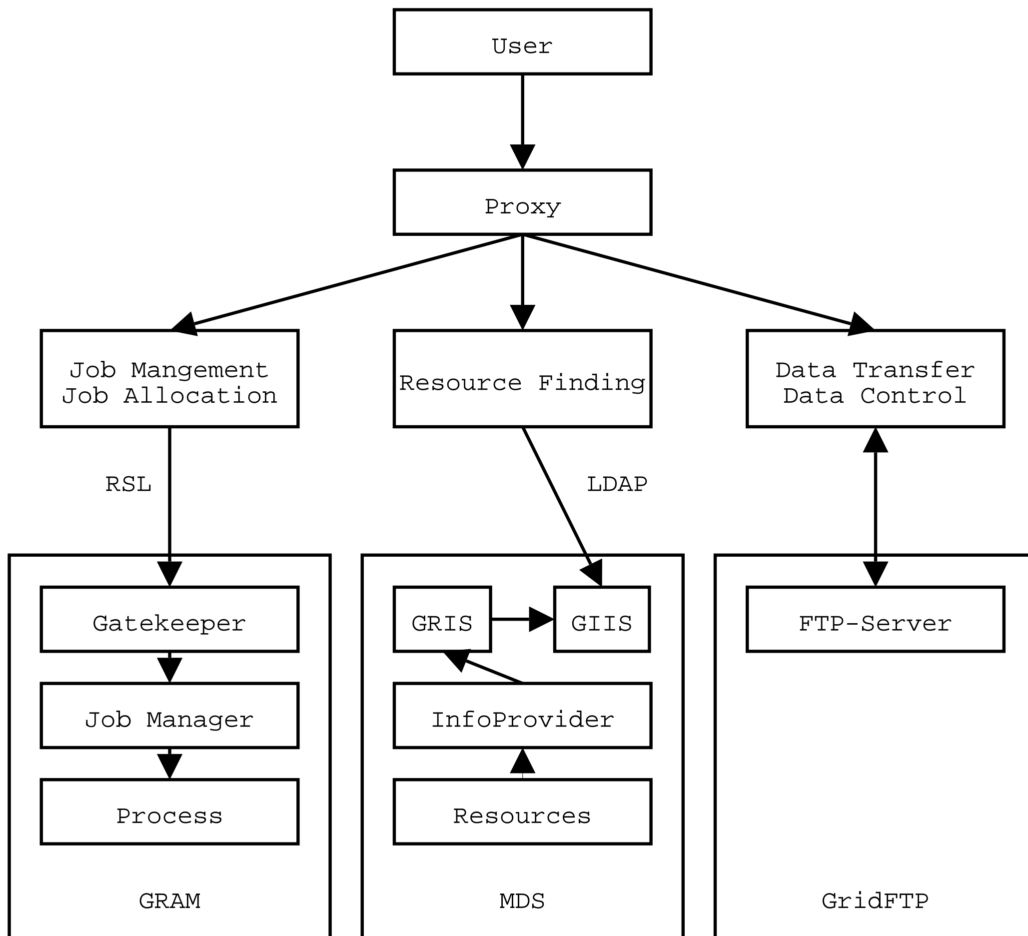


Abbildung 9: Zusammenspiel der Globus Toolkit-Komponenten

Die Komponenten des GTKs aus Abbildung 9 spielen folgendermaßen zusammen: Der Benutzer bzw. das Client-Programm hat sich dem Grid gegenüber authentifiziert und kann ein Proxy-Objekt erstellen. Dieses ermöglicht es ihm, mit dem Grid zu interagieren. Der Benutzer wählt mit der Hilfe des MDS anhand den statischen und dynamischen Informationen eine geeignete Ressource aus. Er übergibt dem Gatekeeper Dämon auf der Ressource eine RSL-Beschreibung des auszuführenden Jobs. Dieser erstellt für den Job einen Jobmanager, der die Aufgabe hat, den Job auszuführen und zu überwachen. Zur Ausführung übergibt der Jobmanager den Job an den lokalen Resource Manager, meistens ein Job-Scheduler (z.B. LoadLeveler[Kannan, 2001]), der den Job schließlich auf seinen Ressourcen ausführt. Am Ende der Ausführung übergibt der Jobmanager die Ergebnisse dem Benutzer und beendet sich.

3.2 Das Grid von IBM

Das Entwicklungslabor der IBM in Böblingen/Germany hat ein eigenes Grid das sog. „Boeblingen LabGrid“ auf der Basis des Globus Toolkits entwickelt [Gargya, 2002][Powers, 2003]. In den nachfolgenden Kapiteln wird die Motivation, der aktuelle Zustand und die geplanten Erweiterungen beschrieben.

Im LabGrid stellt ein Job für den Grid-Scheduler eine atomare Einheit dar und kann nicht weiter in einzelne Tasks aufgeteilt werden.

3.2.1 Motivation

Folgende Ziele verfolgt die IBM mit dem Aufbau eines eigenen Grids:

- Vorrangige Unterstützung der Hardware-Entwicklung bei der Entwicklung von Mikroprozessoren bei Chip-Design und Verifikationssimulation
- Schaffung der Möglichkeit, durch virtuelle Organisationen Ressourcen aus unterschiedlichen Abteilungen teilen zu können
- Aufbau von Erfahrungen im Bereich des Grid Computings
- Feststellung des heutigen Business Value
- Bereitstellung einer Testumgebung für Grid-Anwendungen

3.2.2 Architektur

Das LabGrid besteht aus einer 3-Schicht-Architektur (siehe Abbildung 10).

- **Frontend**
  Als Frontend stehen drei verschiedene Benutzerschnittstellen zur Verfügung. Es gibt die Möglichkeit, per Webbrowser, mittels einer Java GUI oder mit der Hilfe von Perl-Skripten Jobs im Grid auszuführen und zu verwalten. Der auszuführende Job wird mit der XML-Jobbeschreibungssprache (siehe Kapitel 3.5) beschrieben und an den Grid-Scheduler übergeben.
- **Middleend**
  Das Middleend besteht aus dem Grid-Scheduler. Er stellt dem Frontend die vier Servlets Logon, Submit, QueryStatus und Cancel bereit. Damit kann der Benutzer sich bei dem Grid-Scheduler anmelden, einen Job an das Grid übergeben, den Status eines Jobs abfragen und einen Job abbrechen. Die vier Servlets verwenden Java-Klassen, die die entsprechende Funktionalität zur Verfügung stellen. Die Java-Klassen benutzen als Verbindung zu den Tools des Globus Toolkits Java CoG 0.9.13 [JavaCoG, 2003]. Dies ist eine Java API, die z.B. für die Abfrage des zentralen Informationssystems benötigt wird, um einen Job einer Ressource zu übergeben oder einen Job abzubrechen. Zum Grid-Scheduler gehört eine Datenbank, die sämtliche Jobs und deren Status enthält, die über den Grid-Scheduler gestartet werden. Der Status der Jobs wird in regelmäßigen Abständen von dem Housekeeper (siehe Kapitel 3.3.1) aktualisiert. Dazu wird das zentrale Informationssystem verwendet.

- **Backend**
  Das Backend besteht aus den Tools des Globus Toolkits und drei Clustern. Auf den Knoten der Cluster ist das Betriebssystem AIX von IBM installiert. Jeder Cluster wird von dem lokalen Job-Scheduler LoadLeveler[Kannan, 2001] der IBM gesteuert. Der Zugriff auf Dateien und Jobs erfolgt über das gemeinsame verteilte Dateisystem AFS (Andrew File System).

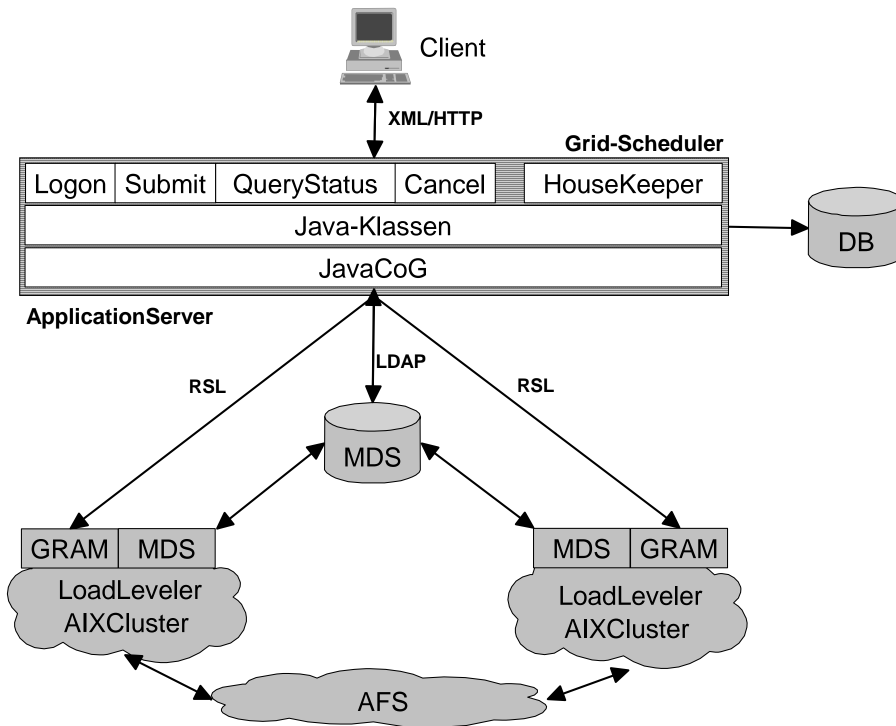


Abbildung 10: Architektur des LabGrids

3.2.3 Aktueller Stand

Dieses Kapitel gibt einen kurzen Überblick über die Geschwindigkeit des LabGrids und stellt die Callback-Lösung für die Erfassung der Jobstatus vor.

Die Abfrage der Ressourcen vom MDS dauerte zwischen 0,4 Sekunden (aktuelle Daten befinden sich im Cache) und 180 Sekunden (Timeout, falls der MDS nicht antwortet). Nach einem Timeout wird der zweite (Backup-)MDS abgefragt. Durchnittlich dauert eine MDS-Anfrage 21 Sekunden. Der Grund für die lange Abfragedauer ist nicht die Netzwerkübertragung (Gigabit-Ethernet) oder das MDS, sondern die Aufrufe der LoadLeveler-Kommandos in den InformationProvidern. Sie dienen zur Abfrage der Ressourcen und deren Attribute. Die Abfrage konnte durch die Reduzierung der LoadLeveler-Aufrufe, dem Weglassen von nicht benötigten Attributen und von Jobinformationen im MDS auf eine durchschnittliche Abfragedauer von 1,3 Sekunden reduziert werden. Die Dauer für die Ausführung der

LoadLeveler-Kommandos hängt sehr stark von der Auslastung des Masterknotens in dem jeweiligen Cluster ab.

Die Übergabe der Jobs von der GRAM-Komponente auf dem Masterknoten einer Ressource an den LoadLeveler dauert durchschnittlich 7 Sekunden.

Der Housekeeper erfasst alle 5 Minuten die Jobs, die zur Zeit ausgeführt werden. Die durchschnittliche Dauer für die Ausführung der meisten Jobs liegt bei ca. 30 Sekunden. Somit wird nur ein Teil der Jobs erfasst. Da, wie oben beschrieben, der neue Information Provider keine Jobinformationen mehr zurückliefert, findet der Housekeeper im MDS keine Jobinformationen mehr vor und ist damit nutzlos.

3.2.4 Beispiel für die Jobausführung

Der Endbenutzer übergibt entweder mit einem Perl-Skript, einer Java GUI oder mit dem Browser einen Job an den Grid-Scheduler. Der Job wird mittels XML-Jobbeschreibungssprache (siehe Kapitel 3.5) an den Grid-Scheduler übergeben. (Architektur und Funktionsweise des Grid-Schedulers ist in Kapitel 3.3.2 beschrieben.) Der Grid-Scheduler fragt das zentrale Informationssystem MDS nach verfügbaren Ressourcen ab und wählt anhand der Policy eine geeignete Ressource aus. Die Beschreibung des Jobs wird in die Jobbeschreibungssprache RSL des Globus Toolkits (siehe Kapitel 5.4.1) übersetzt und an die GRAM-Komponente der ausgewählten Ressource geschickt. In diesem Fall ist dies der Master-Knoten aus einem der drei AIX-Cluster. Der Job wird entgegengenommen und in die Jobbeschreibungssprache von LoadLeveler übersetzt. Danach wird die Jobbeschreibung an LoadLeveler übergeben, welcher den Job an einen seiner Knoten delegiert. Dieser führt den Job schließlich aus.

3.2.5 Zukunft

Momentan ist das Scheduling sehr stark auf bestimmte Simulationsjobs zugeschnitten und soll dahingehend erweitert werden, dass Anwendungen mit bestimmten Ressourcenanforderungen auf den entsprechenden Ressourcen ausgeführt werden können. Desweiteren ist geplant, das LabGrid mit anderen schon existierenden Grids zu verknüpfen.

3.3 Der Scheduler

Die folgenden zwei Kapitel geben einen Überblick über die Komponenten, die Funktionsweise und die Architektur des Schedulers.

3.3.1 Komponenten

Der Scheduler (siehe Abbildung 10 und 11) besteht aus den folgenden Komponenten:

- **Scheduler-DB**
  Die Scheduler-DB ist die Datenbank des Schedulers. In ihr werden alle Jobs, deren Attribute und deren aktueller Status gespeichert. Der Status wird vom Housekeeper aktualisiert.

- **Housekeeper**
  Der Housekeeper fragt in regelmäßigen Abständen vom MDS sämtliche Einträge laufender Jobs ab und aktuallisiert deren Status in der Scheduler-DB. Er ist ein eigenständiges Programm, das in bestimmten Intervallen, hier alle 5 Minuten, aufgerufen wird.

  Sobald ein Job beendet ist, ist er nicht mehr im MDS vorhanden. Das führt zu dem Problem, dass Jobs, deren Laufzeit kürzer als das vorgegebene Intervall ist, nicht mehr vom Housekeeper erfasst werden. Folglich ist es nicht mehr erkennbar, ob ein Job erfolgreich beendet wurde, fehlgeschlagen ist oder abgebrochen wurde. Das Problem lässt sich erst mit dem Monitor-Programm aus Kapitel 8.1 lösen. Über die Schnittstelle von Globus kann der Status eines Jobs nicht erfasst werden.

- **Job-Objekt**
  Das Job-Objekt ist ein Datencontainer für Attribute und Parameter des auszuführenden Jobs. Diese stammen entweder aus einer Anfrage an die Servlets oder aus den Einträgen zu einem Job in der Scheduler-DB. Ein Job-Objekt wird für den Zugriff auf Daten einer Anfrage, die Erstellung einer RSL-Jobbeschreibung und die Abfrage oder den Eintrag eines Jobs in die Scheduler-DB benötigt.

- **XML-Parser**
  Der XML-Parser erhält von den Servlets die Anfrage in der XML-Jobbeschreibungssprache und extrahiert den Inhalt. Der XML-Parser kann nach den Attributen der Anfrage abgefragt werden.

- **Logon-Servlet**
  Das Logon-Servlet nimmt eine Anfrage für ein Logon in der XML-Jobbeschreibungssprache entgegen. Die Anfrage wird dem XML-Parser übergeben. Dieser extrahiert den Benutzernamen und das Passwort. Das Logon-Servlet vergleicht die Daten mit den Einträgen in der Scheduler-DB. Falls der Benutzer authorisiert ist, erstellt es eine gültige Session, um die anderen drei Servlets benutzen zu können.

- **Submit-Servlet**
  Das Submit-Servlet nimmt eine Anfrage zur Ausführung eines Jobs in der XML-Jobbeschreibungssprache entgegen. Die Anfrage wird dem XML-Parser übergeben. Dieser extrahiert die Parameter für die Ausführung eines Jobs. Aus diesen Parametern wird ein Job-Objekt erstellt. Das Job-Objekt wird die ausgewählte Ressource übergeben, und das Job-Objekt generiert die Jobbeschreibung mit der Jobbeschreibungssprache RSL und übergibt diese an die Ressource. Als Antwort der Übergabe erhält das Job-Objekt die Globus-URL, die den Job innerhalb des Grids identifiziert. Der Status und die Globus-URL werden in der Scheduler-DB eingetragen. Das Resultat dieser Aktion wird mit XML beschrieben und an den Benutzer zurückgeliefert.

- **Query-Status-Servlet**
  Das Query-Status-Servlet nimmt eine Anfrage zum aktuellen Status mehrerer Jobs in der XML-Jobbeschreibungssprache entgegen. Die Anfrage wird

dem XML-Parser übergeben. Dieser extrahiert die Liste der Jobs, von welchen der Status zurückgegeben werden soll. Dazu erstellt das Servlet für jeden Job ein Job-Objekt aus den Jobdaten der Scheduler-DB und liefert den Status der Jobs an den Benutzer zurück. Das Resultat dieser Aktion wird mit XML beschrieben und an den Benutzer zurückgeliefert.

- **Cancel-Servlet**
  Das Cancel-Servlet nimmt eine Anfrage zum Abbruch mehrerer Jobs, in der XML-Jobbeschreibungssprache entgegen. Die Anfrage wird dem XML-Parser übergeben und dieser extrahiert die Liste der Jobs, welche abgebrochen werden sollen. Für jeden Job wird ein Job-Objekt mit den Jobdaten aus der Scheduler-DB erstellt, in die RSL übersetzt und an die GRAM-Komponente übergeben, die den Job dann abbricht. Das Resultat dieser Aktion wird mit XML beschrieben und an den Benutzer zurückgeliefert.

- **Policy**
  Die Komponente Policy hat die Aufgabe, die Ressource auf der der Job ausgeführt werden soll, auszuwählen. Sie fragt mit das MDS nach sämtlichen Ressourcen ab. Aus der Menge der Ressourcen wählt die Scheduling-Policy diejenige Ressource aus, welche prozentual am wenigsten ausgelastet ist. Genauer wird der Algorithmus in Kapitel 3.4 erläutert.

3.3.2 Architektur

Die Architektur des Grid-Schedulers der IBM ist in Abbildung 11 dargestellt. Aufgrund der Übersichtlichkeit sind die Logon-, QueryStatus- und Cancel-Servlets (vergleiche Abbildung 10) nicht dargestellt, da diese für das Verständnis der Architektur nicht beitragen. Die Kanten zwischen den einzelnen Komponenten stellen den Datenfluss dar.

Der Klient meldet sich mit dem Logon-Servlet aus Abb. 10 beim Grid-Scheduler an. Dieser überprüft anhand der Benutzerdaten in der lokalen Datenbank, ob der Benutzer authorisiert ist, einen Job auszuführen. Ist dies der Fall, dann kann der Klient mit dem Submit-Servlet einen Job in der XML-Jobbeschreibungssprache aus Kapitel 3.5 an den Grid-Scheduler übergeben. Die Jobbeschreibung wird von dem XML-Parser analysiert, und als Resultat wird ein Job-Objekt erstellt. Die Policy wählt die Ressource mit der Hilfe des MDS aus und übergibt sie dem Submit-Servlet. Der genaue Algorithmus der Scheduling-Policy wird in Kapitel 3.4 genauer beschrieben. Das Submit-Servlet übergibt nun die ausgewählte Ressource an das Job-Objekt. Dieses schreibt seine Attribute und seinen Status in die Scheduler-DB und erzeugt anhand seiner Attribute eine neue Jobbeschreibung mit der Jobbeschreibungssprache RSL des Globus Toolkits (siehe Kapitel 5.4.1). Das Job-Objekt übergibt die Jobbeschreibung an die GRAM-Komponente der ausgewählten Ressource. Von nun an ist die Ressource für die Jobausführung verantwortlich. Das Submit-Servlet erzeugt als Resultat der Jobübermittlung eine Antwort als XML-Dokument und übergibt sie dem Klienten.

Die Architektur des ursprünglichen Schedulers hat folgende **Nachteile**:

- Es fehlt eine Matchmaking-Komponente.

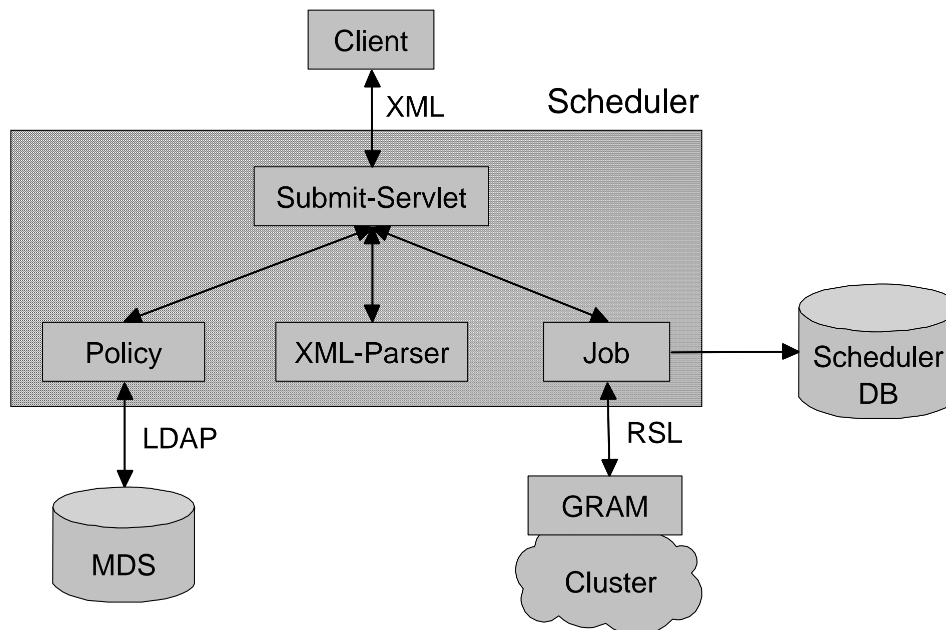


Abbildung 11: Der ursprüngliche Scheduler

- Die Policies sind nicht austauschbar.
- Kein anderes Informationssystem einer Grid-Middleware kann verwendet werden.

3.4 Scheduling-Policy

Der Scheduler der IBM arbeitet mit der nachfolgenden Policy, die anhand von drei Attributen einer Ressource entscheidet, auf welcher Ressource ein Job ausgeführt wird. Die folgenden Attribute stammen von dem lokalen Scheduler LoadLeveler, der auf jeder Ressource im Grid installiert ist. Ein Job wird über die GRAM-Komponente an den LoadLeveler zur Ausführung übergeben. Dabei wird der Job unter einem festgelegten User von AIX an eine bestimmte Klasse übergeben.

- **Klasse**
  In der Konfigurationsdatei von LoadLeveler können mehrere Klassen definiert werden. Einer Klasse sind bestimmte Knoten innerhalb eines Clusters zugeordnet. Auf einem dieser Knoten wird ein Job einer Klasse von LoadLeveler ausgeführt. Die Klasse wird in der XML-Jobbeschreibungssprache mit dem „QUEUE“-Tag angegeben und ist für den beschriebenen Fall in Kapitel 3.5 die Klasse „gridsim“.

- **Slot**
  In der Konfigurationsdatei des LoadLevelers kann jedem Knoten innerhalb

einer Ressource bzw. eines Clusters eine bestimmte Anzahl von Slots pro Klasse zugeordnet werden. Da alle Knoten in den Clustern Mehrprozessormaschinen sind, wird folgendermaßen vorgegangen:

Hat ein Rechner n Prozessoren wird einer davon für das Betriebssystem AIX reserviert. Die restlichen $n - 1$ Prozessoren stehen für die Ausführung von Jobs zur Verfügung. Dabei kann die Anzahl der Slots je Knoten zwischen $0 \leq Anzahl\ der\ Slots \leq (n - 1)$ variieren. Wenn die Anzahl der Slots eines Knotens für eine bestimmte Klasse 0 ist, darf LoadLeveler auf diesem Knoten keinen Job dieser Klasse ausführen. Ein Job benötigt für die Ausführung einen Slot und somit einen Prozessor. Auf einem Prozessor wird maximal ein Job ausgeführt.

- **freeSlots**
  gibt an, wieviele freie Slots eine Ressource für eine bestimmte Klasse hat.
- **maxSlots**
  ist eine Konstante und gibt an, wieviele Slots eine Ressource für eine konkrete Klasse besitzt. Der Wert ist die Summe aller Slots aller Knoten einer Klasse innerhalb eines Clusters.
- **jobWait**
  gibt die Anzahl der wartenden Jobs einer Klasse in der Warteschlange einer Ressource an.

Bei der Erstellung der folgende Policy wird angenommen, dass freeSlots den Wert der zur Verfügung stehenden Kapazität und maxSlots die maximale Kapazität einer Ressource angibt. Weil für die Bestimmung der freien und maximalen Kapazität weitere Attribute benötigt werden, arbeitet die Policy nicht wie gewünscht und sorgt für eine ungleichmäßige Verteilung. Die Probleme werden in Kapitel 7.1.1 genauer untersucht. Auf die Attribute und deren Bedeutung wird bei der Neuentwicklung der Policy in Kapitel 7.1.2 genauer eingegangen.

Die Policy besteht aus zwei Phasen:

1. Phase:
   Die ersten Phase wird ausgeführt, falls eine Ressource freie Kapazität für die Ausführung eines Jobs hat, also $freeSlots > 0$ ist. Es wird diejenige Ressource ausgewählt, welche im Verhältnis zu den übrigen Ressourcen am wenigsten ausgelastet ist. Die Auslastung wird folgendermaßen berechnet:

   $$Auslastung\ der\ Ressource = 1 - \frac{freeSlots}{maxSlots}$$

   Falls mehrere Ressourcen die gleiche Auslastung haben, wird die Ressource mit der größeren Anzahl an $maxslots$ verwendet, weil dadurch die Verteilung der Last auf den Ressourcen gleichmäßiger ist.

2. Phase:
   Die zweite Phase ist nur von Bedeutung, falls keine Ressource freie Kapazität hat. In diesem Fall wählt die Policy diejenige Ressource aus, deren Warteschlange kleiner ist als die maximal zulässige Warteschlangenlänge. Die max. Warteschlangenlänge berechnet sich folgendermaßen:

   $$max.\ Warteschlangenlänge = maxSlots * slotFactor$$
   $$\textit{(slotFactor ist eine beliebig gewählte Konstante)}$$

   Aus dieser Menge an Ressourcen wird diejenige Ressource ausgewählt, die im Verhältnis von ihrer Warteschlangenlänge zu ihrer max. Warteschlangenlänge, die geringste Auslastung hat:

   $$\begin{aligned} Auslastung\ der\ Warteschlange &= \frac{jobwait}{max.\ Warteschlangenlänge} \\ &= \frac{jobwait}{maxslots * slotFactor} \end{aligned}$$

   Für den Fall, dass alle Ressourcen ihre maximale Warteschlangenlänge erreicht haben, kann der Job an keine Ressource übergeben werden, und der Scheduler antwortet dem Klienten mit der Fehlermeldung, dass keine Ressource frei ist.

Der genaue Algorithmus ist im folgenden aufgeführt und liefert für eine Menge an Ressourcen die bestmögliche Ressource zurück:

```
Resource getQualifiedResource(Vector resources) {
  Resource resource, bestHost, bestHostOld = null;
  float bestHostQuot = 0;
  float bestHostQuotOld = 2;
  int bestHostMaxSlots = 0;
  int slotFactor = 5;
  int freeslots, maxslots, jobwait;
  float tmpQuot;

  Iterator hostIt = resources.iterator();
  while (hostIt.hasNext()) {
    resource = (Resource)hostIt.next();

    freeslots = resource.getAttributeValue("freeslots");
    maxslots  = resource.getAttributeValue("maxslots");
    jobwait   = resource.getAttributeValue("jobwait");

    if (freeslots > 0) {
```

– Start 1. Phase ——————————————-

```
   tmpQuot = (float)freeslots/maxslots;

   if ((jobwait > 2*maxslots) && (freeslots > (maxslots/10+2)) ) {
    ...error: something wrong with host
   } else {
    // get the resource with the most free slots in percent
    if ((tmpQuot > bestHostQuot) ||
     ((tmpQuot == bestHostQuot)
     && (maxslots > bestHostMaxSlots))) {
      bestHostQuot = tmpQuot;
      bestHostMaxSlots = maxslots;
      bestHost = resource;
    }
   }
```

– Ende 1. Phase ————————————-

```
  } else {
```

– Start 2. Phase ————————————-

```
   if ((bestHost == null) && (jobwait < maxslots * slotFactor)) {
    tmpQuot = (float)jobwait/(maxslots*slotFactor);
    if (tmpQuot < bestHostQuotOld) {
     bestHostQuotOld = tmpQuot;
     bestHostOld = resource;
    }
   }
```

– Ende 2. Phase ————————————-

```
  }
 } //end while

 if (bestHost == null) {
  if (bestHostOld != null) {
    bestHost = bestHostOld;
  } else {
    ...error no free host found
  }
 }
 return bestHost;
}
```

(Die Variable *bestHostQuotOld* muss mit einem Wert größer oder gleich Eins initialisert werden, damit die If-Bedingung *if (tmpQuot < bestHostQuotOld)* bei dem ersten Mal erfüllt werden kann. Für *tmpQuot* gilt: $0 \leq tmpQuot < 1$)

Die Policy des ursprünglichen Schedulers hat den **Nachteil**, dass die Ressourcen ungeleichmäßig ausgelastet werde (siehe Kapitel 7.1.1).

3.5 XML-Jobbeschreibungssprache

Mit der XML-Jobbeschreibungssprache wird ein Job an den Grid-Scheduler übergeben. Das folgende Beispiel zeigt eine Jobbeschreibung zwischen dem Klienten und dem Scheduler:

```
<IDEGRID>
  <REQUEST>
    <SUBMIT>
      <EXECUTABLE>/bin/echo</EXECUTABLE>
      <ARGUMENT>-n</ARGUMENT>
      <JOBTYPE>single</JOBTYPE>
      <QUEUE>gridsim</QUEUE>
      <STDOUT>/home/tbeicht/file.out</STDOUT>
      <STDIN>/home/tbeicht/file.in</STDIN>
      <STDERR>/home/tbeicht/file.err</STDERR>
      <ENVIRONMENT></ENVIRONMENT>
      <DIRECTORY></DIRECTOR>
      <PROJECT></PROJECT>
      <DRYRUN></DRYRUN>
      <COUNT></COUNT>
      <MAXCPUTIME></MAXCPUTIME>
      <MAXTIME></MAXTIME>
      <MAXWALLTIME></MAXWALLTIME>
      <MINMEMORY></MINMEMORY>
    </SUBMIT>
  </REQUEST>
</IDEGRID>
```

Die Bedeutung der Elemente sowie deren Wert zwischen den „Submit“-Tags kann eins zu eins auf die GRAM RSL Parameter [Globus, 2003a][Globus, 2003b] abgebildet werden. Beispielsweise gibt das „EXECUTABLE“-Tag den Job und das „ARGUMENT“-Tag die Argumente an, mit denen der Job aufgerufen werden soll.

Die Jobbeschreibungssprache des ursprünglichen Schedulers hat folgende **Nachteile**:

- Es können keine Jobanforderungen für das Matchmaking übergeben werden.
- Es kann keine Policy angegeben werden.

3.6 Nachteile des Schedulers

Die vorherigen Kapitel haben gezeigt, dass der Scheduler im LabGrid folgende Nachteile hat:

- Es fehlt eine Matchmaking-Komponente.
- Die Policies sind nicht austauschbar.

- Es kann kein anderes Informationssystem von dem Scheduler verwendet werden.
- Die Ressourcen werden ungeleichmäßig ausgelastet.
- Es können keine Jobanforderungen für das Matchmaking mit der Jobbeschreibung übergeben werden.
- Es kann keine Policy in der Jobbeschreibung angegeben werden.

4 Anforderungen an das Framework

Dieses Kapitel stellt die Anforderungen zur Lösung der Probleme des Schedulers im LabGrid dar. Dabei handelt sich einerseits um Anforderungen, die IBM zwingend benötigt, als auch Anforderungen die wünschenswert sind, und je nach Aufwand und verfügbarer Zeit umgesetzt werden können. Am Ende des Kapitels werden die Vorteile durch die Erfüllung der Anforderungen verdeutlicht.

4.1 Anforderungen der IBM

1. Der ursprüngliche Scheduler kann keine potentiellen Ressourcen durch die Angabe von Jobanforderungen auswählen und unterscheidet Ressourcen nur anhand ihrer Auslastung und nicht z.B. auch an ihrer Funktionalität oder Leistungsfähigkeit. Dadurch können keine heterogene Ressourcen im Grid bedient werden. Das neue Framework soll die Auswahl der potentiellen Ressourcen mit der Hilfe des sogenannten Matchmakings treffen.

   XML-Request wird das Dokument genannt, welches mit der XML-Jobbeschreibungssprache erstellt wurde. Sie bietet in der ursprünglichen Version keine Möglichkeit an, um Jobanforderung zu spezifizieren, unter welchen Bedingungen ein Job ausgeführt werden soll. Der Matchmaker benötigt für seine Funktion Jobanforderungen, die in dem XML-Request angegeben werden müssen. Daher muss die Jobbeschreibungssprache erweitert werden. Im ersten Schritt sollen sich diese Anforderungen auf die Systeminformationen, wie Prozessortyp, Name des Betriebssystems und der Betriebssystemversion beschränken. Damit in der Zukunft weitere Anforderungen der Jobs berücksichtigt werden können, soll es eine einfache Möglichkeit geben, das Matchmaking um neue Anforderungen zu erweitern.

2. Der ursprüngliche Scheduler hat den Nachteil, dass die Policy eng mit dem Scheduler gekoppelt ist. Neben dem Matchmaking ist die Schnittstelle und die Möglichkeit der Auswahl einer Scheduling-Policy die zweite große Anforderung an das Framework.

   Der Benutzer soll die Möglichkeit haben, die zu verwendende Policy manuell auszuwählen. Mit der ursprüngliche Jobbeschreibungssprache kann die Policy in dem XML-Request nicht explizit angegeben werden. Darum muss die Jobbeschreibungssprache erweitert werden. Falls der XML-Request jedoch keine Policy vorschreibt, wird auf eine standardmäßig vor eingestellte Scheduling-Policy zurückgegriffen. Es muss dafür eine generische Schnittstelle für die Anbindung von Scheduling-Algorithmen geben. Eine neue Scheduling-Policy soll sich einfach implementieren und einbinden lassen.

   In dem zu implementierenden Prototyp soll der vorhandene Scheduling-Algorithmus verwendet werden. Dieser lastet die Ressourcen jedoch sehr ungleichmäßig aus. Daher soll ein neuer Algorithmus ausgewählt und implementiert werden. Der neue Algorithmus wird eine optimierte Version des alten Algorithmus darstellen.

3. Die jetzige XML-Jobbeschreibungssprache muss für die Beschreibung der

Jobanforderungen und für die Auswahl der Scheduling-Policy erweitert werden.

4. Beim aktuellen Scheduler ist es sehr aufwendig ein anderes Informationssystem einer anderen Grid-Middleware abzufragen. Aus diesem Grund wird eine Schnittstelle zur Anbindung von unterschiedlichen Informationssystemen benötigt.

Falls die Informationen im MDS für das Matchmaking und für das Scheduling nicht ausreichen, müssen neue Information Provider implementiert und beim MDS registriert werden.

Die Informationen über die aktuell ausgeführten Jobs sollen, wie beim alten Scheduler in der lokalen DB2-Datenbank des Schedulers verwaltet werden.

Der Scheduler der IBM wird vom Benutzer mit der Hilfe der vier Servlets gesteuert. Diese sind „Logon“ für die Anmeldung, „Submit“, um Jobs auszuführen, „QueryStatus“, um deren aktuellen Zustand abzufragen, und „Cancel“, um diese abzubrechen. Diese Servlets sollen erhalten bleiben, damit die vorhandenen Frontends weiterhin verwendet werden können. Da das Framework das Matchmaking anhand von Job-Anforderungen durchführt, müssen die Frontends diese Möglichkeit im XML-Request berücksichtigen. Das gleiche gilt auch für die Auswahl der Scheduling-Policy, wobei diese Angabe optional ist.

Das neue Framework und der Prototyp sollen wie der vorhandene Scheduler mit JDK 1.3.1, Java CoG 0.9.13 implementiert und auf dem Applikationsserver Tomcat 4.0.3 lauffähig sein. Die Ressourcen verwenden das Globus Toolkit 2.0 und 2.2.

4.2 Nutzen der Lösung

Das Framework erlaubt Anwendungen mit unterschiedlichen Ressourcenanforderungen auszuführen. Damit ist es möglich, die Ressourcen der verschiedenen Abteilungen für projektspezifische Aufgaben, wie in diesem Fall z.B. für Simulationsjobs, zu nutzen. Das Framework bietet die Möglichkeit, unterschiedliche Scheduling-Policies für unterschiedliche Anwendungen zu verwenden. Durch eine flexible Schnittstelle, die den Austausch der Scheduling-Policies erlaubt, wird die Entwicklung von neuen Schedulingalgorithmen unterstützt, und die Algorithmen können durch den Austausch auf Effektivität und Effizienz untersucht werden. Desweiteren bietet das Framework eine Schnittstelle für Informationsquellen anderer Resource Management Systeme und eine Schnittstelle für die Steuerung von Jobs. Damit kann das Framework auch mit anderen Grid-Middleware-Systemen arbeiten.

5 Verwandte Arbeiten

5.1 Matchmaking

5.1.1 Condor ClassAds

In [Raman 1998] wird ein sehr flexibles Matchmaking-Framework beschrieben, welches von den Schedulern Condor und Condor-G verwendet wird. Das Matchmaking funktioniert nach folgendem Ansatz:

Sowohl Jobs als auch Ressourcen können ihre Anforderungen und Bedingungen in einer Art Kleinanzeige einer Zeitung (ClassAd) stellen. Der Matchmaker sucht anhand von Anforderungen und Bedingungen die passenden Kombinationen von ClassAds heraus und ordnet diese nach einem Wert, der den Grad der Übereinstimmung von Job und Ressource angibt. Wie und mit welcher Gewichtung der Wert berechnet wird, kann in jedem ClassAd als eine Formel angegeben werden. Dazu können auch die Attribute des anderen ClassAds ausgewertet werden. Die Kombination an ClassAds, die am besten zueinander passt, wird ausgewählt, und die ClassAds werden dem jeweiligen Gegenstück mit dem „Matchmaking-Protokoll“ übermittelt. Die dazugehörige Ressource oder der Job kann nun selbst entscheiden, ob die Empfehlung des Matchmakers angenommen wird oder nicht. Der Matchmaker hat seine Aufgabe erledigt. Von diesem getrennt, kann der Job nun das „Claiming-Protokoll“ benutzen. Es beschreibt, wie eine Ressource genutzt werden kann.

5.1.2 Matchmaking in Globus

In Globus können mit der Jobbeschreibungssprache RSL aus Kapitel 5.4.1 die Ressourcenanforderungen eines Jobs beschrieben werden. Das Matchmaking wird von dem „Resource Co-Allocator“ durchgeführt. Es gibt jedoch keine ähnliche Möglichkeit von Seiten der Ressourcen, Bedingungen an die Jobs zu stellen [Raman 1998].

5.1.3 Matchmaking in Legion

Legion verwendet einen objektorientierten Ansatz. Alle Komponenten, Jobs und Ressourcen sind in Legion Objekte und können daher leicht und flexibel durch Vererbung erweitert und spezialisiert werden. Das Matchmaking wird als „Object Placement Problem“ beschrieben und kann mit den ClassAds von Condor verglichen werden. Ressourcen können auch Bedingungen an die Jobs stellen. [Raman 1998][Karpovich, 1996]

5.1.4 Matchmaking in herkömmlichen Schedulern

Bei den lokalen Schedulern LoadLeveler [Kannan, 2001], PBS [Ferreira, 2002] und LSF [Ferreira, 2002] können Ressourcen explizit durch die Verwendung der jeweiligen Jobbeschreibungssprache oder implizit durch die Angabe einer Klasse von Ressourcen ausgewählt werden. Auf dem Master-Knoten eines Clusters können

Klassen definiert werden, zu der Knoten mit einer bestimmten Konfiguration innerhalb eines Clusters gehören. Durch die Angabe einer Klasse wird somit das Matchmaking schon durchgeführt. Der Job wird nun nur noch an einen der freien Knoten innerhalb der Klasse übergeben und ausgeführt.

5.2 Untersuchung existierender Scheduler

Dieses Kapitel stellt exemplarisch drei bekannte und verbreitete Grid-Scheduler vor.

5.2.1 AppLeS

Das AppLeS-Projekt [Foster, 1999][Berman, 1997] verwendet für das Scheduling einen dezentralen Agentenansatz. Dabei hat jede Grid-Applikation einen eigenen Agenten, der die Aufgabe hat, den optimalen Ausführungsplan für seine Applikation auf den Ressourcen für die geplante Ausführungszeit zu erstellen. Der AppLeS-Scheduler verfolgt den Ansatz, dass jede Komponente im Grid in irgendeiner Form Einfluss auf die Anwendung hat. Die Informationen für die Erstellung des Ausführungsplans werden aus dem Performance-Modell der Anwendung, dynamischen Informationen und anwendungsspezifischen Informationen gewonnen. Das Performance-Modell muss vom Benutzer erstellt werden, der die Komponenten der Anwendung und deren Einfluss auf die Performance beschreibt. Der Network Weather Service steht allen Agenten zur Verfügung und liefert dynamisch Vorhersagen über die Auslastung des Netzwerks für die geplante Ausführungszeit der Anwendung (siehe Abbildung 12).

Um den besten Ausführungsplan auszuwählen, muss der Agent:

1. Die Mengen der nutzbareren Ressourcen auswählen (Matchmaking) (**Resource Selector**)
2. Einen Ausführungsplan für jede Menge erstellen (**Schedule Planner**)
3. Für jeden Ausführungsplan und der dazugehörigen Menge an Ressourcen eine Vorhersage über die Ausführungszeit treffen (**Performance Estimator**)
4. Die beste Kombination aus Ausführungsplan und Ressourcenmenge auswählen (**Coordinator**)
5. Die beste Kombination auf den Ressourcen ausführen (**Actuator**)

Diese Komponenten verwenden Informationen aus dem **Information Pool**, der aus folgenden Komponenten besteht:

- Der **Network Weather Service** sagt dynamische Informationen über den Systemstatus und die Auslastung der Ressourcen vorher.
- Das **User Interface** liefert Informationen über die Charakteristik und die Struktur der Anwendung, sowie die Bedingungen unter denen die Anwendung ausgeführt werden soll.

- Die Komponente **Models** enthält Modelle verschiedener Klassen von Standardanwendungen, um damit das Verhalten von Anwendungen abschätzen zu können.

Der AppLeS-Scheduler wird für parallele Anwendungen eingesetzt. Der Nachteil des AppLeS-Schedulers ist, dass sein Ausführungsplan nur für die eine Anwendung optimiert ist und nicht die ökonomische Nutzung des Grids berücksichtigt.

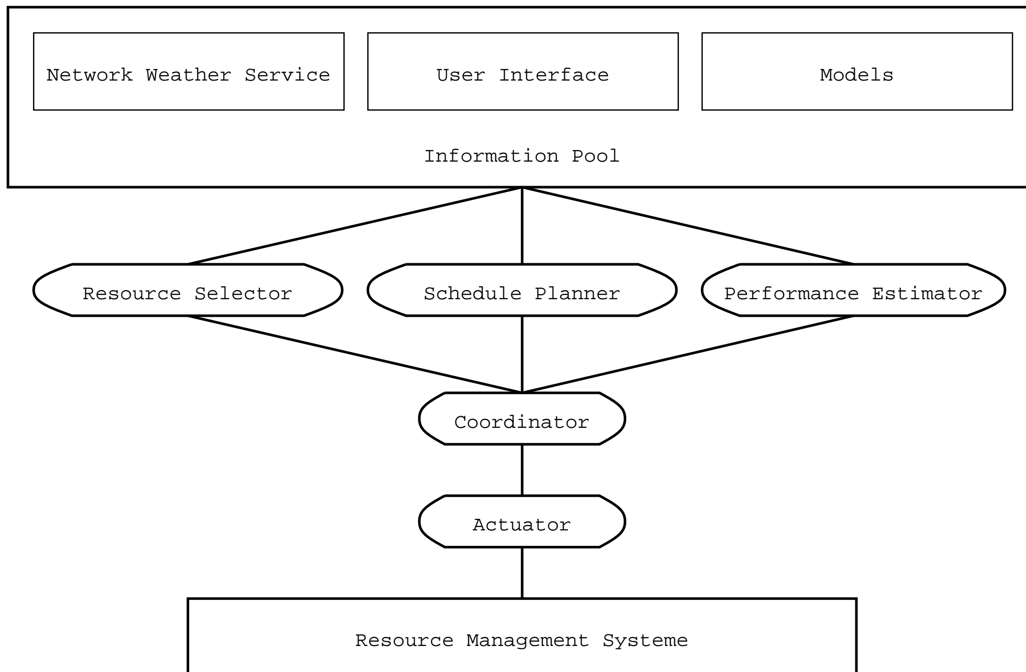


Abbildung 12: AppLeS-Architektur

Die genaue Arbeitsweise von AppLeS ist in [Berman, 1997] beschrieben.

5.2.2 Nimrod-G

Der Nimrod-G-Scheduler [Buyya, 2000a] verwendet im Gegensatz zum AppLeS-Scheduler einen zentralisierten Ansatz und berücksichtigt auch die ökonomische Nutzung des gesamten Grids. Er wird hauptsächlich für wissenschaftliche Anwendungen eingesetzt. Er erlaubt, parameterisierbare Anwendungen mit der Angabe der Parameterbereiche selbstständig parallel und verteilt auszuführen und die Ergebnisse zu sammeln.

Nimrod-G besteht aus folgenden Komponenten (siehe Abbildung 13):

- **Task-Farming Engine**
  Die persistente Task-Farming Engine koordiniert das Ressourcen-Trading, Scheduling, Data-Staging, die Ausführung der Jobs und das Sammeln der Ergebnisse.

- **Grid Explorer**
  Der Grid Explorer hat die Aufgabe, Ressourcen zu erfassen.

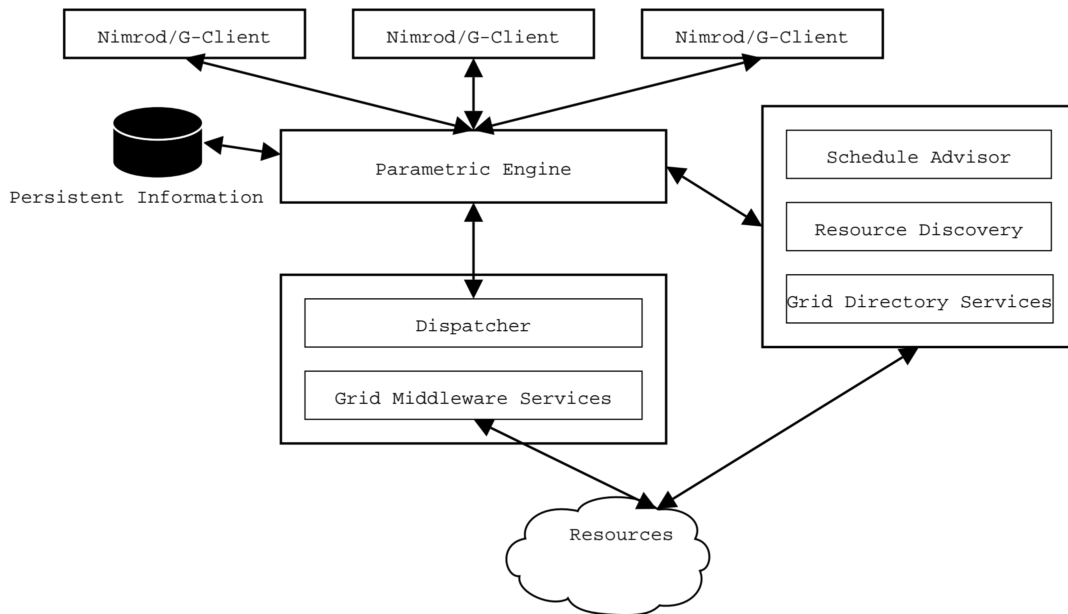


Abbildung 13: Nimrod/G-Architektur

- **Trading Manager**
  Der Trading Manager muss den Preis für die Nutzung der einzelnen Ressourcen festlegen.
- **Schedule Advisor**
  Der Schedule Advisor weist den Jobs Ressourcen zu.
- **Nimrod-G Agent Dispatcher**
  Er hat die Aufgabe, Agenten auf den Ressourcen einzusetzen.
- **Agent**
  Dieser verwaltet die Ausführung eines Jobs auf einer Ressource.

Der Vorteil von Nimrod-G ist, dass eine Anwendung mehrfach parallel gestartet wird, und jede der Anwendungen arbeitet auf einem Teil der Daten oder mit bestimmten Parametern, ohne dass die Anwendung speziell dafür entwickelt wurde. Nimrod-G wurde speziell für wissenschaftliche Anwendungen entwickelt. Für jeden Job muss in einer Nimrod-G-spezifischen Sprache eine Parametrisierung vorgenommen werden.

5.2.3 Condor-G

Der Condor-G Grid-Scheduler [Frey, 2001] (siehe Abbildung 14) ist auf der Basis des Globus-Toolkits und den Erfahrungen des lokalen Schedulers Condor an der University of Wisconsin entstanden. Condor-G verwendet für den Zugriff auf Ressourcen die Protokolle von Globus und die Methoden der lokalen Condor Scheduler, um dem Benutzer die Ressourcen aus den verschiedensten administrativen Bereichen anzubieten.

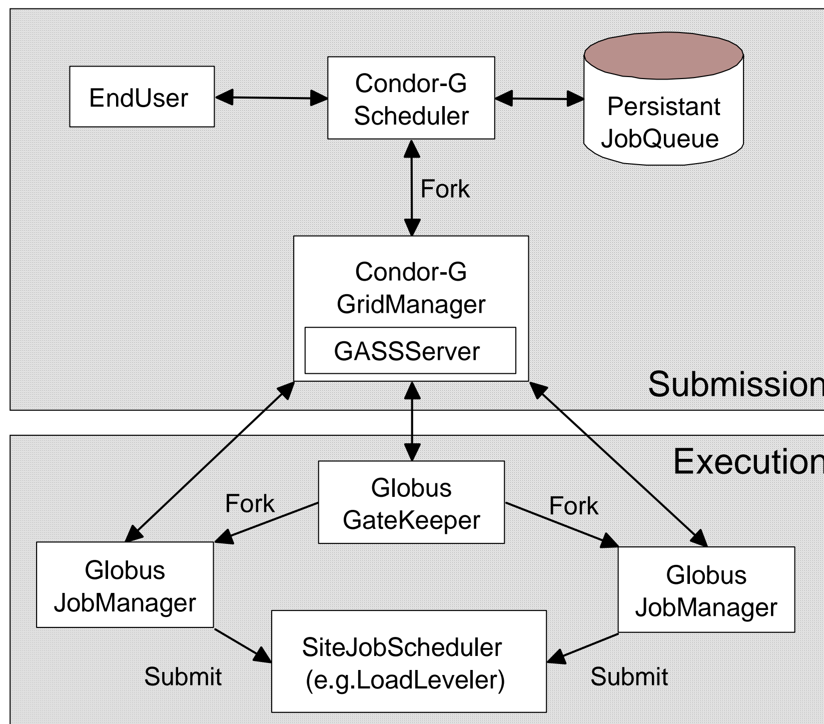


Abbildung 14: Condor-G im Zusammenspiel mit Globus [Frey, 2001]

Der Benutzer übergibt einen Job an den Condor-G Scheduler. Dieser wählt die Ressource anhand von Informationen vom Condor-G Collector aus, startet und überwacht den Job und benachrichtigt den Benutzer im Falle eines Fehlers oder der Beendigung des Jobs. Sobald der erste Job eines Benutzers an den Condor-G Scheduler übergeben worden ist, wird der Grid-Manager-Dämon gestartet. Er ist für die Abarbeitung aller Jobs eines Benutzers zuständig und beendet sich, sobald keine Jobs mehr zu bearbeiten sind. Für jeden Job startet der Grid-Manager über den Globus Gatekeeper per GRAM-Protokoll einen Globus JobManager. Dieser wiederum besorgt sich per GASS-Protokoll vom Grid-Manager den auszuführenden Job sowie die benötigten Dateien. Wenn der Job vom JobManager auf die Ressource übergeben und gestartet worden ist, benachrichtigt dieser den GridManager. Der Condor-G Scheduler erhält vom GridManager diese Information und speichert den Jobstatus und die URL des Jobs in der Datenbank.

Eine weitere Möglichkeit, den Job auszuführen, ist der sog. „GlideIn“-Mechanismus. Dabei startet der JobManager nicht den eigentlichen Job auf der Ressource, sondern einen Dämonprozess. Dieser macht die Verfügbarkeit der Ressource im Condor-G Collector publik. Die Portabilität und der Schutz der Ressourcen wird dadurch erreicht, dass jeder Task in einer „Sandbox“ ausgeführt wird. Bei jedem Task wird regelmäßig ein Checkpointing durchgeführt und auf dem ursprünglichen System abgelegt. Somit kann der Task im Falle eines Fehlers oder, falls die Ressource, z.B. ein Desktop, für andere Dinge benötigt wird, auf eine neue Ressource migriert werden.

5.2.4 Weitere Grid-Scheduler

Neben diesen drei Grid-Schedulern gibt es noch weitere bekannte Scheduler wie:

- **Silver/Maui**
  Der Silver Scheduler [Schopf, 2002] wird am Ohio Supercomputer Center eingesetzt. Er bietet die Möglichkeit zur Reservierung von Ressourcen, und er besitzt einen sehr guten Simulator, mit dessen Hilfe auf der Grundlage von Log-Files das Verhalten der lokalen Schedulerregeln angepasst werden kann. Der große Nachteil von Silver ist es, dass er nur mit Maui als lokalen Scheduler zusammenarbeitet.

- **Prophet**
  Prophet [Weissman, 1999] verteilt automatisch SPMD (Single Program Multiple Data) auf Ressourcen. Dazu verwendet er Informationen der Anwendung und der Ressourcen, um der Art und die Anzahl der Ressourcen festzulegen. Prophet teilt die Anwendung in Teilanwendungen und Teildaten auf, um damit die asynchrone und parallele Bearbeitung der partitionierten Daten zu ermöglichen.

- **MARS**
  MARS [Gehring, 1996] ist ein Framework für einen Grid-Scheduler, der anhand von dynamischen Informationen (Prozessorauslastung, Netzwerkauslastung und statistischen Daten) eine Lastverteilung und die Migration von Tasks auf andere Ressourcen vornimmt. Die Implementierung unterstützt in C implementierte Jobs, wobei parallele Tasks über das Message Passing Interface (MPI) kommunizieren können. Die Entwicklung des Frameworks ist eingestellt.

5.3 Scheduling-Policies

Scheduling-Algorithmen können in unterschiedliche Klassen (siehe Kapitel 2.3.1) eingeteilt werden. Exemplarisch werden einige Policies vorgestellt:

5.3.1 Interaktiver Modus

- **OLB**
  Oppertunistic Load Balancing (OLB) weist die Jobs in beliebiger Reihenfolge der nächsten zur Verfügung stehenden Ressource zu [Wu, 2000].

- **MCT**
  Minimum Completion Time (MCT) weist den Job der Ressource zu, welche am frühesten mit dessen Ausführung fertig ist [Maheswaren, 1999].

- **MET**
  Minimum Execution Time (MET) weist den Job der Ressource zu, welche die kürzeste Ausführungszeit für den Job hat [Maheswaren, 1999].

- **FCFS**
First-Come First-Serve (FCFS) arbeitet mit einer geordneten Liste von freien Ressourcen. Ein Job wird immer der ersten Ressource in der Liste zugeteilt, und die Ressource wird aus der Liste entfernt. Sobald der Job ausgeführt wurde, wird die Ressource wieder an dem Ende der Liste eingefügt. Mit dieser Strategie wird der höchste Durchsatz im Hinblick auf die Ausführungszeit eines Jobs erreicht. Dieses wird jedoch auf Kosten der ungleichmäßigen Auslastung der Ressourcen erreicht [James, 1999].

5.3.2 Batch-Modus

- **Min-Min**
Für jeden Job wird die Ausführungszeit auf jeder Ressource berechnet. Derjenige Job mit der geringsten Ausführungszeit und wird ausgewählt und auf der dazugehörenden Ressource ausgeführt. Die restlichen Jobs werden nach dem gleichen Verfahren auf die Ressourcen verteilt [Wu, 2000].

- **Max-Min**
Dieser Algorithmus ähnelt dem Min-Min-Algorithmus. Es wird ebenfalls für jeden Job die kürzeste Ausführungszeit und die jeweilige Ressource berechnet. Aus dieser Menge wird der Job mit der längsten Ausführungszeit ausgewählt und auf der dazugehörigen Ressource ausgeführt. Die restlichen Jobs werden nach dem gleichen Prinzip verteilt [Wu, 2000].

- **Segmented Min-Min**
Der Segmented Min-Min-Algorithmus wurde entwickelt, weil der Min-Min-Algorithmus zuerst die kurzen Jobs und danach die längeren Jobs verteilt. Dies führt zu einer ungleichmäßigen Auslastung der Ressourcen. Die Idee dabei ist, die Jobs nach ihrer berechneten Ausführungszeit zu ordnen. Diese geordnete Liste wird in Segmente unterteilt und zuerst das Segment mit den längsten Jobs nach dem Min-Min Algorithmus verteilt und danach das Segment mit den nächst kleineren Jobs usw. Dadurch kann eine gleichmäßigere Auslastung erreicht werden [Wu, 2000].

- **Self-Tuning dynP**
Der dynP-Algorithmus versucht auf die Eigenschaft zu reagieren, dass die Leistungsfähigkeit von Policies von der Charakteristik der Jobs abhängt. Dazu wird für eine Menge an Jobs der Ausführungsplan für die drei Policies FCFS, Min-Min und Max-Min berechnet. Je nach gewünschtem Qualitätskriterium, z.B. geringe durchschnittliche Antwortzeit, wird die entsprechende Policy angewendet [Streit, 2002a][Streit, 2002b].

5.4 Jobbeschreibungssprachen

Dieses Kapitel stellt zum einen die Jobbeschreibungssprache des Globus Toolkits und zum anderen die Standardisierung einer neuen Jobbeschreibungssprache des Global Grid Forums vor. Sie dienen dazu, dass Benutzer Jobs an den Scheduler übergeben können.

5.4.1 Resource Specification Language (RSL)

Für den Austausch von Ressourcenbeschreibungen innerhalb des Grids dient die Resource Specification Language (RSL) [Globus, 2003a].

- **Syntax**
  Die Komponenten der Globus Resource Management Architektur tauschen RSL-Ausdrücke aus, um in der Zusammenarbeit mit anderen Komponenten Verwaltungsaufgaben durchzuführen. Die RSL liefert dazu die Grundsyntax, um komplexe Ressourcenbeschreibungen zu erstellen. In dieses Grundgerüst fügen die Resource Management Komponenten Attribut/Wert-Paare ein, die jeweils eine Eigenschaft der Komponente beschreiben. Im folgenden ist die BNF-Grammatik für eine RSL-Anfrage dargestellt:

  ```
  specification   := request
  request         := conjunction | disjunction | parameter
  conjunction     := & request-list
  disjunction     := | request-list
  request-list    := ( request ) request-list | ( request )
  parameter       := parameter-name op value
  op              := = | > | < | >= | <= | !=
  value           := ([a..Z][0..9][_])+
  ```

- **Semantik**
  Die Konjunktion „&“ bzw. Disjunktion „|“ entsprechen einer logischen UND- bzw. ODER-Verknüpfung aus Konjunktionen, Disjunktionen oder Parametern. Die Komparatoren vergleichen, ob der Wert eines Parameters größer als „>“, kleiner als „<“, größer gleich „>=“, kleiner gleich ”<‘=“ oder ungleich „!=“ ist. Ein gültiger Wert für „value“ kann aus Ziffern, Buchstaben und Unterstrichen bestehen.

- **Beispiel**
  Ein Beispiel mit möglichen GRAM RSL Parametern:

  ```
  &((executable=myProg.sh)(argument="a  b")(minMemory >= 512)
  (count = 5))
  ```

  Die Anfrage lautet:

  „Führe das Programm myProg.sh mit den Argumenten a und b aus. Das Programm benötigt fünf Knoten mit mindestens 512 MB Speicher.“

5.4.2 Job Submission Description Language (JSDL)

JSDL wird von einer Arbeitsgruppe des Global Grid Forums (GGF) entwickelt [GGF, 2003a]. Das GGF ist eine Gemeinschaft von Forschern, die es sich zur Aufgabe gemacht haben, Grid-Technologien zu entwickeln und voranzutreiben [GGF, 2003b]. Das Ziel der XML-Sprache JSDL ist es, Interoperabilität zwischen

aktuellen und zukünftigen Schedulern durch die Standardisierung und Vereinigung der Möglichkeiten vorhandener Sprachen zu gewährleisten. Mit JSDL werden Jobs und deren benötigte Umgebung zur Ausführung beschrieben. Ein JSDL-Dokument enthält die Anforderungen an die Ressourcen, die Beschreibung von Abhängigkeiten zu anderen Jobs, die Beziehungen zu Dateien und die Architektur des auszuführenden Jobs. Der Benutzer übergibt dieses JSDL-Dokument für die Ausführung an den Scheduler (siehe Abbildung 15). Dabei kann es sich um einen lokalen Scheduler oder einen Grid-Scheduler handeln. Die Voraussetzung des Schedulers ist es, dass er die JSDL versteht. Dafür kann er eine API zur Verfügung stellen. Distributed Resource Management Application API (DRMAA) ist ein Beispiel für solch eine API, die von einer weiteren Arbeitsgruppe des entwickelt wird. Grid-Scheduler müssen weitere Möglichkeiten haben, um in der Zusammenarbeit mit lokalen Schedulern Ressourcen zu reservieren oder Jobs auszuführen. Hier wird an die Notwendigkeit einer standardisierte Resource Description Language (RDL) gedacht, die Sprachen wie RSL und ClassAds ersetzen soll.

Von DRMAA gibt es einen ersten Vorschlag für eine Spezifikation der API. Bei JSDL arbeitet die Arbeitsgruppe an einem ersten Entwurf. Bis jetzt gibt es ein Dokument mit den Anforderungen. RDL ist eine Idee und es gibt noch keine konkrete Arbeitsgruppe, die sich in enger Zusammenarbeit mit der Arbeitsgruppe von JSDL mit RDL auseinandersetzt.

Das LabGrid nutzt zur Übermittlung von Jobs an den Grid-Scheduler eine eigene XML-Sprache, die für zusätzliche Anforderungen, wie die benötigte Softwarekonfiguration erweitert werden muss. Zur Übermittlung der Jobs an den lokalen Scheduler LoadLeveler wird weiterhin RSL von Globus verwendet. Zum jetzigen Zeitpunkt gibt es noch keine endgültigen Standardisierungen, die für ein Grid in der Produktion eingesetzt werden können. Dies betrifft JSDL, RDL und DRMAA. Daher können die Entwicklungen der GGF erst in der Zukunft verwendet werden.

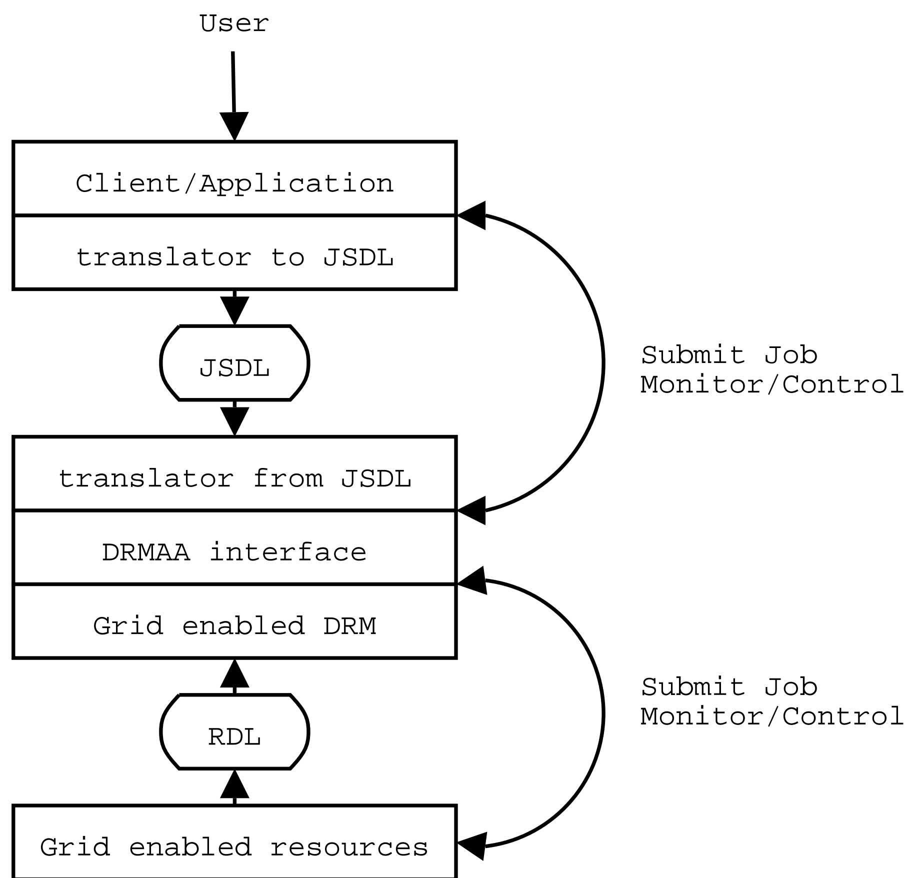


Abbildung 15: Prozess der Job-Übergabe

6 Architektur des Frameworks

Im ersten Teil wird auf den ersten Entwurf der Architektur des Frameworks und dessen Vor- und Nachteile eingegangen. Der zweite Teil stellt die Architektur des endgültigen und umgesetzten Frameworks sowie eine mögliche Erweiterung vor.

In den folgenden Abbildungen zur Architektur des Schedulers sind zugunsten der Übersichtlichkeit Logon-, QueryStatus- und Cancel-Servlet aus der Abbildung 10 nicht dargestellt, da diese für das Verständnis der Architektur nicht beitragen. Die Servlets haben sich in der Funktion und Struktur nicht verändert. Lediglich von dem Submit-Servlet wurde ein Teil der Kontrolle über den Ablauf der Jobübermittlung an das Policy-Interface aus Kapitel 6.1.1 übertragen. Die Kanten zwischen den einzelnen Komponenten stellen den Datenfluss dar.

6.1 Das Framework

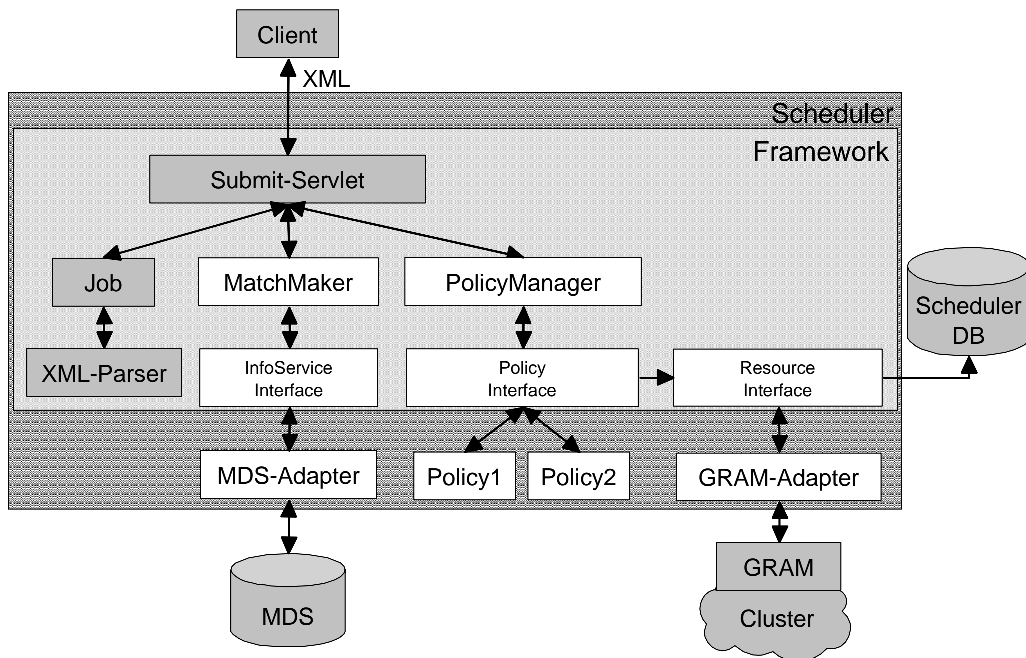


Abbildung 16: Das Framework

Die Architektur des ursprünglichen Schedulers besitzt die Nachteile, dass eine Matchmaking-Komponente fehlt, die Policy nicht austauschbar ist und kein anderes Informationssystem einer Grid-Middleware verwendet werden kann (siehe Kapitel 3.3.2).

Der erste Entwurf der Architektur für das neue Framework ist in Abbildung 16 dargestellt. Die XML-Jobbeschreibungssprache wurde für die Auswahl der Scheduling-Policy und der Angabe der Jobanforderungen an die Ressource erweitert (siehe Kapitel 6.4). Der Ablauf bezüglich Aufruf des Submit-Servlets durch

den Klienten, Überprüfung der Autorisierung, Parsen der Jobbeschreibung durch den XML-Parser und Erstellung des Job-Objekts hat sich gegenüber der Darstellung in Kapitel 3.3.2 nicht geändert.

6.1.1 Neue Komponenten

Neu in dieser Architektur ist der MatchMaker, der PolicyManager und die drei Schnittstellen InfoService-Interface, Policy-Interface und Resource-Interface.

- **MatchMaker**
  Der MatchMaker ist für die Auswahl der potentiellen Ressourcen zuständig.

  Weil es keine Möglichkeit zur genaueren Spezifikation der Ressourcen durch Jobanforderungen gibt, erhält der ursprüngliche Scheduler bzw. dessen Policy sämtliche Ressourcen vom zentralen Informationssystem MDS. Er muss sich somit darauf verlassen, dass alle Jobs auf allen Ressourcen ausführbar sind.

  Nun bietet die XML-Jobbeschreibungssprache die Möglichkeit an, die Menge der Ressourcen durch die Jobanforderungen eines Jobs einzuschränken. Der MatchMaker liefert anhand der Jobanforderungen, und der benötigten Attribute einer Ressource für die Ausführung der gewählte Policy und dem Zugriff auf das zentrale Informationssystem alle diejenigen Ressourcen zurück, die folgende zwei Bedingungen erfüllen:

  - Die Ressource besitzt die benötigten Attribute für die Ausführung der Scheduling-Policy.
  - Die Ressource erfüllt alle Jobanforderungen, die in der XML-Jobbeschreibung angegeben sind.

  Durch die Erfüllung dieser Bedingungen werden nur noch die potentiellen Ressourcen zurückgeliefert, auf denen der Job garantiert ausführbar ist. Damit können auch Jobs, die nicht auf allen Ressourcen ausführbar sind, dem Grid-Scheduler übergeben werden, bzw. es können weitere heterogene Ressourcen in das Grid eingebunden werden.

  Eine Alternative dazu wäre, Ressourcen zu klassifizieren und durch die Angabe der Klasse auszuwählen, bzw. ein Matchmaking mit den Jobanforderungen und den Eigenschaften einer Klasse durchzuführen.

- **PolicyManager**
  Er hat die Aufgabe, die vorhandenen Scheduling-Policies zu verwalten. Dazu werden die Policies beim PolicyManager registriert. Über das Policy-Interface hat der PolicyManager die Möglichkeit, die benötigten Attribute einer Ressource für die Ausführung der Policy abzufragen.

  Über die XML-Jobbeschreibungssprache kann die Policy vom PolicyManager ausgewählt werden und erhählt den auszuführenden Job und die Menge potentieller Ressourcen vom Submit-Servlet.

Die spezifischen Komponenten des Globus Toolkits und die Policies wurden durch die drei Schnittstellen InfoService-Interface, Policy-Interface und Resource-Interface von dem Framework entkoppelt. Die Aufgabe der einzelnen Schnittstellen sind folgende:

- **InfoService-Interface**
  Das InfoService-Interface ist eine generische Schnittstelle, um Informationen über Ressourcen vom Informationssystem der Grid-Middleware zu erhalten. Das Interface wird von einem Adapter, dem MDS-Adapter, für den Zugriff auf das MDS implementiert. Er ermöglicht je nach Informationssystem den spezifischen Zugriff und liefert die Antworten der Anfragen zurück.
- **Policy-Interface**
  Das Framework kann jede Scheduling-Policy verwenden, die das Policy-Interface implementiert. Es bietet dem Policy-Manager somit für jede implementierte Scheduling-Policy die gleiche Schnittstelle zur Übergabe der Jobs und der Menge der potentiellen Ressourcen an. In Abbildung 16 sind mögliche Policies durch die Komponenten Policy1 und Policy2 dargestellt.
- **Resource-Interface**
  Das Ressource-Interface ist die Schnittstelle, um Jobs an die Grid--Middleware zu übergeben oder um Jobs abzubrechen. Die Schnittstelle wird je nach Grid-Middleware von einem spezifischen Adapter implementiert, der in diesem Fall die Möglichkeit über den GRAM-Adapter bietet, einen Job an die GRAM-Komponente einer bestimmten Ressource zu übergeben.

Die Schnittstellen werden in Kapitel 6.3 genauer betrachtet.

6.1.2 Funktionsweise

Sobald das Submit-Servlet aufgerufen wird, registriert es sämtliche Policies an dem PolicyManager. Das Servlet kann mit dem Namen der Policy aus der XML-Jobbeschreibung über den PolicyManager die benötigten Attribute der Ressourcen für die Ausführung der Policy abfragen. Die benötigten Attribute und die Jobanforderungen werden dem MatchMaker mitgeteilt. Damit kann der MatchMaker mit nur einer Anfrage über das InfoService-Interface das zentrale Informationssystem abfragen. Vom Scheduler werden keine weiteren Anfragen an das Informationssystem benötigt. Die exaktere Formulierung der Anfrage reduziert in der Antwort die Anzahl der Ressourcen und deren Attribute im Vergleich zu dem ursprünglichen Scheduler.

Falls der PolicyManager die Attribute der Ressourcen für eine Policy nicht abfragen könnte, wie es bei dem ursprünglichen Scheduler der Fall ist, müsste der MatchMaker entweder die potentiellen Ressourcen mit sämtlichen Attributen zurück liefern, oder die Policy müsste für die Auswahl der Ressource nochmals weitere Anfragen zu den Ressourcen an das zentrale Informationssystem stellen.

Als Alternative kann der Scheduler die Resultate aus dem Informationssystem zwischenspeichern und selbstständig seinen Zwischenspeicher aktualisieren. Dadurch würden mehrere Anfragen an den Zwischenspeicher zeitlich nicht mehr relevant,

und die Anfragedauer an das Informationssystem fiele weg. Der Nachteil dabei ist, dass die Informationen über die Ressourcen ein weiteres Mal zwischengespeichert würden und damit älter als eine Anfrage an das Informationssystem sein könnten. Diese Lösung wird von der IBM implementiert.

Der ursprüngliche Scheduler befragt zuerst das MDS nach allen Namen der Ressourcen. Im zweiten Schritt befragt er das MDS nach allen Klassen (siehe Kapitel 3.4) der Ressourcen, die die benötigten Attribute für die Policy haben.

Die eine Anfrage des Frameworks an das MDS und die reduzierten Antwortdaten resultieren dem gegenüber in einer kürzeren Antwortzeit und einer geringeren Netzwerkbelastung. Die Vorraussetzung dafür ist, dass sich alle Attribute einer Ressource in der LDAP-Hierarchie auf einer Kontextebene befinden. Falls sich ein Attribut in der LDAP-Hierarchie auf einer anderen Kontextebene befindet, wird eine weitere Anfrage benötigt.

Im nächsten Schritt übergibt das Submit-Servlet die potentiellen Ressourcen und das Job-Objekt an den PolicyManager. Dieser leitet diese Informationen an die entsprechende Policy weiter. Von nun an ist die Policy für die Ausführung des Jobs verantwortlich. Abhängig von der Implementierung der konkreten Policy generiert das Submit-Servlet die Antwort mit XML sofort oder erst nach der Übergabe des Jobs an die Ressource. Schließlich gibt es die Antwort an den Klienten zurück.

Dies ist ein weiterer wesentlicher Unterschied in der Architektur zu dem ursprünglichen Scheduler. Eine Policy kann jetzt selbst den Job über das Resource-Interface und den konkreten Adapter an die Ressource übergeben. Bei der vorherigen Architektur muss die Policy sofort eine Ressource auswählen, damit das Submit-Servlet über das Job-Objekt den Job an die Ressource übergeben und die Antwort für den Klient generieren kann. Für den Fall, dass die Policy blockiert bzw. verzögert ausliefert, würde bis zu der Auslieferung keine Rückmeldung an den Klienten erfolgen. Nun hat eine Policy die Möglichkeit, Jobs in einer Warteschlange zu sammeln und verzögert auszuliefern, wie es bei verschiedenen Schedulingalgorithmen notwendig ist. Dazu gehören die Algorithmen aus Kapitel 5.3.2, die im Batch-Modus arbeiten.

Damit können Jobs auf der Ebene des Grid-Schedulers in einer Warteschlange gehalten werden, bevor sie in eine Warteschlange auf den Ressourcen eingereiht werden. Das hat folgende Vorteile:

1. Es kann bei der Auswahl der Ressource Rücksicht auf die anderen Jobs in der Warteschlange genommen werden. Dadurch läßt sich eine gleichmäßigere Auslastung des Grids erreichen.

2. Der Job hat immer noch die Chance, auf der für ihn optimalen bzw. auf der nächsten freien Ressource im Grid und nicht nur auf einer Ressource aus der Teilmenge des Grids zur Ausführung zu kommen.

Damit eine Policy Jobs verzögert ausliefern kann, muss die Policy einen neuen Thread starten, der für die Abarbeitung des Jobs zuständig ist. Zusätzlich muss die Implementierung die Methode *schedule* der Klasse AbstractPolicy überschreiben.

Die **Vorteile dieser Architektur**:

- Durch den Einsatz der drei Schnittstellen InfoService-Interface, Policy-Interface und Ressource-Interface in der Verbindung mit spezifischen Adaptern wird das Framework von konkreten Policies und der darunterliegenden Grid-Middleware entkoppelt. Somit ist das Framework generisch und dadurch wesentlich flexibler und universeller einsetzbar.
- Es können mehrere Policies verwaltet werden. Damit kann für bestimmte Jobs oder bei einer bestimmten Auslastung, wie bei dynP [Streit, 2002a][Streit, 2002b], zwischen zwei Jobs auf eine andere Policy umgeschaltet werden.
- Es können unterschiedliche Informationssysteme abgefragt werden.
- Es kann eine andere Grid-Middleware verwendet werden.
- Es ist nur ein Zugriff auf das zentrale Informationssystem für das Scheduling nötig.
- Die MatchMaking-Komponente erlaubt es, Jobs im Grid auszuführen, welche nicht auf jeder Ressource ausführbar sind.

6.2 Optimierung des Frameworks

Ein Nachteil der vorgestellten Architektur ist jedoch, dass sobald eine Policy Jobs verzögert ausliefert, das Resultat aus dem MatchMaking-Prozess veraltet ist und der Job unter Umständen auf einer nicht mehr optimalen Ressource ausgeführt wird. Das hat je nach gewählter Policy und dem Zustand des Grids zur Folge, dass beispielsweise das Grid nicht wie gewünscht gleichmäßig ausgelastet wird oder der Job nicht schnellstmöglich ausgeführt wird. Um diesem Nachteil entgegen zu treten, muss die Policy die Möglichkeit haben, den MatchMaker unmittelbar vor der Übergabe des Jobs über das Resource-Interface abzufragen, damit die Informationen über die Ressourcen aktuell sind. Weiterhin kann die Policy im Falle einer verzögerten zeitlich eng beieinander liegenden Auslieferung von Jobs selbst entscheiden, ob der MatchMaker überhaupt für jeden einzelnen Job abgefragt wird oder nicht. Mit einer Anfrage können z.B. mehrere gleichartige Jobs bedient werden. Viele Informationssysteme verwenden einen Cache, um kürzere Antwortzeiten durch das Zwischenspeichern von Informationen für eine bestimmte Zeit zu erreichen. Die Cachezeit gibt in diesem Fall das maximale Alter einer Information an. Für das Scheduling macht es keinen Unterschied, ob das zentrale Informationssystem mehrmals bei einer begrenzten Menge von schnell aufeinander folgenden Jobs abgefragt wird oder nicht. Aber es kann der Durchsatz des Schedulers erhöht und das zentrale Informationssystem entlastet werden, wenn das zentrale Informationssystem nur einmal abgefragt wird.

Das Problem wird durch die Architektur, wie in Abbildung 17 dargestellt, gelöst. Sie bietet die Möglichkeit, dass Policies selbst das zentrale Informationssystem über den MatchMaker abfragen können. Das Submit-Servlet ruft jetzt nicht mehr den MatchMaker, sondern direkt den PolicyManager mit der gewünschten Policy und dem auszuführenden Job-Objekt auf. Der PolicyManager wiederum ruft die Policy auf, welche den MatchMaker direkt befragt und über das Resource-Interface den Job zur Ausführung bringt.

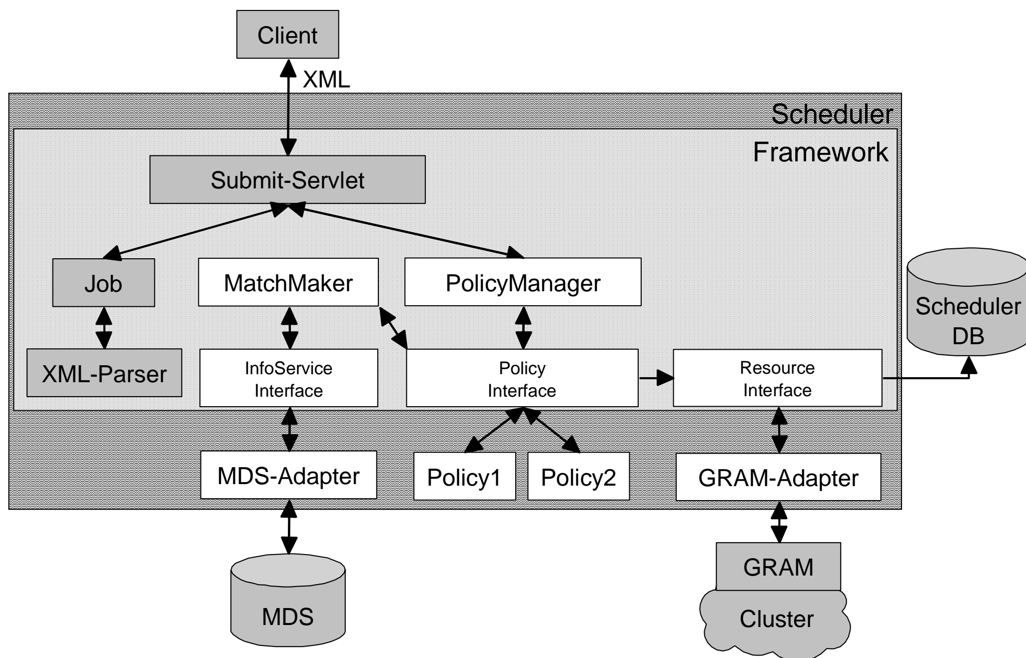


Abbildung 17: Das optimierte Framework

Die geänderte Architektur entkoppelt die Komponenten. Diese arbeiteten jetzt service-orientierter, weil der Kontrollfluss und der Austausch sämtlicher Daten nicht mehr komplett über das Submit-Servlet, wie im ursprünglichen Scheduler, gesteuert werden.

Im Hinblick auf die Policies bietet das Framework mit dieser Architektur ein hohes Maß an Flexibilität, da es folgendes ermöglicht:

- Verwendung mehrerer Policies und deren Verwaltung
- Einfache Abfrage der potentiellen Ressourcen durch die Policy beim MatchMaker
- Eine Policy kann selbstständig und verzögert Jobs über das Resource-Interface ausführen, ohne den weiteren Kontrollfluss des Schedulers zu behindern. Beispielsweise kann die Policy in einem eigenen Thread ausgeführt werden.

Für die folgenden Probleme, die bei einer gewissen Anzahl von Ressourcen entsteht, wurde keine Lösung in dem entwickelten Framework umgesetzt:

Für die Skalierbarkeit ist es notwendig, dass die Anzahl der Ressourcen, die der MatchMaker zurück liefert, auf eine einstellbare Zahl, z.B. x, begrenzt ist. Das führt zu dem Problem, dass möglicherweise der Job nicht auf der optimalen Ressource ausgeführt wird. Dies hat damit auch einen Einfluss auf die gleichmäßige Auslastung des Grids. Angenommen, das Informationssystem würde bei einer

Anfrage die Ressourcen immer in der gleichen Reihenfolge zurück liefern. Dann würden im Extremfall die Jobs immer nur auf den ersten x Ressourcen verteilt werden. Es muss demnach eine Möglichkeit geben, falls die ersten x Ressourcen ausgelastet sind, dass nach den nächsten x Ressourcen gefragt werde kann.

Das MDS bietet sowohl die Möglichkeit an, die Anzahl der Antworten zu begrenzen, als auch die maximale Dauer einer Anfrage anzugeben. LDAP, das für den Zugriff auf das MDS verwendet wird, sieht dafür keine Möglichkeiten vor.

Das Frameworks setzt folgende Anforderungen um:

- Das Framework besitzt eine Matchmaking-Komponente für die Auswahl der potentiellen Ressourcen.
- Es können mehrere Policies verwaltet werden und sind einfach austauschbar.
- Es kann ein anderes Informationssystem verwendet werden.

6.3 Schnittstellen

Beim Entwurf von Schnittstellen gibt es das Problem, dass Schnittstellen entweder zu spezifisch oder zu allgemein sind. Hier wurde darauf geachtet, dass die Schnittstellen schlank sind und nur die nötigen Methoden enthalten, damit es nur eine geringe Kopplung gibt, und die Schnittstellen leicht implementiert werden können. Zusätzlich wurde darauf geachtet, dass keine spezifischen Eigenheiten der verwendeten Grid-Middleware, z.B. vom Globus-Toolkit, in den Schnittstellen enthalten sind.

6.3.1 Anbindung von Informationssystemen

Das Informationssystem einer Grid-Middleware, z.B. dem MDS, wird über das **InfoService-Interface** mit einem Adapter, z.B. dem MDS-Adapter, (siehe Abbildung 18) eingebunden.

Das Ziel der Anbindung ist es, Informationen über Ressourcen zu erhalten. Dazu wird eine Methode benötigt, welche alle verfügbaren Ressourcen zurück liefert. Weiterhin soll es eine Methode geben, damit zur Laufzeit die Anbindung an das Informationssystem neu konfiguriert werden kann. Mit Hilfe dieser Methode könnte zwischen verschiedenen Informationssystemen auswählt werden.

Um diese Anforderungen zu erfüllen, wurden folgende zwei Methoden entworfen:

- *Vector getResources(Vector requirements, Vector scheduleAttr)*
  Diese Methode bietet die Möglichkeit zur Abfrage der potentiellen Ressourcen vom Informationssystem. Über die zwei Parameter können die Jobanforderungen (*requirements*) und die benötigten Attribute für das Ausführen der Policy (*scheduleAttr*) übergeben werden. Die Jobanforderungen werden als Vector von Objekten der Klasse *Requirement* und die benötigten Attribute als Vector von Objekten der Klasse *String* übergeben. Der Rückgabewert ist ein Vector von Objekten der Klasse *Resource* und enthält alle potentiellen Ressourcen mit den gewünschten Attributen.

- *void reload(String propertiesFile, String mappingFile)*
  Mit der Methode *reload* kann zur Laufzeit der Adapter, der das InfoService-Interface implementiert, neu konfiguriert werden. Dazu kann der Pfad zu dessen Properties-File und zu dessen Mapping-File angegeben werden.

Jedes Informationssystem hat eigene Attributnamen. Da die Attribute z.B. von der Policy für die Auswahl der Ressource ausgewertet werden müssen, müsste bei einem Wechsel des Informationssystems oder dem Ändern eines Attributnamens der Code des Frameworks und sämtlichen Policies angepasst werden. Um dieses Problem zu lösen, gibt es für jedes Informationssystem ein eigenes Mapping-File. Es enthält eine Menge von Abbildungen von einem Informationsdienst-spezifischen Attributnamen auf einen für das Framework verständlichen Attributnamen. Damit sollen Informationsdienst-spezifische Eigenheiten unabhängig vom Framework bleiben.

Der folgende Auszug aus der Mapping-File für die Implementierung der Schnittstelle Information Service durch den MDS-Adapter verdeutlicht dessen Aufbau:

```
<MAPPING NAME="MDS">
  <MAP NAME="hostname">Mds-Host-hn</MAP>
  <MAP NAME="os">Mds-os-name</MAP>
  <MAP NAME="os-version">Mds-os-version</MAP>
  <MAP NAME="arch">Mds-architecture</MAP>
</MAPPING>
```

Das MDS-spezifische Attribut „Mds-Host-hn“ wird z.B. auf das Attribut „hostname“ innerhalb des Frameworks abgebildet.

Als Alternative für die Abfrage der Ressourcen kann auch nur der Klassenname (siehe in Kapitel 3.4) von einem lokalen Scheduler angegeben werden. Bei einem Grid mit Clustern, welche alle z.B. LoadLeveler verwenden, wie es bei der IBM der Fall ist, würde dies funktionieren.

Die Wahl der Methode getResources mit den Parametern requirements und scheduleAttr hat den Vorteil, dass sie ganz allgemein angeben, welche Anforderungen an die Ressource gestellt werden und welche Attribute für die Ausführung der Policy benötigt werden. Die Implementierung kann in diesem Fall sämtliche Ressourcen und auch alle Attribute der Ressourcen zurückgeben. Die Auswahl, welche Ressource geeignet ist, trifft der MatchMaker. Die Implementierung hat jedoch auch die Möglichkeit, anhand der beiden Parametern das Ergebnis auf das notwendigste zu reduzieren. Als Beispiel kann man sich eine Datenbankanfrage vorstellen, welche in ihrem Select-Statement die benötigten Attribute und im Where-Statement die Anforderungen an die Ressource angeben.

6.3.2 Anbindung von Scheduling-Policies

Scheduling-Policies werden über das Policy-Interface integriert.

Das Ziel einer Scheduling-Policy ist es, aus einer Menge an Ressourcen die optimale Ressource für einen bestimmten Job auszuwählen. Die Anforderungen an das

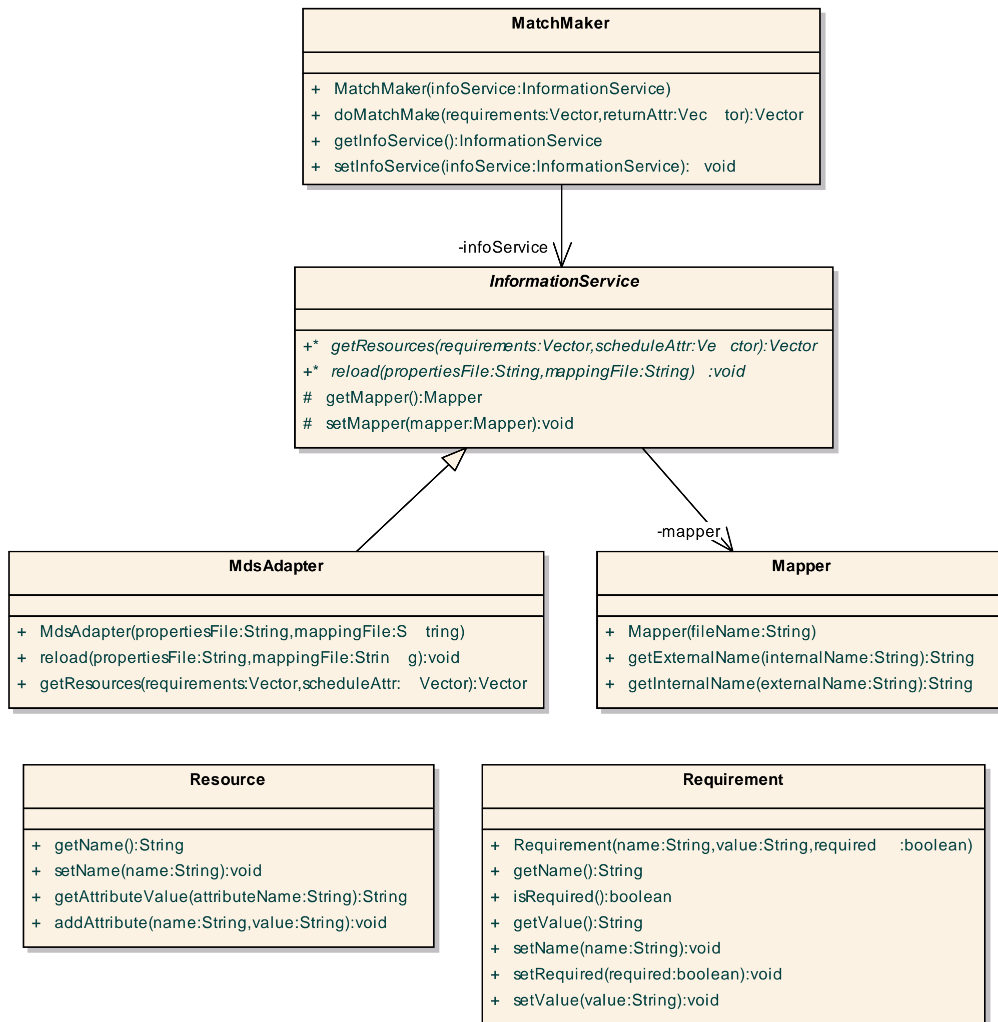


Abbildung 18: MatchMaker mit der Schnittstelle zum Informationssystem

Interface sind, dass Policies einfach implementiert und integriert werden können. Dazu wird eine Methode benötigt, welcher der Job und Informationen über die Ressourcen übergeben wird, und die optimale Ressource zurück liefert. Der Job kennt seine Anforderungen an die Ressource. Zusätzlich soll der Name einer Policy abfragbar sein. Als Alternative kann auch eine Methode verwendet werden, welcher nur der Job übergeben wird und diese erhält die Informationen über die Ressourcen vom Informationssystem und MatchMaker. Für das Interface wurde diese Methode gewählt, weil die Policy damit selbst entscheiden kann, wann und wo ein Job ausgeführt wird. Das Scheduling kann dabei online als auch offline erfolgen. Die Methode, welche Informationen über die Ressourcen als Parameter bekommt, hat den Nachteil, dass die Informationen von einem bestimmten Zeitpunkt sind. Je länger mit der Übergabe eines Jobs an die Ressourcen gewartet wird, desto älter und damit auch ungenauer werden die Informationen.

Um das Policy-Interface zu implementieren, muss eine Klasse für die Policy von der Klasse *AbstractPolicy* abgeleitet und die folgenden zwei abstrakten Methoden (siehe Abbildung 19) implementiert werden:

- *Vector getScheduleAttr()*
  Diese Methode gibt einen Vector mit Objekten der Klasse *String* zurück. Jedes Objekt enthält einen Namen eines Attributs einer Ressource. Die Attribute werden für die Ausführung dieser konkreten Policy benötigt.

- *Resource getQualifiedResource(Vector resources, Job job)*
  Die Methode *getQualifiedResource* liefert aus der Menge der potentiellen Ressourcen die Ressource, welche aus der Sicht der Policy für den nächsten Job am besten geeignet ist.

Falls eine Policy Jobs verzögert ausliefern will, muss diese die Methode *void schedule(Job job)* überschreiben. Sie übergibt den Job über das Resource-Interface an die ausgewählte Ressource.

In der aktuellen Implementierung arbeitet die Methode im interaktiven Modus. Sie fragt mit Hilfe des MatchMakers, den Attributen für die Policy und den Jobanforderungen nach den potentiellen Ressourcen. Diese werden der Methode getQualifiedResource übergeben. Der Job wird dann schließlich mit der optimale Ressource an den ResourceManager übergeben, und der Job wird ausgeführt.

6.3.3 Anbindung einer Grid-Middleware

Das Interface für die Anbindung einer Grid-Middleware dient zu der Steuerung der Ausführung von Jobs auf Ressourcen. Über dieses Interface kann ein Job an die Grid-Middleware übergeben, abgebrochen und der Status abgefragt werden. Dazu wurden folgende Methoden entworfen:

- *void cancel(Job job)*
  Dieser Methode kann ein Objekt der Klasse Job übergeben werden, und der Adapter bricht die Ausführung des Jobs ab.

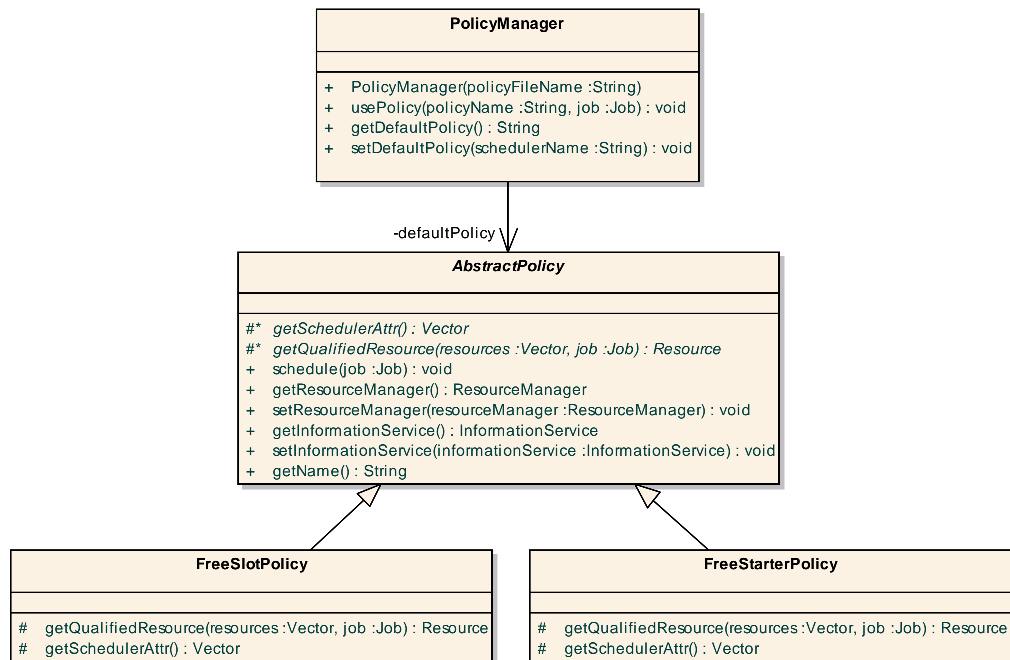


Abbildung 19: PolicyManager mit der Schnittstelle zu den Policies

- *void execute(Job job, Resource resource)*
  Die Methode *execute* benötigt als Parameter ein Objekt der Klasse Job als auch ein Objekt der Klasse Ressource. Der Job wird daraufhin auf der angegebenen Ressource ausgeführt.

Die beiden Methoden sind in Abbildung 20 dargestellt.

Zusätzlich ist noch vorstellbar, eine weitere Methode für das Rescheduling und Checkpointing (siehe Kapitel 2.3.1) von Jobs anzubieten. Der Grid-Scheduler entscheidet, ob der Zustand eines Jobs gesichert, auf eine andere Ressource verlagert oder zu einem späteren Zeitpunkt weiter ausgeführt werden soll. Die Jobinformationen kommen in der ursprünglichen Version über das Housekeeper-Konzept und in der jetzigen Lösung über die Callback-Lösung in die Scheduler-DB und können von dort abgefragt werden.

Die Methoden wurden möglichst generisch gewählt. Damit ein Job ausgeführt werden kann, muss der Grid-Middleware nur der Job und die Ressource auf der dieser Job ausgeführt werden soll, mitgeteilt werden. Um einen Job abzubrechen, muss man nur wissen, welcher Job abgebrochen werden soll. Für die Ressource, auf der dieser Job ausgeführt wird, ist die Grid-Middleware verantwortlich.

6.4 XML-Jobbeschreibungssprache

Die bisherige XML-Jobbeschreibungssprache des LabGrids aus Kapitel 3.5 muss für die Jobanforderungen und für die Auswahl der Policy erweitert werden.

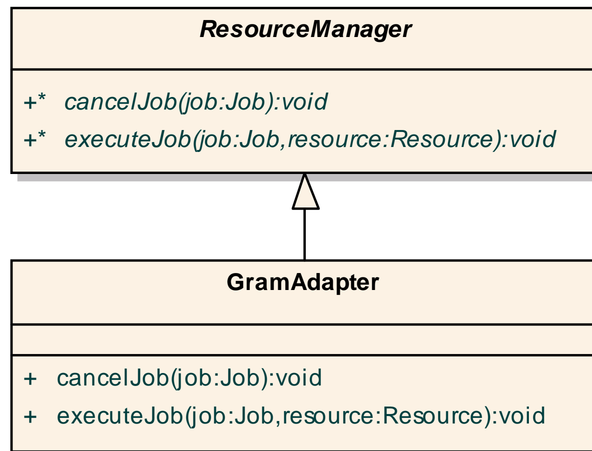


Abbildung 20: Schnittstelle zur Grid-Middleware mit dem GRAM-Adapter

Der Klient übergibt dem Scheduler den XML-Request, welcher den auszuführenden Job beschreibt. Für das neue Framework wurde dieser XML-Request um zwei zusätzliche Tags erweitert:

1. **Job-Anforderung für das Matchmaking**

   ```
   <REQUIREMENT NAME="os" REQUIRED="true">AIX</REQUIREMENT>
   ```

   beschreibt die Anforderung, dass der Job auf einer Ressource mit dem Attribute „os“ und dem Wert „AIX“ auszuführen ist. Dies bedeutet, dass der Job auf dem Betriebssystem „AIX“ lauffähig ist. Das Attribute „REQUIRED“ gibt an, ob diese Anforderung von der Ressource erfüllt sein muss (Wert: „true“) oder bevorzugt wird (Wert: „false“). Falls das Attribut „REQUIRED“ nicht angegeben wird, wird es standardmäßig auf „true“ gesetzt.

2. **Auswahl der Policy**

   ```
   <POLICY NAME="Round-Robin"></POLICY>
   ```

   beschreibt die Policy, die für das Scheduling verwendet werden soll. In diesem Fall ist es die „Round-Robin“-Policy. Zwischen Start- und Endtag kann noch ein String mit Parametern übergeben werden.

Das folgende Listing stellt den XML-Request für eine Jobausführung dar:

```
<IDEGRID>
  <REQUEST>
    <SUBMIT>

      <EXECUTABLE>/bin/echo</EXECUTABLE>
```

```
    <...>

    <REQUIREMENT NAME="os"           REQUIRED="true">AIX</REQUIREMENT>
    <REQUIREMENT NAME="os-version"   REQUIRED="true">5</REQUIREMENT>
    <REQUIREMENT NAME="arch">        REQUIRED="true">PowerPC</REQUIREMENT>

    <POLICY NAME="Round-Robin"></POLICY>

   </SUBMIT>
  </REQUEST>
</IDEGRID>
```

Der Job benötigt für die Ausführung als Betriebssystem AIX in der Version 5 und eine PowerPC-Rechnerarchitektur.

Die Erweiterung der Jobbeschreibungssprache erfüllt mit diesen zwei zusätzlichen Tags die Anforderungen, dass Ressourcenanforderungen und die Auswahl einer bestimmten Policy an den Scheduler übergeben werden können.

7 Prototyp

Für die Evaluierung des Frameworks wurde ein Prototyp erstellt. Dieser besteht aus dem Framework und der Implementierung der drei Schnittstellen InfoService-, Policy- und Resource-Interface. In diesem Kapitel wird die Implementierung der drei Schnittstellen näher betrachtet.

7.1 Scheduling-Policy

Der Algorithmus für die Auswahl der Ressource wird über das Policy-Interface an das Framework angebunden. In diesem Kapitel wird die Notwendigkeit für eine neue Scheduling-Policy diskutiert und die Implementierung des neuen Algorithmus vorgestellt.

7.1.1 Probleme der ursprünglichen Policy

Der ursprüngliche Algorithmus soll die Jobs an diejenige Ressource übergeben, welche prozentual am wenigsten ausgelastet ist. Das Problem dabei ist, dass für die Ermittlung der Auslastung nur das Verhältnis zwischen $freeSlots$ und $maxSlots$ verwendet wird. Diese Attribute alleine können jedoch nicht die freie und die gesamte Kapazität einer Ressource angeben, da diese noch durch die in Kapitel 7.1.2 erklärten Attribute $freeStarter$, $maxStarter$ und $maxjobs$ beschränkt werden. Dies führt zu einer sehr ungleichmäßigen Verteilung, wie in Abbildung 21 nach der Übergabe von zehn Jobs dargestellt ist. Werden jetzt noch mehr Jobs an den Grid-Scheduler übergeben, würde sich die Warteschlange auf Cluster B weiter füllen. Das hat den Grund, dass die Auslastung von Cluster B konstant bei $1 - \frac{freeSlots}{maxSlots} = \frac{45}{50} = 0,1$ ist, weil das Attribut $freeSlot$ nur abnimmt, falls ein Job ausgeführt wird. Das Attribut $freeSlots$ nimmt nicht ab, wenn er der Job in die Warteschlange eingereiht wird. Somit ist Auslastung von Cluster B (0,9) solange geringer und scheinbar die optimale Ressource, bis Job2 oder Job3 von Cluster A oder C beendet werden.

Dauern die Jobs auf Cluster A und C länger, werden neue Jobs in die Warteschlange von B eingereicht, bis diese die Länge $2 * maxSlots$ erreicht hat:

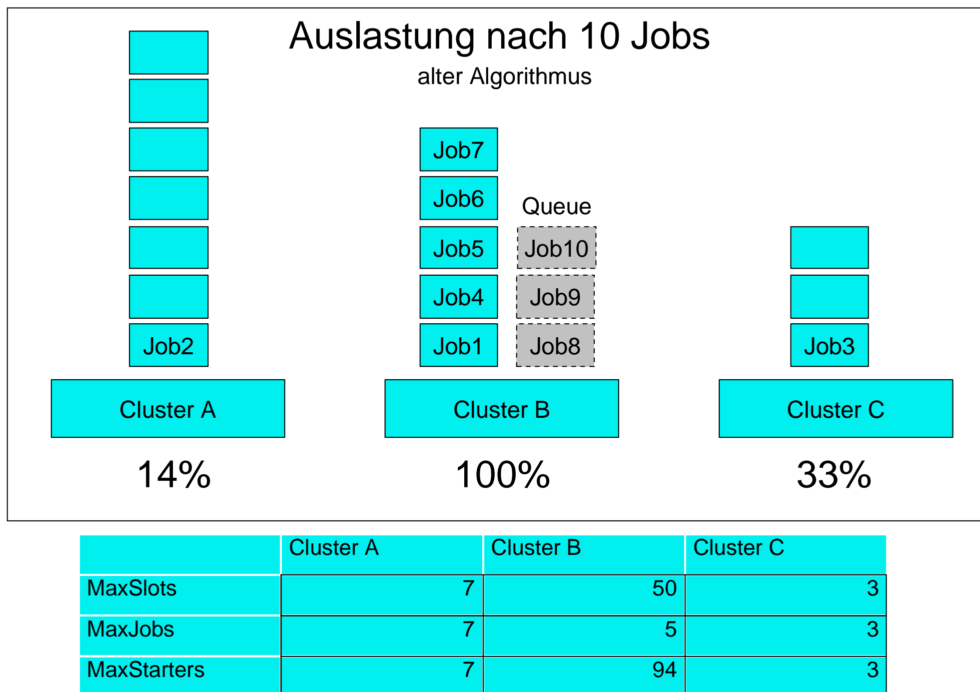
Auszug aus der ursprünglichen Policy:

```
if ((jobwait > 2*maxslots) && (freeslots > (maxslots/10+2)) ) {
 ...error: something wrong with host
}
```

Somit werden nach den ersten zehn Jobs die nächsten

$$2 * maxSlots - Jobs\ in\ der\ Warteschlange = 2 * 50 - 3 = 97\ Jobs$$

in die Warteschlange von Cluster B eingereiht. Bis zu diesem Zeitpunkt haben Cluster A und C nur einen Job. Danach gibt es für Cluster B jedesmal eine Fehlermeldung, dass mit der Ressource etwas nicht stimmt, und nach dem gleichen Verfahren werden die Jobs auf Cluster A und C weiterverteilt.



	Cluster A	Cluster B	Cluster C
MaxSlots	7	50	3
MaxJobs	7	5	3
MaxStarters	7	94	3

Abbildung 21: Auslastung mit der alten Policy

7.1.2 Der neue Algorithmus

Die neue Policy benötigt folgende Attribute (Die Bedeutung der Begriffe Klasse und Slots wird in Kapitel 3.4 erklärt.):

- **freeSlots**
  gibt an, wieviele freie Slots der Cluster für eine bestimmte Klasse hat.
- **maxSlots**
  ist eine Konstante und gibt an, wieviele Slots der Cluster für eine konkrete Klasse besitzt. Der Wert ist die Summe aller Slots aller Knoten einer Klasse innerhalb des Clusters.
- **jobWait**
  gibt die Anzahl der wartenden Jobs einer Klasse in der Warteschlange des Clusters an.
- **freeStarter**
  gibt die Anzahl der Jobs an, die noch auf dem Cluster sofort ausgeführt werden dürfen. Dabei beschränkt sich dieser Wert nicht auf eine bestimmte Klasse, sondern bezieht sich auf sämtliche Jobs des LoadLevelers.
- **maxStarter**
  gibt die Anzahl sämtlicher Jobs an, die gleichzeitig auf dem Cluster ausgeführt werden dürfen. Dies ist ein globaler Wert und nicht auf eine Klasse

beschränkt. Diese Konstante wird in der Konfigurationsdatei des LoadLevelers festgelegt und kann als Maß für die Leistungsfähigkeit einer Ressource angesehen werden.

- **maxjobs**
  gibt die Anzahl der Jobs eines Benutzers an, die gleichzeitig auf dem Cluster ausgeführt werden dürfen. Diese Konstante wird in der Konfigurationsdatei des LoadLevelers festgelegt.

Für die Funktionsweise der Policy benötigt der Grid-Scheduler in dem Cluster eine eigene LoadLeveler-Klasse und einen eigenen Benutzer, weil ansonsten die oben aufgeführten Attribute nicht sinnvoll ausgewertet werden können.

Damit ein Job vom LoadLeveler sofort ausgeführt werden kann, müssen folgende Bedingungen erfüllt sein:

1. $freeSlots > 0$
   Es dürfen noch Jobs der entsprechende Klasse ausgeführt werden.
2. $freeStarter > 0$
   Es dürfen noch Jobs auf einer Ressource gestartet werden.
3. $maxSlots - freeSlots < maxjobs$
   Die max. Anzahl an Jobs, die ein Benutzer bzw. der Grid-Scheduler auf einer Ressource ausführen darf, ist noch nicht erreicht.

Die Anzahl der Jobs, die der Grid-Scheduler zusätzlich auf einer Ressource starten darf, wird durch die drei Bedingungen begrenzt und berechnet sich folgendermaßen:

Anzahl laufender Jobs des Grid-Schedulers:

$$running = maxslots - freeslots$$

Anzahl der Jobs, die der Grid-Scheduler noch starten darf:

$$freejobs = maxJobs - running$$

freie Kapazität einer Ressource: max. Anzahl noch zu startender Jobs:

$$minFree = min(freestarter, freeslots, freejobs)$$

Die Anzahl der Jobs, die maximal von dem Grid-Scheduler gleichzeitig auf einer Ressource gestartet werden darf, ist durch die Attribute $maxstarter$, $maxslots$ und $maxjobs$ begrenzt:

Kapazität einer Ressource: maximale Anzahl parallel laufender Jobs:

$$minMax = min(maxstarter, maxslots, maxjobs)$$

Auslastung der Ressource aus der Sicht des LoadLevelers:

$$Auslastung = 1 - \frac{freestarters}{maxstarters}$$

Auslastung der Ressource aus der Sicht des Grid-Schedulers:

$$Auslastung = 1 - \frac{minFree}{minMax}$$

Auslastung der Ressource nach der Übergabe eines Jobs aus der Sicht des Grid-Schedulers für $minFree > 0$:

$$Auslastung = 1 - \frac{minFree - 1}{minMax}$$

Der Algorithmus ist von der Struktur her wie der ursprüngliche Algorithmus (siehe Kapitel 3.4) aufgebaut und besteht aus zwei Phasen:

1. Phase:
   Die ersten Phase wird ausgeführt, falls eine Ressource freie Kapazität für die Ausführung eines Jobs hat, also $minFree > 0$ ist. Es wird diejenige Ressource ausgewählt, welche am wenigsten nach der Übergabe eines Jobs ausgelastet ist. Dabei wird die Auslastung der Ressourcen nach der Übergabe eines Jobs aus der Sicht des Grid-Schedulers verwendet, damit die Kapazität $minMax$ die eine Ressource dem Grid-Scheduler zur Verfügung stellt, von allen Ressourcen zu gleichen Teilen verwendet wird. Also:

   $$Auslastung = 1 - \frac{minFree - 1}{minMax}$$

   Hier ist ein großer Unterschied zu dem ursprünglichen Algorithmus, weil der alte Algorithmus die Auslastung der Ressourcen vor der Übergabe und nicht wie jetzt nach einer möglichen Übergabe betrachtet. Das Ziel ist es, die Ressourcen gleichmäßig auszulasten. Angenommen es gibt zwei Ressourcen. Ressource A hat eine max. Kapazität von 100 und B eine max. Kapazität von 2. Auf A laufen 51 Jobs und auf B 1 Job. Somit ist A zu 51% und B zu 50% ausgelastet. Nach dem alten Algorithmus würde der nächste Job auf B ausgeführt, weil diese Ressource weniger ausgelastet ist. Danach wäre A zu 51% und B zu 100% ausgelastet. Nach dem neuen Algorithmus wird der Job auf A ausgeführt. Und somit wäre A zu 52% und B zu 50% ausgelastet, was einer gleichmäßigeren Verteilung entspricht.

   Falls mehrere Ressourcen die gleiche Auslastung haben, wird die Ressource mit der größeren Kapazität $minMax$ verwendet, weil im Verhältnis diese Ressource am wenigsten durch den Job belastet wird.

2. Phase:
   Die zweite Phase ist nur von Bedeutung, falls keine Ressource freie Kapazität hat. In diesem Fall wählt die Policy diejenige Ressource aus, deren Warteschlange kleiner ist, als die maximal zulässige Warteschlangenlänge. Die zweite Phase ist mit der des alten Algorithmuses aus Kapitel 3.4 identisch, nur dass jetzt auch die Auslastung der Warteschlangenlänge nach einer möglichen Übergabe eines Jobs betrachtet wird (vgl. erste Phase).

```
      tmpQuot = (float)(jobwait + 1)/(minMax*slotFactor);
```

Der Unterschied der Verteilung der beiden Algorithmen kann in Abbildung 21 und 22 nach der Übergabe von 10 Jobs betrachtet werden.

Die neue Policy muss die beiden Methoden getScheduleAttr und getQualifiedResource aus Kapitel 6.3.2 implementieren.

```
Resource getQualifiedResource(Vector resources) {

  Resource resource, bestHost, bestHostOld;
  float bestHostQuot = 0;
  float bestHostQuotOld = 2;
  int bestHostStarter = 0;
  int slotFactor = 5;
  int freeslots, jobwait, freestarters;
  int maxslots, maxstarters, maxjobs;
  int minFree, minMax;
  float tmpQuot;

  Iterator hostIt = resources.iterator();

  while (hostIt.hasNext()) {
    resource = (Resource)hostIt.next();

    freeslots = resource.getAttributeValue("queue-freeslots");
    maxslots  = resource.getAttributeValue("queue-maxslots");
    jobwait   = resource.getAttributeValue("queue-jobwait");
    freestarters = resource.getAttributeValue("queue-freestarters");
    maxstarters  = resource.getAttributeValue("queue-maxstarters");
    maxjobs = resource.getAttributeValue("queue-maxjobs");

    minFree = Math.min(Math.min(freestarters, freeslots),
                       maxjobs-(maxslots-freeslots));
    minMax = Math.min(Math.min(maxstarters, maxslots), maxjobs);

    if (minFree > 0 && jobwait == 0) {
```

– Start 1. Phase ——————————————-

```
      tmpQuot = (float)(minFree-1)/minMax;
      if ((tmpQuot > bestHostQuot) ||
        ((tmpQuot == bestHostQuot) &&
        (maxstarters > bestHostStarter))) {
        bestHostQuot = tmpQuot;
        bestHostStarter = maxstarters;
        bestHost = resource;
      }
```

– Ende 1. Phase ——————————————-

```
    } else {
```

– Start 2. Phase ————————————————-

```
      if ((bestHost == null) &&
          (jobwait < maxslots * slotFactor)) {
        tmpQuot = (float)(jobwait + 1)/(minMax*slotFactor);
        if (tmpQuot < bestHostQuotOld) {
          bestHostQuotOld = tmpQuot;
          bestHostOld = resource;
        }
      }
```

– Ende 2. Phase ————————————————-

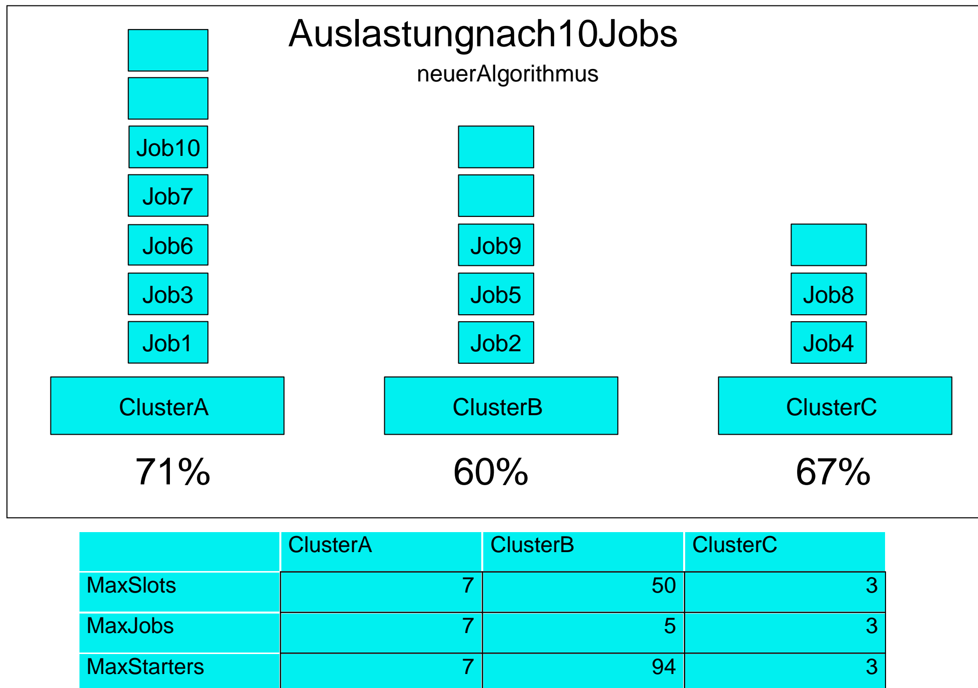
```
    }
  } //end while

  if (bestHost == null) {
    if (bestHostOld != null) {
    bestHost = bestHostOld;
    } else {
    ...error no free host found
    }
  } //end if

  return bestHost;
}
```

7.1.3 Vergleich der beiden Policies

Der Durchsatz und die Antwortzeit dürfte bei sieben oder weniger Jobs bei dem alten und neuen Algorithmus in etwa gleich sein, weil jeder Job sofort auf einem Prozessor ausgeführt wird. Die Verteilung ist wie in Abbildung 21 und 22 dargestellt sehr unterschiedlich. Ab dem achten Job ist der Durchsatz und die Antwortzeit mit dem neuen Algorithmus besser, weil die nächsten Jobs beim alten Algorithmus in die Warteschlange von Cluster B kommen und beim neuen Algorithmus auf einem Prozessor einer Ressource zugeteilt werden.

Werden jedoch immer soviele Jobs in das Grid übergeben, dass jeder Cluster ausgelastet ist, ist der Durchsatz bei dem alten und dem neuen Algorithmus gleich. Nur die Antwortzeit variiert bei dem neuen Algorithmus nicht so stark, da die Jobs gleichmäßiger verteilt werden.

Mit dieser neuen Policy wird die Anforderung für die Entwicklung einer verbesserten Policy erfüllt. Diese Policy löst das Problem, dass die Ressourcen ungleichmäßig ausgelastet werden. In Kapitel 8 wird das Verhalten der beiden Policies untersucht.



	ClusterA	ClusterB	ClusterC
MaxSlots	7	50	3
MaxJobs	7	5	3
MaxStarters	7	94	3

Abbildung 22: Auslastung mit der neuen Policy

7.2 GRAM-Adapter

Der GRAM-Adapter bildet die Verbindung zwischen Framework und Grid-Middleware. Er implementiert die Schnittstelle Resource-Interface aus Kapitel 6.3.3 und hat die Aufgabe, einen Job an die ausgewählte Ressource zu übergeben. Der Adapter ist für das Globus Toolkit 2.0 erstellt. Falls eine andere Grid-Middleware eingesetzt werden soll, muss dieser Adapter ausgetauscht werden.

Der GRAM-Adapter implementiert die beiden Funktionen *executeJob* und *cancelJob*.

- executeJob(Job job, Resource resource)
  Zu Beginn wird überprüft, ob es für den Benutzer ein gültiges Proxy-Objekt für die Kommunikation mit den Komponenten des Globus Toolkits gibt (siehe Abbildung 9). Falls es kein Proxy-Objekt gibt, wird ein neues Proxy-Objekt erstellt. Mit dem Proxy-Objekt und der RSL (siehe Kapitel 5.4.1) wird ein Globus-spezifisches Job-Objekt erstellt und bis zu dreimal versucht, an den Gatekeeper (siehe Abbildung 9) bzw. die Ressource zu übermitteln. Nach der Übermittlung wird dem Job-Objekt des Frameworks die Globus-spezifische Job-URL, unter der dieser Job innerhalb der Grid-Middleware identifiziert werden kann, gesetzt.

- cancelJob(Job job)
  Um einen Job auf einer Ressource abbrechen zu können, wird ein Proxy-Objekt (siehe Abbildung 9) erstellt. Anhand der Globus-spezifischen Job-

URL von der Übergabe des Jobs kann der Job identifiziert und abgebrochen werden.

7.3 MDS-Adapter

Der MDS-Adapter implementiert die Schnittstelle InfoService-Interface (vergleiche Kapitel 6.3.1). Sie dient zur Abfrage der Informationen über zur Verfügung stehende Ressource von dem zentralen Informationssystem. Der Adapter ist für das Globus Toolkit 2.0 ausgelegt und fragt dessen zentrales Informationssystem (MDS, siehe Abbildung 9) ab.

Er implementiert den Konstruktor *MdsAdapter* und die beiden Methoden *reload* und *getResources*.

- MdsAdapter(String propertiesFile, String mappingFile)
  Mit dem Konstruktor wird eine Instanz der Klasse MdsAdapter erzeugt und ruft die Methode *reload* auf.

- reload(String propertiesFile, String mappingFile)
  Diese Methode liest die Konfigurationsdatei und das Mapping-File ein.

- getResources(Vector requirements, Vector scheduleAttr)
  Die Methode *getResources* erstellt mit den Jobanforderungen und den benötigten Attributen für die Policy eine LDAP-Anfrage. Diese wird an das MDS geschickt. Antwortet das MDS nicht, wird das Backup-MDS abgefragt. Aus der Antwort vom MDS wird eine Menge an Objekten der Klasse Ressource erzeugt, und die Methode gibt diese in einem Objekt der Klasse Vector zurück.

8 Evaluierung

Die Evaluierung ist die Bewertung von einem abgeschlossenen Vorhaben. In diesem Fall soll der entstandene Scheduler bewertet werden. Er besteht aus dem Framework, den Policies und den Globus-spezifischen Implementierungen der Schnittstellen.

„Die Vorraussetzung für eine Evaluierung ist eine zielorientierte Planung von sowohl quantitativ messbaren Kennzahlen oder Indikatoren als auch qualitativ verbal dargestellter Absichtserklärungen.“[TU Wien, 2003]

Die folgenden zwei Kapitel 8.1 und 8.2 untersuchen das Verhalten der beiden Policies und den Aufwand, der für die Integration von heterogenen Systemen in das Grid nötig ist.

Bei der folgenden Evaluierung wird davon ausgegangen, dass der Grid-Scheduler in eine eigene LoadLeveler-Klasse seine Jobs übergeben kann und die Jobs unter einem eigenen Benutzer ausführt.

8.1 Evaluierung der Policies

In diesem Kapitel wird evaluiert, inwieweit sich die ursprüngliche Policy aus Kapitel 3.4 von der neuen Policy aus Kapitel 7.1.2 unterscheidet. Dabei liegt das Hauptaugenmerk auf der gleichmäßigen Auslastung der Ressourcen, weil dies der Grund für die Neuentwicklung der Policy war. Weiterhin wird die durchschnittliche Antwortzeit, von der Übergabe eines Jobs bis zum Ende seiner Ausführung, betrachtet.

Im ersten Schritt werden Metriken zu der Auslastung der Ressourcen und zu der Jobausführung vorgestellt. Im nächsten Schritt wird die Durchführung der Tests beschrieben und die Ergebnisse der Durchführung aufgelistet und diskutiert.

Damit die Metriken über die Ausführungszeit eines Jobs erhoben werden können, müssen diese erfasst werden. Weiterhin muss der Zustand eines Jobs feststellbar sein. Die Erfassung des Zustands war die Aufgabe des Housekeepers (siehe Kapitel 3.2.3).

Als mögliche Lösung für die Probleme mit dem Housekeeper bietet das LoadLeveler API die Möglichkeit an, einem Job bei seiner Ausführungsübergabe ein sog. Monitorprogramm mitzugeben. Es wurde das Monitorprogramm entwickelt, welches bei jeder Änderung des Jobstatus von LoadLeveler im AFS aufgerufen wird. Dieses Monitorprogramm schreibt den neuen Jobstatus von dem jeweiligen Knoten, auf dem der Job ausgeführt wird, in die Scheduler-Datenbank. Vorteile dieser Callback-Lösung sind, dass der Housekeeper nicht mehr benötigt wird, das MDS dadurch entlastet wird, und für jeden Job der aktuelle Status zu jedem Zeitpunkt von der Scheduler-Datenbank abgefragt werden kann. Der Nachteil ist, dass die Lösung nicht skaliert. Dieses Problem besteht bei dem Scheduler auch dadurch, dass es ein zentraler Scheduler ist. In Abbildung 23 ist die Callback-Lösung dargestellt.

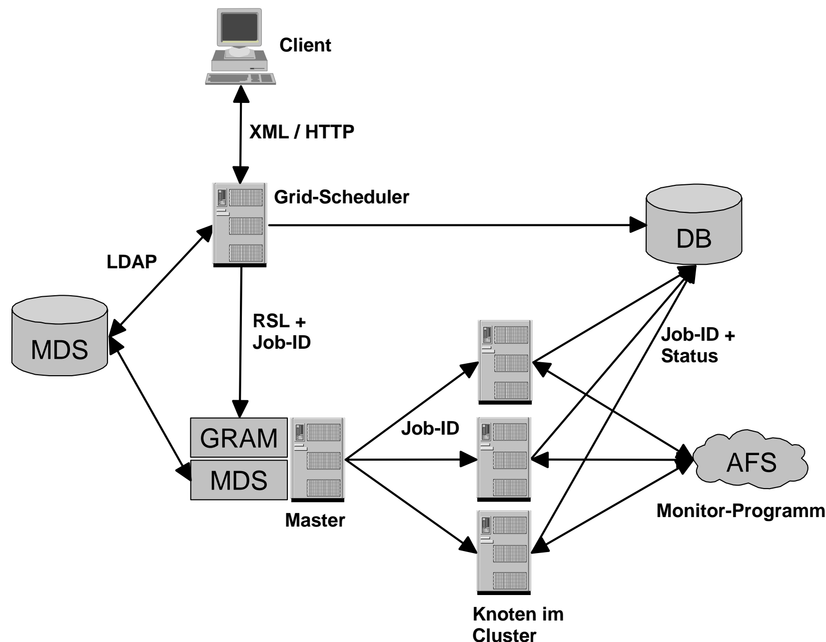


Abbildung 23: Die Callback-Lösung

8.1.1 Auslastung

Die Metriken in diesem Kapitel wurden von den LoadLeveler-spezifischen Attributen aus dem Kapitel 7.1.2 und den vier Zeitpunkten von Jobstatusänderungen in der Scheduler-DB (siehe Abbildung 24) abgeleitet.

Die Bedeutung der vier Zeitpunkte:

- **SubmitStartTime**
  Sobald der Endbenutzer dem Grid-Scheduler einen Job übergibt, wird der Zeitpunkt zu dem jeweiligen Job in der Scheduler-DB festgehalten.

- **SubmitEndTime**
  Wenn der Grid-Scheduler den Job erfolgreich an eine Ressource übergeben hat, wird der Zeitpunkt in die Scheduler-DB geschrieben.

- **CLDispatchTime**
  Dieser Zeitpunkt gibt an, wann der lokale Scheduler den Job aus seiner Warteschlange nimmt und auf einem Knoten in seinem Cluster ausführt. Ein Monitorprogramm, welches von dem lokalen Scheduler bei jeder Statusänderung des Jobs aufgerufen wird, schreibt den Zeitpunkt in die Scheduler-DB. Das Monitorprogramm wird von dem Knoten aus gestartet, auf welchem der Job ausgeführt wird.

- **CLEndTime**
  CLEndTime gibt den Zeitpunkt an, wann ein Job abgearbeitet wurde. Der

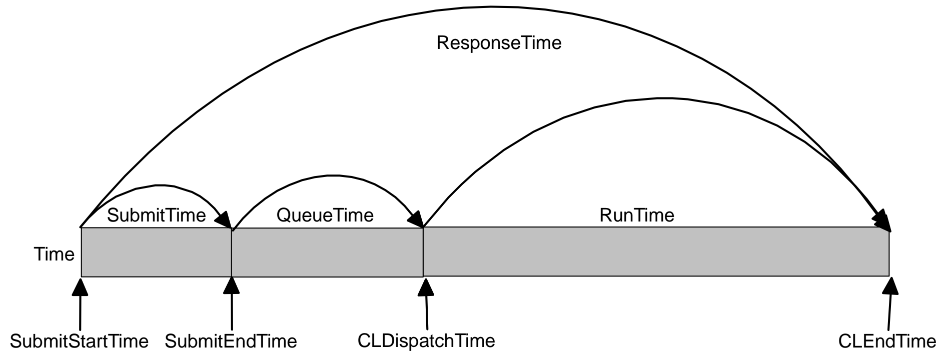


Abbildung 24: Zeitpunkte während der Jobausführung

Eintrag des Zeitpunkts erfolgt in die Scheduler-DB wie bei der CLDispatch-Time.

Die Auslastung gibt das Verhältnis an, zu welchem Grad die Leistung einer Ressource zu der Gesamtleistung in Anspruch genommen wird. Die Auslastung der Ressourcen (z.B. Cluster) setzt sich zusammen aus der Auslastung, die durch die Ausführung von Jobs durch den Grid-Scheduler und direkt vom Endbenutzer veranlasst wurde.

- **Auslastung der Ressourcen**
  Die Auslastung der Ressourcen ist aus der Sicht des lokalen Schedulers dargestellt und unabhängig von der Kapazität, die dem Grid zur Verfügung gestellt wird:

$$loadResource = 1 - \frac{\sum_{i=1}^{\#resources} freeStarters_i}{\sum_{i=1}^{\#resources} maxStarters_i}$$

- **Auslastung des Grids**
  Die Auslastung des Grids setzt sich aus der freien Kapazität und der maximal zur Verfügung stehenden Kapazität des Grids zusammen:

$$freeCapacity = \sum_{i=1}^{\#resources} \min(freeSlots_i, freeStarters_i, freeJobs_i)$$

$$maxCapacity = \sum_{i=1}^{\#resources} \min(maxSlots_i, maxStarters_i, maxJobs_i)$$

$$loadGrid = 1 - \frac{freeCapacity}{maxCapacity}$$

- **Auslastung durch andere Benutzer**
Die Auslastung durch andere Benutzer beschreibt, wie hoch die Auslastung einer Ressource durch Jobs ist, die direkt von dem Endbenutzer auf der Ressource ausgeführt werden. Diese Jobs wurden somit nicht von dem Grid-Scheduler an die Ressource übergeben.

$$loadOtherUser = \frac{usedStarter - (maxSlots - freeSlots)}{maxStarter}$$

8.1.2 Ausführungsdauer

Anhand den vier gemessenen Zeitpunkten aus Abbildung 24 können die folgenden Ausführungszeiten ermittelt werden:

- **Durchschnittliche Antwortzeit**
Die avgResponseTime gibt die durchschnittliche Dauer von der Jobübergabe des Endbenutzers bis zur kompletten Abarbeitung eines Jobs an.

$$avgResponseTime = \frac{\sum_{i=1}^{\#jobs} (CLEndTime_i - SubmitStartTime_i)}{\#jobs}$$

- **Durchschnittliche Zeit für die Übergabe eines Jobs an eine Ressource**
Die avgSubmitTime gibt die durchschnittliche Zeitspanne an, wie lange es von der Jobübergabe des Benutzers über die Auswahl der Ressource bis zur vollständigen Übergabe des Jobs an die Ressource dauert.

$$avgSubmitTime = \frac{\sum_{i=1}^{\#jobs} (SubmitEndTime_i - SubmitStartTime_i)}{\#jobs}$$

- **Durchschnittliche Wartezeit eines Jobs in der Queue des lokalen Schedulers**
Die avgQueueTime gibt die durchschnittliche Wartezeit eines Jobs in der Warteschlange des lokalen Schedulers an.

$$avgQueueTime = \frac{\sum_{i=1}^{\#jobs} (SubmitEndTime_i - CLDispatchTime_i)}{\#jobs}$$

- **Durchschnittliche Laufzeit eines Jobs auf einer Ressource**
  Die avgRunTime gibt die durchschnittliche Dauer für die Ausführung eines Jobs auf einer Ressource an.

$$avgRunTime = \frac{\sum_{i=1}^{\#jobs} (CLEndTime_i - CLDispatchTime_i)}{\#jobs}$$

8.1.3 Durchsatz und Wirkungsgrad

- **Durchsatz des Grids**
  Der Durchsatz beschreibt, wieviele Jobs in einer bestimmten Zeit von dem Grid vollständig abgearbeitet werden können.

$$performance = \frac{\#jobs}{CLEndTime_{\#jobs} - SubmitStartTime_1}$$

- **Wirkungsgrad des Grids**
  Der Wirkungsgrad des Grids gibt an, wie Effizient das Grid im Vergleich zur Ausführung der Jobs direkt auf einer Ressource durch den Endbenutzer arbeitet.

$$efficiency = \frac{avgRunTime}{avgResponseTime}$$

8.1.4 Durchführung

Für die Durchführung der Performance-Tests steht das LabGrid der IBM zur Verfügung. Es soll untersucht werden, inwieweit sich die alte von der neuen Policy unterscheidet.

Die Testumgebung besteht für den Test aus drei Clustern, auf denen das Betriebssystem AIX und die Clustersoftware LoadLeveler installiert sind. Als Job für die Tests wird der Simulationsjob für die Hardwareentwicklung verwendet.

Die Auslastung der Ressourcen wird von einem kleinen Programm, welches alle 20 Sekunden das zentrale Informationssystem MDS nach den Ressourcen abfragt, erfasst. Die Ergebnisse der Anfrage werden in einer Textdatei im CSV-Format abgelegt. Damit können die Daten einfach von einem Tabellenkalkulationsprogramm eingelesen und weiterverarbeitet werden.

Es wird die alte und neue Policy mit jeweils kurzen und langen Simulationsjobs, welche fünfmal länger als die kurzen Jobs laufen, untersucht. Dazu wird dem Grid-Scheduler bei jedem Testlauf eine Stunde lang sequentiell Jobs zur Ausführung übergeben. Bei den kurzen Jobs wurde keine künstliche Verzögerung zwischen der Übergabe der Jobs eingesetzt. Für die Testläufe mit den langen Jobs wurde eine Verzögerung von 30 Sekunden eingesetzt, damit die Warteschlange der Ressourcen nicht so schnell volllaufen. Bei jedem Testlauf werden nach einer Stunde keine Jobs mehr an den Scheduler übergeben und gewartet, bis alle Jobs berechnet sind.

Das Monitorprogramm, welches die Zeiten über die Änderungen des Jobstatus in die Scheduler-DB schreibt, darf auf dem Cluster B nicht installiert werden, weil dieser Cluster zu der Produktionsumgebung anderer Abteilungen der IBM gehört. Ein kurzzeitiger Ausfall dieses Clusters würde sehr hohe Ausfallkosten verursachen. Die Folge davon ist, dass die Laufzeiten der Jobs auf dem Cluster B nicht erfasst werden können. Somit können einige Metriken, die die Zeitpunkte über die Änderungen des Jobstatus benötigen, nicht erhoben werden.

Die Durchführungsdauer für einen Testlauf wurde bei der Planung auf 24 Stunden festgelegt. Diese Durchführungsdauer erwies sich aber im Laufe mehrerer Testläufe als nicht durchführbar. Dies hat folgende Gründe:

- Die Cluster gehören unterschiedlichen Abteilungen, und dadurch kommt es öfters vor, dass sich die Konfiguration der Cluster unerwartet ändert.
- Der Mitarbeiter, welcher für die Simulationsjob verantwortlich ist, muss in gewissen Zeitabständen die Ergebnisse der Simulationsjobs weiterverarbeiten, weil sonst die Kapazität der Fileserver nicht ausreicht. Dieser Vorgang ist leider nur teilautomatisiert. Das Simulationsmodell liefert je nach Simulationsfall unterschiedlich große Mengen an Ausgabedaten. Im normalen Betrieb werden die Jobs mit einer größeren Verzögerung zwischen zwei Jobs an den Grid-Scheduler übergeben.
- Desweiteren kommt es vor, dass das zentrale Informationssystem keine Daten zurückliefert und neu gestartet werden muss, oder dass temporäre Dateien von Globus auf dem Masterknoten nicht mehr automatisch gelöscht werden sondern manuell gelöscht werden müssen. Das sind einige der Probleme mit dem Globus Toolkit 2.0.

Damit die Ergebnisse vergleichbar sind, wurde die Ausführungsdauer eines Testlaufs auf eine Stunde begrenzt. Die Testläufe mussten öfters wiederholt werden, bis ein Testlauf ohne Probleme durchlief. Während der Testläufe sind jedoch keine Probleme am Scheduler aufgetreten.

8.1.5 Ergebnisse

Die Abbildungen 25 und 26 zeigen die Ergebnisse der Testläufe mit einer Dauer von einer Stunde.

Die grauen Balken stellen die Auslastung einer Ressource aus der Sicht des Grid-Schedulers in Prozent dar. Es wird nicht die eigentliche CPU-Auslastung sondern

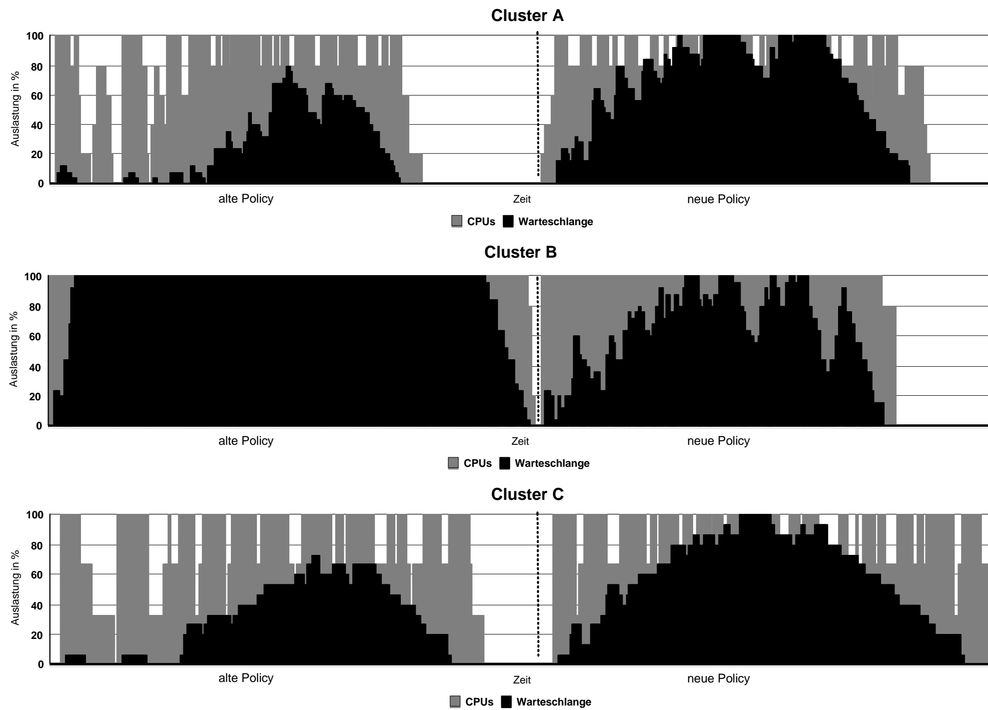


Abbildung 25: Auslastung der Ressourcen mit kurzen Jobs (aus der Sicht des Grid-Schedulers)

die Belegung der CPUs auf einer Ressource angezeigt. Die schwarzen Balken zeigen die Auslastung der Warteschlange zu ihrer maximalen Warteschlangenlänge. Die Auslastung einer Ressource wurde aus der Sicht des Grid-Schedulers, wie in Kapitel 7.1.2, erfasst. Die maximale Warteschlangenlänge wurde wie in der neuen Policy mit *maxCapacity * slotFactor* angenommen. Der Scheduler mit der alten Policy kann jedoch mehr Jobs in die Warteschlange stellen, weil in der alten Policy die maximale Warteschlangenlänge mit *maxslots * slotFactor* angegeben ist.

Die Diagramme bestätigen die theoretische Betrachtung über das Verhalten der beiden Policies aus Abbildung 21 und 22:

Die alte Policy übergibt die Mehrzahl der Jobs in die Warteschlange von Cluster B, obwohl Cluster A und C nicht vollständig ausgelastet sind. In der theoretischen Betrachtung wird an Cluster A und C jeweils ein Job übergeben, bis die Warteschlange von Cluster B voll ist. Bei dem Testlauf bekommen Cluster A und C vereinzelt mehr Jobs, was bei den beiden Testläufen der kurzen Jobs in Abbildung 25 auffällt. Das kommt daher, dass das MDS nur alle 30 Sekunden aktualisiert wird, und somit ist der Cluster A oder C für diese Zeitspanne die optimale Ressource. Bei einer durchschnittlichen Übertragungszeit von ca. 5 Sekunden vom Grid-Scheduler an die Ressource werden somit rund 6 Jobs an Cluster A oder C in diesem Zeitraum übergeben.

Es ist ersichtlich, dass die neue Policy weitaus gleichmäßiger die Ressourcen auslastet als die alte Policy. Die durchschnittliche Auslastung der Ressourcen aus der Sicht des Grid-Schedulers während der vier Testläufe ist in den folgenden zwei

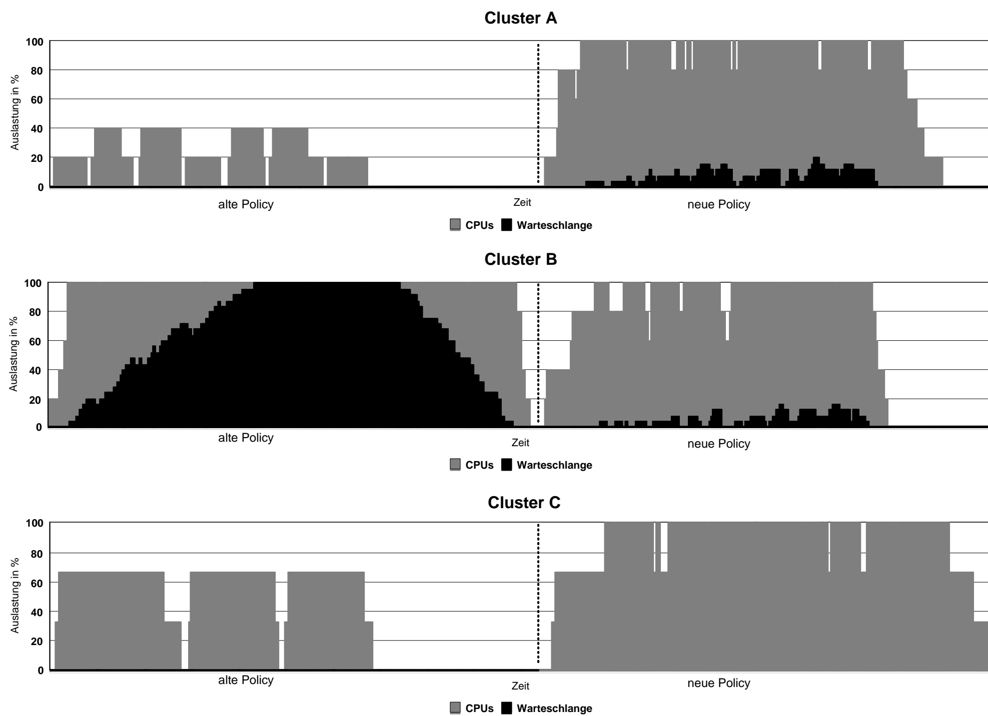


Abbildung 26: Auslastung der Ressourcen mit langen Jobs (aus der Sicht des Grid-Schedulers)

Tabellen dargestellt:

	Policy	Cluster A	Cluster B	Cluster C
Kurze Jobs:	alt	60% (95 Jobs)	98% (281 Jobs)	69% (37 Jobs)
	neu	73% (106 Jobs)	77% (203 Jobs)	85% (39 Jobs)

	Policy	Cluster A	Cluster B	Cluster C
Lange Jobs:	alt	18% (11 Jobs)	96% (88 Jobs)	41% (6 Jobs)
	neu	72% (36 Jobs)	65% (56 Jobs)	80% (13 Jobs)

Es wurde angenommen, dass die Jobs mit der neuen Policy schneller berechnet werden, weil die Ressourcen gleichmäßiger ausgelastet werden, und die Jobs, wie z.B. bei Cluster B, nicht so lange in der Warteschlange warten müssen. Die Testläufe haben jedoch gezeigt, dass diese Annahme nicht immer zutrifft. Bei den beiden Testläufen mit den kurzen Simulationsjobs wurden die Jobs mit der alten Policy schneller berechnet, als mit der neuen Policy. Die Ursache dafür ist, dass Cluster B im Gegensatz zu Cluster C nur ein Viertel der Zeit zur Berechnung eines Jobs braucht. Auf die Lösung dieses Problems wird in Kapitel 8.1.6 eingegangen.

Abbildung 27 und 28 zeigen die Auslastung des Grids und die Auslastung der Ressourcen aus der Sicht von LoadLeveler während der Testläufe.

Das obere Diagramm der Abbildung 27 zeigt, dass zu Beginn des Testlaufs mit den kurzen Jobs und der alten Policy das Grid relativ ungleichmäßig ausgelastet war. Der Grund dafür ist, dass der Großteil der Jobs zuerst auf den Cluster B übergeben wird und danach die beiden anderen Cluster bedient werden. In dem unteren

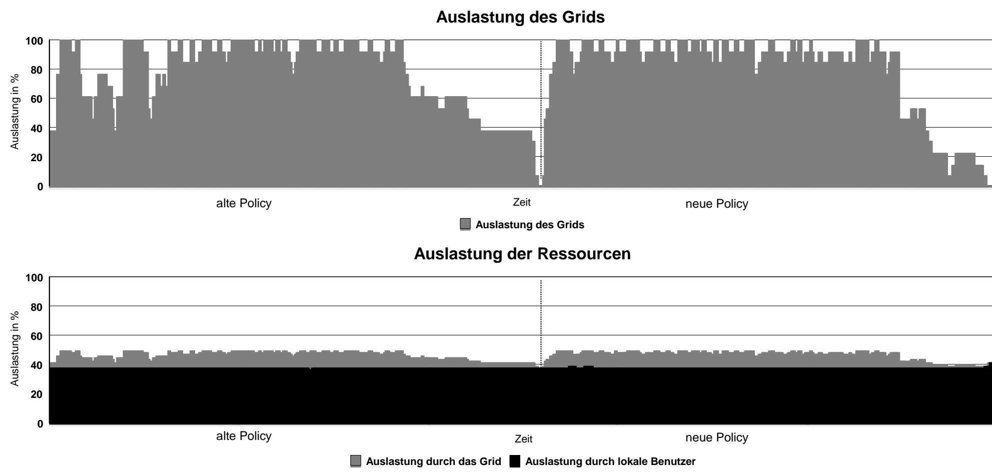


Abbildung 27: Auslastung des Grids mit kurzen Jobs

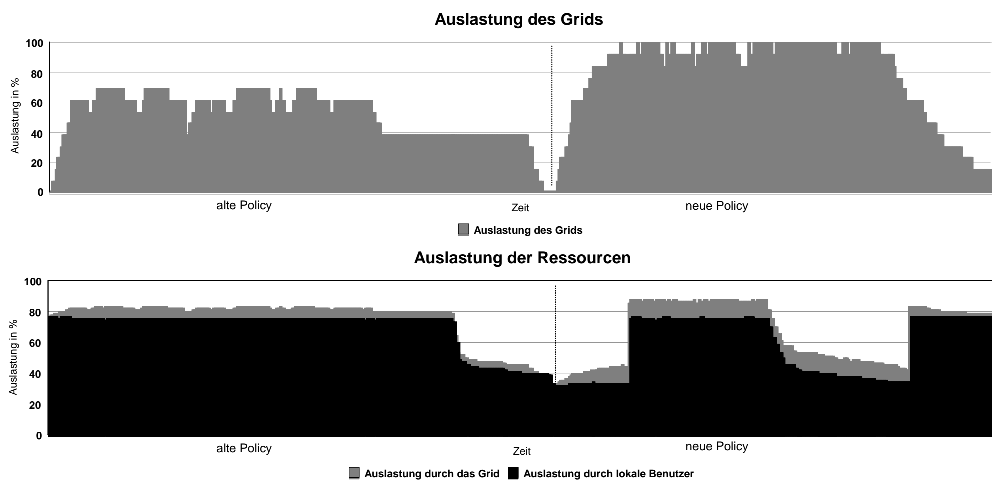


Abbildung 28: Auslastung des Grids mit langen Jobs

Diagramm sieht man, dass die Auslastung durch lokale Benutzer der Ressourcen konstant bei ca. 40% lag.

Abbildung 28 stellt die Testläufe mit den langen Jobs dar. In dem oberen Diagramm wird deutlich, dass die neue Policy das Grid bzw. die dafür bereitgestellten Ressourcen zu 100% nutzen kann. Die alte Policy dagegen nutzt das Grid maximal nur zu 69%. Das untere Diagramm zeigt, dass die Last auf den Ressourcen durch die lokalen Benutzer zwischen 33% und 77% schwankt.

Am Ende der Testläufe wurde die Installation des Monitorprogramms auf Cluster B genehmigt. Dadurch ist es möglich die fehlenden Metriken zu erfassen.

Zwei weitere Testläufe mit einer Dauer von je drei Stunden sollen untersuchen, wie sich die durchschnittliche Antwortzeit der Jobs bei den beiden Scheduling-Policies verhält. Bei der IBM ist der Scheduler rund um die Uhr im Einsatz, und es wird mit einer konstanten Rate Jobs an den Grid-Scheduler übergeben. Aus diesem Grund wird für die Untersuchung der beiden Policies ein Zeitraum ausgewertet, wo, im Gegensatz zu den vorherigen Testläufen, vor und nach dem Zeitraum mit einer konstanten Rate Jobs an den Grid-Scheduler übergeben werden. Die folgende Tabelle zeigt den Vergleich der Metriken der beiden Policies:

Metrik	alte Policy	neue Policy	Verbesserung
Anzahl der Jobs	294	297	
Jobs Cluster A	30 (10%)	82 (28%)	
Jobs Cluster B	252 (86%)	202 (68%)	
Jobs Cluster C	12 (4%)	13 (4%)	
responseTime all jobs	196,5min	184,8min	
performance	90 Jobs/h	96 Jobs/h	+ 7%
efficiency	42,5%	88,9%	+ 109%
avgResponseTime	659s	413s	+ 37%
minResponseTime	17s	9s	
maxResponseTime	1608s	1808s	
avgSubmitTime	5s	6s	- 20%
minSubmitTime	3s	3s	
maxSubmitTime	161s	160s	
avgQueueTime	373s	40s	+ 89%
minQueueTime	0s	0s	
maxQueueTime	758s	371s	
avgRunTime	280s	367s	- 31%
minRunTime	5s	5s	
maxRunTime	1586s	1660s	

Die Messergebnisse zeigen, dass die neue Policy die Ressourcen besser ausnützt und daher den Durchsatz auf 96 Jobs pro Stunde steigern kann. Gleichzeitig wird der Wirkungsgrad des Grids um 109% auf 88,9% gesteigert. Die durchschnittliche Antwortzeit verringert sich um 37%, und die durchschnittliche Wartezeit eines Jobs verringert sich um 89%. Die durchschnittliche Laufzeit eines Jobs und die durchschnittliche Dauer für die Übergabe eines Jobs an die Ressourcen haben sich erhöht, weil durch die neue Policy mehr Jobs auf den langsameren Clustern A und C ausgeführt wurden.

Die Laufzeit der Jobs liegt zwischen 5 Sekunden und 1660 Sekunden. Um eine Aussage treffen zu können, wie die Verteilung der Laufzeiten der einzelnen Jobs ist, wurde ermittelt, dass bei dem Testlauf mit der alten Policy bei 94% der Jobs und bei der neuen Policy bei 91% der Jobs auf Cluster B die Laufzeit zwischen 150 Sekunden und 300 Sekunden lag. Dies zeigt, dass die Varianz der Laufzeit der Jobs nicht sehr groß ist und das Ergebnis dadurch nicht verfälschen sollte.

Keine der Ressourcen aus der Sicht von LoadLeveler war zu irgendeinem Zeitpunkt der Testläufe zu 100% ausgelastet und hätte dadurch den Test beeinflusst.

8.1.6 Diskussion

Die Testläufe zeigen eindeutig, dass die neue Policy die Ressourcen gleichmäßiger nutzt. Die Testläufe haben jedoch ein anderes Problem aufgedeckt, welches bisher vernachlässigt wurde. Die Jobs wurden auf dem Cluster B viermal schneller ausgeführt als auf dem Cluster C und doppelt so schnell wie auf dem Cluster A.

Für die optimale Nutzung der Ressourcen ist es wichtig, dass die Leistungsfähigkeit im Bezug auf die Prozessorleistung berücksichtigt wird. Das kann folgendermaßen erreicht werden:

- Es wird ein weiteres Attribut in dem zentralen Informationssystem eingeführt, welches die Leistungsfähigkeit einer Ressource für einen Job in der Form eines Benchmark-Wertes abbildet. Jedoch muss man sich dafür auf ein bestimmtes Verfahren einigen, um diesen Wert zu ermitteln. Für heterogene Ressourcen ist solch ein Wert jedoch zu ungenau, da unterschiedliche Arten von Ressourcen unterschiedliche Metriken für die Abbildung ihrer Leistungsfähigkeit benötigen.
- Eine weitere Möglichkeit wäre, dass die Policy die Ausführungszeit aus vorangegangenen gleichartigen Jobs über die Scheduler-DB ermittelt und damit für jede Ressource die Laufzeit eines Jobs schätzen kann. Dadurch können die Jobs mit dieser Heuristik im Hinblick auf die Gesamtausführungszeit der Jobs mit der MCT (Minimum Completion Time) aus dem Kapitel 5.3 noch besser verteilt werden, und die Gesamtausführungszeit für eine bestimmte Anzahl von Jobs wäre geringer.

  Bei der neuen Policy ist in Abbildung 26 erkennbar, dass Cluster C wesentlich später als Cluster B mit der Berechnung der Jobs fertig ist. Mit einer Policy, die die Laufzeit der letzten n Jobs berücksichtigt, würden die Cluster A und B mehr Jobs bekommen, und alle Cluster wären etwa zu derselben Zeit mit der Berechnung der Jobs fertig.
- Eine zusätzliche Verbesserung der oben vorgeschlagenen Policy wäre ein Umschalten auf eine Batch-Modus Policy aus Kapitel 5.3, falls alle Ressourcen ausgelastet sind. Die Jobs würden nicht in periodischen Abständen oder bei der Ansammlung einer bestimmten Anzahl von Jobs an die Ressourcen übergeben werden, sondern würden erst, wenn eine Ressource freie Kapazitäten hat, an die Ressourcen übergeben. Dadurch kann die Gesamtbearbeitungszeit für eine Menge von Jobs reduziert oder auf den Ausfall einer Ressource schneller reagiert werden.

Die Testläufe hätten auch mit der Hilfe einer Simulation durchgeführt werden können. Jedoch ist es kaum möglich alle Eigenschaften eines Grids zu berücksichtigen. Viele Wissenschaftler haben bei der Entwicklung einer neuen Policy nur die Möglichkeit, die Policy neben der theoretischen Betrachtung mit der Hilfe eines Simulators zu evaluieren. Weil das Entwicklungslabor der IBM ein eigenes Grid in der Produktion betreibt, war es eine gute Möglichkeit, die Evaluierung unter realen Bedingungen durchzuführen.

Für die Simulation von eines Grids und für die Untersuchung von Scheduling-Algorithmen gibt es folgende Werkzeuge:

- MicroGrid ist von der University of California [Song, 2000].
- SimGrid ist von der University of California [Legrand, 2003].
- Bricks ist von der Ochanomizu University of Tokyo [Takefusa, 1999].
- GridSim ist von der University of Melbourne [Murshed, 2002].

8.2 Integration von heterogenen Ressourcen

Um die Flexibilität des Frameworks und den Aufwand eine neue Policy in den Scheduler zu integrieren, zu untersuchen, soll im Vergleich zu der vorherigen Evaluierung eine heterogene Ressource integriert werden.

8.2.1 Durchführung

Die Diplomarbeit eines Studenten der IBM [Luemkemann, 2003] hat sich mit der Einbindung von Linux auf einer IBM zSeries in das Grid befasst. Es werden verschiedene Linux Distributionen in unterschiedlichen Versionen von einem zSeries-Server dem Grid zur Verfügung gestellt. Sobald ein Job an den Server übergeben wird, wird die entsprechende Distribution gestartet und der Job ausgeführt. Es können parallel mehrere Distributionen gestartet werden, und innerhalb von einer Distribution können mehrere Jobs ausgeführt werden. Somit wird jede Distribution höchstens einmal gestartet. Wenn alle Jobs auf einer Distribution beendet sind, wird nach einer bestimmten Zeit die Distribution heruntergefahren.

Es wird eine neue Policy integriert, die anhand von den Attributen, die der zSeries-Server zur Verfügung stellt, die optimale Ressource auswählt, den Job übergibt und dieser auf der entsprechenden Distribution ausgeführt wird.

8.2.2 Ergebnisse

Die Erstellung der Policy und die Integration in das Framework war trivial. Die neue Policy wählt anhand des besseren Verhältnises von Warteschlangenlänge zu der maximalen Warteschlangenlänge die entsprechende Ressource aus. Zusätzlich müssen die benötigten Attribute der Ressource in das Mapping-File des MDS-Adapters eingetragen werden. Die Policy muss in der Datei Policies.xml eingetragen werden.

In der XML-Jobbeschreibungssprache müssen die Anforderungen und die Policy beschrieben werden:

```
<requirement name="distribution" value="SLES" required="true">
</requirement>

<requirement name="distribution-version" value="8" required="true">
</requirement>

<policy>com.ibm.de.boeblingen.IDEGrid.scheduler.FreeQueueScheduler
</policy>
```

Mit dieser Jobbeschreibung kann in diesem Fall ein Job auf einer Ressource, welche die benötigten Attribute der Policy liefert, ausgeführt werden. Die RSL, welche von dem Grid-Scheduler generiert wird, sieht folgendermaßen aus:

```
&( arguments = "60" )( executable = "/bin/sleep" )
( queue = "zLnx" )( environment = ( "jid" "7516" )
( "resourceproperties" "distribution:SLES%version:8" ))
```

8.2.3 Diskussion

Die Ergebnisse zeigen, dass eine neue Policy benötigt wird, da heterogene Systeme unterschiedliche Attribute haben, und eine Policy nicht mit beliebigen Attributen arbeiten kann, solange die Semantik der Attribute für die Policy unbekannt ist. Einen Benchmark-Wert einer Ressource für die Angabe ihrer Leistungsfähigkeit genügt auch nicht, da jeder Job andere Anforderungen an eine Ressource stellen kann. Falls es die Möglichkeit zur Messung der Laufzeit von Jobs gibt, kann mit der Hilfe einer Heuristik, wie in Kapitel 8.1.6 dargestellt, die optimale Ressource ermittelt werden. Dazu ist jedoch eine bestimmte Anzahl von gleichartigen Jobs nötig.

9 Zusammenfassung

9.1 Lösung der Probleme, Bewertung der Ergebnisse

Das neue Framework bietet mit der Matchmaking-Funktionalität die Möglichkeit einen Scheduler aufzubauen, um Ressourcen anhand von Jobanforderungen auszuwählen. Dadurch können heterogene Ressourcen in das Grid eingebunden werden und Jobs mit unterschiedlichen Anforderungen auf den entsprechenden Ressourcen ausgeführt werden.

Die konkrete Ressource für die Ausführung eines Jobs wird von einer Scheduling-Policy ausgewählt. Das Framework bietet eine Schnittstelle, um verschiedene Policies einzubinden. Damit können Policies einfach ausgetauscht und je nach Bedarf mit einem sehr geringen Aufwand neue Policies implementiert und eingebunden werden.

Neben der Schnittstelle für die Policies bietet das Framework noch eine Schnittstelle für die Anbindung eines Informationssystems und eine Schnittstelle, um die Ausführung von Jobs über die Grid-Middleware zu steuern.

Es wurde eine neue Policy entwickelt, weil die ursprüngliche Policy auf nicht optimalen Annahmen arbeitete und somit die Jobs sehr ungleichmäßig auf die Ressourcen verteilt hat. Die neuen Policy verteilt die Jobs wesentlich gleichmäßiger und nutzt dadurch die bereitgestellten Ressourcen annäherungsweise optimal.

9.2 Neue Erkenntnisse

Die Evaluierung hat gezeigt, dass es schwierig ist, eine sinnvolle generische Policy zu finden, die für alle denkbaren heterogene Ressourcen verwendbar ist. Der Grund dafür ist, dass jede Ressource andere Attribute dem Informationssystem zur Verfügung stellen kann und Jobs unterschiedliche Anforderungen an die Ressourcen haben. Für die Umsetzung einer solchen Policy muss es sowohl eine standardisierte Beschreibung der Ressourcen als auch eine standardisierte Beschreibung der Semantik der Attribute einer Ressource geben.

Einem Endbenutzer kann nicht zugemutet werden, dass er anhand eines Jobs eine Vorauswahl der Ressourcen durch eine Policy oder z.B. durch einen Klassennamen der lokalen Scheduler trifft. Schließlich ist das Ziel von Grid-Computing die Bereitstellung von einem einfachen, standardisierten und kostengünstigen Zugang zu Rechenkapazitäten und anderen Dienstleistungen.

10 Ausblick

In diesem Kapitel sind Ideen für die Zukunft des Frameworks festgehalten.

10.1 Mögliche Erweiterungen des Frameworks

Dieses Kapitel enthält einige Vorschläge, um die Funktionalität des Frameworks zu erhöhen.

10.1.1 Mögliche Erweiterungen der Jobbeschreibungssprache

Es gibt zwei mögliche Erweiterungen der Jobbeschreibungssprache, um die Anforderung an Ressourcen genauer spezifizieren zu können:

1. **Logische Verknüpfung**
   Die Anforderungen können folgendermaßen logisch verknüpft werden:

   ```
   <OR>
     <AND>
       <REQUIREMENT NAME="os"         VALUE="Suse"/>
       <REQUIREMENT NAME="os-version" VALUE="5.1"/>
     </AND>
     <REQUIREMENT NAME="os" VALUE="AIX" />
   </OR>
   ```

   Diese Verknüpfung bedeutet, dass der Job entweder auf dem Betriebssystem „Suse“ mit der Version „5.1“ oder unter „AIX“ auszuführen ist. Um sämtliche logische Verknüpfungen ausdrücken zu können, wird noch die NOT-Verknüpfung benötigt.

2. **Komparator** Der Komparator wird benötigt, um Attribute auf einen bestimmten Wert oder auf einen Wertebereich zu überprüfen.

   ```
   <REQUIREMENT NAME="os-version" VALUE="5.1" COMP="gr"/>
   ```

   Das Attribut „COMP“ kann folgende drei Werte annehmen:

   - „gr“: größer als der Wert von „VALUE“
   - „lo“: kleiner als der Wert von „VALUE“
   - „eq“: gleich dem Wert von „VALUE“
   - „ne“: ungleich dem Wert von „VALUE“

   Somit bedeutet die obige Anforderung, dass der Job ein Betriebssystem mit der Versionsnummer größer als „5.1“ benötigt.

10.1.2 Verwaltung mehrerer Informationssysteme

Eine mögliche Erweiterung des Frameworks ist die Verwaltung von mehreren Adaptern für das InfoService-Interface. Der MatchMaker würde bei einer Anfrage alle registrierten InfoService-Adapter abfragen und als Resultat Ressourcen aus mehreren Informationssystemen zurück liefern. Damit könnten im Hinblick auf eine virtuelle Organisation mehrere Organisationen mit unterschiedlichen Grid--Middleware-Systemen durch den Grid-Scheduler verbunden und bedient werden. Dafür muss nach der Auswahl der Ressource durch die Policy das dazugehörige Grid-Middleware-System der ausgewählten Ressource bekannt sein, damit über den entsprechenden Adapter des Resource-Interfaces der Job an die entsprechende Ressource in das dazugehörige Grid-Middleware-System übergeben werden kann.

10.1.3 Automatische Auswahl der Policy

Für die automatische Auswahl der Policy, je nach Auslastung der Ressourcen, werden genaue Informationen über die Scheduling-Policies benötigt, wann welcher Algorithmus verwendet werden soll. Der „Self-Tuning dynP Job-Scheduler“ [Streit, 2002a][Streit, 2002b] aus Kapitel 5.3.2 wendet eine von drei Policies je nach Auslastung der Ressourcen an.

10.2 Open Grid Service Architecture (OGSA)

OGSA (Open Grid Service Architecture) [Foster, 2002b] ist die Weiterentwicklung der Architektur aus dem „Sanduhren“-Modell, die im Globus Toolkit 1.X und 2.X implementiert ist. GTK 1.X und 2.X bestehen aus einer Vielzahl an Komponenten mit je einer eigenen API und Protokollen. Für OGSA werden die Technologien des Grids mit den Vorteilen der Web Services genutzt, um damit eine allgemeinere und flexiblere Architektur zu schaffen. Diese ist für die reibungslose Eingliederung von verteilten heterogenen und dynamischen virtuellen Organisationen nötig. Die Interoperabilität wird durch standardisierte Schnittstellendefinitionen durch WSDL (Web Service Description Language) und durch die Verwendung von standardisierten Protokollen SOAP (Simple Object Access Protocol) auf der Basis von HTTP oder FTP erreicht. Die Architektur wird als zweite Generation [Zhihui, 2003] der Grid Architektur bezeichnet und definiert sogenannte Grid Services. Dieses sind Web Services, welche eine vorgegebene Menge an Schnittstellen für Discovery, Instanziierung, Lifetime Management, Notification und deren Handhabbarkeit bereitstellen.

Das Globus Toolkit 3.0 ist die Open-Source Implementierung von OGSI (Open Grid Service Infrastructure), welches die Spezifikation des Global Grid Forums ist, und die Interaktion zwischen einem Klienten und einem Web Service spezifiziert [Tuecke, 2002].

10.3 Anpassung des Frameworks für Globus Toolkit 3.x

Die IBM plant die Anpassung des Schedulers für die Nutzung der Grid-Middleware Globus Toolkit 3.0. Ein Grund dafür ist, dass es für das GTK 2.x von Globus nur noch bis zum Jahresende Support geleistet wird.

Im vorherigen Kapitel 10.2 wurde die Architektur von GTK 3.0 angesprochen. Sie unterscheidet sich grundlegend in der Architektur und API von GTK 2.x. Aus diesem Grund müssen die Implementierungen der Schnittstellen für die Abfrage des Informationssystems und zur Steuerung der Ausführung von Jobs angepasst werden.

Der einzige Teil, der von dem Framework noch angepasst werden muss, ist der RSL-spezifische Teil im Framework. Das ist ein reiner Datencontainer, der in einen generisch Datencontainer umgeschrieben werden muss. Dann ist das Framework komplett unabhängig von der Grid-Middleware.

Die Performance des Schedulers mit GTK 3.0 wird voraussichtlich wesentlich niedriger sein. Der Grund dafür ist, dass GTK 3.0 Grid Services unter Java verwendet. Die Implementierung für die Steuerung der Jobausführung (GRAM-Adapter) muss den Master Managed Job Factory Service (MMJFS) verwenden. Für die Implementierung der Abfrage des Informationssystems (MDS-Adapter) muss der Index Service abgefragt werden. Für die Nutzung von einem Service muss zuerst die Factory aktiviert werden, danach kann eine Instanz des Service erzeugt werden. Eine Instanz muss nur für die erste Nutzung eines Dienstes erstellt werden. Danach kann diese wiederverwendet werden. Erste Prototypen haben ergeben, dass die Generierung einer Instanz rund 17 Sekunden und die Übergabe eines Jobs an eine Ressource rund 11 Sekunden dauert. Für das Schedulen eines Jobs werden somit zwei Instanzen für die Abfrage der Ressource und für die Übergabe des Jobs an die Ressource benötigt. Zusätzlich kommt dann noch die Zeit für die Abfrage der Ressourcen hinzu. Somit dauert die Abfrage der Ressourcen und die Übergabe eines Jobs an eine Ressource länger als 56 Sekunden. Diese Zeit könnte mit einem Pool an Instanzen reduziert werden. Im Vergleich dazu benötigt der aktuelle Scheduler für die Abfrage und die Übergabe eines Jobs rund 5 Sekunden.

Weitere wichtige Gründe für den Umstieg auf GTK 3.0 ist die Standardisierung der Protokolle, Dienste und Schnittstellen. Dadurch, dass die Komponenten standardisiert und OGSI-fähig sind, gibt es eine bessere Interoperabilität der Komponenten untereinander, und Komponenten anderer Anbieter können eingebunden werden [Gargya, 2003].

11 Begriffslexikon

Begriff	Beschreibung
AFS	Andrew File System; verteiltes Filesystem ähnlich zu NFS
AIX	UNIX Betriebssystem von IBM
API	Application Programming Interface; besteht aus Datentypen, Datenstrukturen und Funktionen zur Erstellung von Anwendungen
Application-Schicht	oberste Ebene der Schichtenstruktur einer Grid-Architektur; siehe Kapitel 2.2.2
AppLeS	Application Level Scheduler; dezentraler Grid-Scheduler der University of California, San Diego (siehe Kapitel 5.2.1)
Ausführungsplan	Ergebnis eines Schedulers; beschreibt, wann welche Aufgabe (z.B. Job) ausgeführt wird
BNF-Grammatik	Backus-Naur-Form; Grammatik zur Beschreibung von kontextfreien formalen Sprachen
Checkpointing	Festhalten des Zustands der Verarbeitung eines Jobs (siehe Kapitel 2.3.1)
ClassAds	Beschreibung der Anforderungen und Vorzüge einer Ressource oder eines Jobs vergleichbar mit einer Anzeige in der Zeitung; wird für das Matchmaking bei Condor-G eingesetzt
Cluster	siehe Kapitel 2.2.1
Collective-Schicht	Schicht einer Grid-Architektur für die Interaktion zwischen Ressourcen; siehe Kapitel 2.2.2
Condor-G	Grid-Scheduler der University of Wisconsin (siehe Kapitel 5.2.3)
Connectivity-Schicht	Schicht einer Grid-Architektur; legt die Kommunikations- und Authentifizierungsprotokolle für Grid-spezifische Netzwerktransaktionen fest; siehe Kapitel 2.2.2
DB2	Datenbanksystem von IBM
DNS	Domain Name Service
Fabric-Schicht	unterste Ebene der Schichtenstruktur einer Grid-Architektur; siehe Kapitel 2.2.2
FIFO	First In First Out; Strategie zur Einfügung und Entnahme von Einträgen aus Warteschlangen
Framework	gibt die Architektur für eine Klasse von Anwendungen vor; siehe Kapitel 2.4
GFTP	Protokoll für den Datenaustausch; Grid File Transfer Protocol (siehe Kapitel 3.1.2)
GGF	Global Grid Forum; Gemeinschaft zur Entwicklung von Grid-Technologien
GIIS	Grid Index Information Service; Informationssystem (vgl. MDS)

Globus Toolkit	Sammlung von Tools zum Aufbau einer Grid-Infrastruktur (siehe Kapitel 3.1)
GRAM	Grid Resource Allocation Management (siehe Kapitel 3.1.2); Komponente zur Verwaltung von Anwendungen auf einer Ressource
Grid	Hard- und Softwareinfrastruktur zur Nutzung von Ressourcen (siehe Kapitel 2.1.2)
Grid Computing	Verwendung eines Grids für die Ausführung von Anwendungen
Grid-Middleware	Softwareinfrastruktur für den einheitlichen Zugriff auf heterogene Ressourcen
Grid-Scheduler	verteilt Jobs an andere Grid-Scheduler oder lokale Scheduler
GSI	Grid Security Infrastructure; Grid-Komponente zur Gewährleistung der Sicherheit
GTK	siehe Globus Toolkit
GUI	Graphical User Interface
Heterogen Computing	Nutzung von heterogenen Systemen
Heuristik	„Ein Prozess, welcher ein Problem lösen kann, aber keine Lösung garantiert, wird als Heuristik bezeichnet“ nach Novell, Shawn und Simon, 1963
Information Provider	Programm, welches einen Informationsdienst mit Informationen versorgt
Informationsdienst	Dienst, der Informationen bereitstellt
Kostenmodell	beschreibt die Kosten für eine Leistung
Java CoG	ermöglicht den Zugriff von Java aus auf die Dienste des Globus Toolkits
Job	stellt eine atomare Aufgabe für einen Grid-Scheduler dar, die er mit einer Ressource lösen muss
JSDL	Job Submission Description Language
LoadLeveler	lokaler Scheduler von IBM
LabGrid	Grid des IBM Entwicklungslabors in Böblingen
LDAP	Light Weight Directory Access Protocol; zur Abfrage von Informationen aus dem X.500 Verzeichnis
LDIF	LDAP Data Interchange Format; beschreibt Verzeichnisse und Verzeichniseinträge des X.500 Verzeichnisses
Lokaler Scheduler	verteilt Jobs direkt an Endsysteme; z.B. innerhalb eines Clusters
Mapping	Abbildung
Mars	Framework für einen Grid-Scheduler der Uni-Paderborn (siehe Kapitel 5.2.4)
Matchmaking	Auswahl von potentiell in frage kommende Ressourcen für die Ausführung eines Jobs

MDS	Monitoring and Discovery Service; Informationsdienst des Globus Toolkits (siehe Kapitel 3.1.2)
Metacomputing	Zusammenschluss von Clustern, die nach außen eine homogene Ressource darstellen
Migration	Übertragung der Bearbeitung eines Jobs auf eine andere Ressource
MPP-System	Massive Parallel Processor-System; Mehrprozessorrechner mit verteiltem Speicher, nachrichtengekoppelt
Nimrod-G	Grid-Scheduler der Monash University/Australien (siehe Kapitel 5.2.2)
Performance-Modell	Beschreibung der Ausführungsgeschwindigkeit eines Jobs
Policy	Scheduling-Algorithmus; beschreibt, wie die Aufgaben auf die Ressourcen verteilt werden
Prophet	Grid-Scheduler (siehe Kapitel 5.2.4)
Prototyp	hier: Implementierung eines konkreten Grid-Schedulers mit der Hilfe des in dieser Arbeit entwickelten Frameworks
Rescheduling	Verfahren zur Optimierung des Ausführungsplans (siehe Kapitel 2.3.1)
Resource-Schicht	Schicht einer Grid-Architektur; definiert Protokolle zur Verwaltung einzelner Ressourcen der Fabric-Schicht
Ressourcen	Komponenten, die für den gemeinsamen Zugriff freigegeben sind, z.B. Rechen- und Speicherkapazität, Bandbreite, Softwarelizenzen oder auch ein Elektronenmikroskop. In dieser Diplomarbeit ist in den meisten Fällen ein Cluster damit gemeint.
Round-Robin	Policy zur fairen Verteilung reihum von Aufgaben auf Ressourcen
RSL	Resource Specification Language; Ressourcenbeschreibungssprache im Grid
Scheduler	Programm zur Verteilung von Jobs an Ressourcen mit Hilfe von Policies
Scheduler-Policy	siehe Policy
SDK	Software Development Kit; Menge von Programmen zur Erstellung von Anwendungen
Servlet	Java Programm, welches auf einem Server läuft und die Funktionalität von CGI's übernimmt (z.B. DB-Zugriff).
Silver/Maui	Grid-Scheduler vom Ohio Supercomputer Center (siehe Kapitel 5.2.4)
SPMD	Single Program Multiple Data. Es wird für jede Partition an Daten die gleiche Anwendung asynchron ausgeführt.

Staging	Vorladen von Daten für die spätere Benutzung z.B. auf das System oder in den Cache
Systemkomponenten	Endsysteme, Cluster, ein Intranet oder das Internet
Tomcat	ist ein open-source Servlet Container oder auch Application Server der Apache Software Foundation
Virtuelle Organisation	Beispielsweise ein organisationsübergreifender Verbund an Ressourcen; siehe Kapitel 2.1.4
Warteschlange	geordnete Datenstruktur die aus gleichartigen Einträgen besteht und der Reihe nach abgearbeitet wird, z.B. FIFO-Liste
XML	Extended Markup Language zur Beschreibung von Daten
XML-Tag	Begrenzer eines Elements einer XML-Datei; Start- und Endtag begrenzen ein Element. z.B. `<NAME>Tobias</NAME>`

Literatur

[Berman, 1997] Francine Berman, Richard Wolski, *The AppLeS Project, A Status Report*, Proceedings of the 8th NEC Research Symposium, Berlin, Germany, May 1997

[Berman, 2003] Fran Berman, Geoffrey C. Fox, Anthony J. G. Hey, *Grid Computing - Making the Global Infrastructure a Reality*, Wiley, 2003

[Buyya, 2000a] Rajkumar Buyya, David Abramson, Jonathan Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computional Grid*, HPC Asia 2000, Beijing, China, May 14-17, 2000, p.283-289

[Buyya, 2000b] Rajkumar Buyya, Steve Chapin, and David DiNucci, *Architectural Models for Resource Management in the Grid*, The First IEEE/ACM International Workshop on Grid Computing (GRID 2000), Bangalore, India, Springer Verlag LNCS Series, Germany, Dec. 17, 2000

[Catlett, 2002] Charlie Catlett, *The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility*, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID02), 2002

[CAVE5d, 1996] G. H. Wheless, C. M. Lascara, A. Valle-Levinson, D. P., Brutzman, W. Sherman, W. L. Hibbard, and B. Paul., *Virtual chesapeake bay: Interacting with a coupled physical/biological model.*, IEEE Computer Graphics and Applications, 16(4):5257, July 1996

[CERN, 2003] *Homepage vom CERN*, European Organization for Nuclear Research, 2003 http://www.cern.ch, 2003

[Chun, 2001] Gregory Chun, Michael O. McCracken, Josh Wills, *Scheduling in Metacomputing Environments*, University of California, San Diego, 2001

[Donno, 2002] F.Donno, L.Gaido, A.Ghiselli, F.Prelz, M.Sgaravatto, *First Prototype of DataGRID: The DataGRID Collaboration*, TERENA Networking Conference 2002, 2002

[Earth System Grid, 2003] *An Ontology for Scientific Information in a Grid Environment: the Earth System Grid*, 3rd International Symposium on Cluster Computing and the Grid, , Tokyo, Japan, May 12 - 15, 2003, p.626

[Ferreira, 2002] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, N. Bieberstein, *Introduction to Grid Computing with Globus*, IBM, 2002

[Foster, 1998] Ian Foster, Carl Kesselmann, *The Globus Project: A Status Report*, Proc. IPDPS/SPDP 1998 Heterogeneous Computing Workshop, 1998, p. 4-18

[Foster, 1999] Ian Foster, Carl Kesselman, *The Grid - Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999

[Foster, 2001] Ian Foster, Carl Kesselmann, Steven Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Lecture Notes in Computer Science, Volume 2150, 2001

[Foster, 2002a] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, *Grid Services for Distributed System Integration*, IEEE Computer 35(6), 37-46, 2002

[Foster, 2002b] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.*, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002

[Frey, 2001] James Frey, Todd Tannenbaum, Miron Livny, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, 2001

[Gargya, 2002] Tony Gargya, *BBLab Grid Presentation at Grid University*, Montpellier, IBM, 2002

[Gargya, 2003] Tony Gargya, Boas Betzler, *Unter Strom: Grid Computing im Enterprise-Einsatz*, Linux Enterprise, Software & Support Verlag GmbH, 03/2003

[Gehring, 1996] Jörn Gehring, Alexander Reinefeld, *A Framework for Minimizing the Job Execution Time in a Metacomputing Environment*, Proceedings of Future general Computer Systems, 1996

[Gehring, 1999] Jörn Gehring, Thomas Preiß, *Scheduling a Metacomputer With Uncooperative Sub-schedulers*, Job Scheduling Strategies for Parallel Processing, Springer Verlag, 1999, p.179-201

[Larson, 2003] Larson SM, Snow CD, Shirts MR, Pande VS., *Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology.*, Computational Genomics, Horizon Press, 2003

[Legrand, 2003] A. Legrand, L. Marchal, H. Casanova, *Scheduling Distributed Applications: The SIMGRID Simulation Framework*, in Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan, May 2003

[GGF, 2003a] *JSDL Working Group Homepage*, http://www.epcc.ed.ac.uk/ãli/WORK/GGF/JSDL-WG/, 2003

[GGF, 2003b] *Homepage vom Global Grid Forum*, http://www.ggf.org/, 2003

[Globus, 2003a] *Homepage von Globus*, http://www.globus.org, 2003

[Globus, 2003b] *Homepage von Globus, Thema RSL*, http://www-fp.globus.org/gram/gram_rsl_parameters.html, 2003

[Hamscher, 2000] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, Ramin Yahyapour *Evaluation of Job-Scheduling Strategies for Grid Computing*, Proceedings of 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000) at 7th International Conference on High Performance Computing (HiPC-2000), Bangalore, India, LNCS 1971, p.191-202

[Futrelle, 2001] J. Futrelle, *Building a Repository of Distributed Heterogeneous Scientific Data for NEESGrid*, COSMOS/PEER-LL Workshop on Archiving and Web Dissemination of Geotechnical Data October 4 and 5, Pacific Earthquake Engineering Research Center, Berkeley, 2001

[James, 1999] H.A. James, K.A. Hawick and P.D. Coddington, *Scheduling Independent Tasks on Metacomputing Systems*, Proc. of Parallel and Distributed Computing Systems (PDCS99), Fort Lauderdale, August 1999, Tech Note DHPC-066

[JavaCoG, 2003] *Homepage von Java CoG 0.9.13*, http://www-unix.globus.org/cog/java/0.9.14/, 2003

[Kannan, 2001] Subramanian Kannan, Mark Roberts, Peter Mayes, Dave Brelsford, Joseph F Skovira, *Workload Management with LoadLeveler*, IBM Redbook, 2001

[Karpovich, 1996] John F. Karpovich, *Support for Object Placement in Wide Area Heterogeneous Distributed Systems*, Technical Report CS-96-03, University of Virginia, 1996

[Kennedy, 2002] Ken Kennedy, Mark Mazina, John Mellor-Crummey, Keith Cooper, Linda Torczon, Fran Berman, Andrew Chien, Holly Dail, Otto Sievert, *Toward a Framework for Preparing and Executing Adaptive Grid Programs*, International Parallel and Distributed Processing Symposium, 2002

[Korpela, 2003] Korpela, E., Werthimer, D., Anderson, D., Cobb, J., and Lebofsky, M., *SETI@home: Massively Distributed Computing for SETI*, Computing in Science and Engineering, 3(1), 2001

[Litzkow, 1997] M. Litzkow, T. Tannenbaum, J. Basney, M. Livny, *Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System*, University of Wisconsin-Madison Computer Science, Technical Report 1346, 1997.

[Lorch, 2002] Markus Lorch, Dennis Kafura, *Symphony - A Java-based Composition and Manipulation Framework for Computational Grids*, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, Germany, 21 - 24. May 2002

[Luemkemann, 2003] Jan Lümkemann, *Integration of Linux on zSeries into Böblingen LabGrid*, Berufsakademie Stuttgart, 2003

[Maheswaren, 1999] M. Maheswaren, S. Ali, H.J. Siegel, D. Hensgen, *Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems*, 8th Heterogeneous Computing Workshop, 1999

[Murshed, 2002] Manzur Murshed and Rajkumar Buyya, *Using the GridSim Toolkit for Enabling Grid Computing Education*, International Conference on Communication Networks and Distributed Systems Modeling and Simulation (CNDS 2002), San Antonio, Texas, USA, January 27-31, 2002

[Netera Alliance, 2003] *Homepage von Netera Alliance*, http://www.netera.ca/grid/whatis.htm, 2003

[Ovtcharov, 2002] Gueorgui Ovtcharov, *Scheduling in Grid Computing*, Universität Stuttgart, Ausarbeitung zum Hauptseminar Internet-Technologien der nächsten Generation, WS 2002/2003

[Powers, 2003] Dan Powers, Jean-Pierre Prost, *Application Adoption at IBM*, Presentation, Global Grid Forum Applications and Testbeds Research Group, GGF7, Tokyo, Japan, 2003

[Raman 1998] Rajesh Raman, Miron Livny, Marvin Solomon, *Matchmaking: Distributed Resource Management for High Throughput Computing*, HPDC 1998, 1998, p.140

[Raman, 2000] Rajesh Raman, Miron Livny, Marvin Solomon, *Resource Management through Multilateral Matchmaking*, HPDC 2000, 2000, p.290-291

[Schopf, 2002] Jennifer Schopf, *A General Architecture for Scheduling on the Grid*, JPDC, Special Issue on Grid Computing, April 2002

[Simec, 2002] Dean Simec, *Einführung in Grid Computing*, Universität Stuttgart, Ausarbeitung zum Hauptseminar Internet-Technologien der nächsten Generation, WS 2002/2003

[Song, 2000] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, A. Chien, *The Microgrid: a Scientific Tool for Modeling Computational Grids*, In Proceedings of SC2000, Dallas, Texas, 2000

[Streit, 2002a] Achim Streit, *The Self-Tuning dynP Job Scheduler*, International Parallel and Distributed Processing Symposium: IPDPS 2002 Workshops, Fort Lauderdale, Florida, April 15 - 19, 2002, p.87

[Streit, 2002b] Achim Streit, *The Self-Tuning Job Scheduler Family with Dynamic Policy Switching*, Job Scheduling Strategies for Parallel Processing, 8th International Workshop, JSSPP 2002, Edinburgh, Scotland, UK, July 24, 2002, p.1-23

[Subramani, 2002] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, P. Sadayappan, *Distributed Job Scheduling on Computational Grids*

using Multiple Simultaneous Requests, Proceedings of 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), July 2002

[Takefusa, 1999] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima., *Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms*, Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8), 08/1999, p.97-104

[TU Wien, 2003] *Homepage der TU Wien*, http://www.tuwien.ac.at/zv/pce/evaluierung.shtml, 2003

[Tuecke, 2002] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, *Open Grid Service Infrastructure*, WG, Global Grid Forum, Draft 2, 7/17/2002

[Weissman, 1999] Jon B. Weissman, *Prophet: automated scheduling of SPMD programs in workstation networks*, Concurrency - Practice and Experience 11(6), 1999, p.301-321

[Wu, 2000] Min-You Wu, Wei Shu, *Segmented Min-Min: A Static Mapping Algorithm for Meta-tasks on Heterogeneous Computing Systems*, 9th Heterogeneous Computing Workshop, Cancun, Mexico, 2000, p.375

[Wydler, 2001] Robert Wydler, *Objektorientierte, serverseitige Rahmenwerke*, http://www.educeth.ch/informatik/vortraege/frameworks/docs/text_frameworks.pdf, Eidgenössische Technische Hochschule Zürch, 2001

[Yang, 2002] Jinglei Yang, *Resource Management in Grid Computing*, Universität Stuttgart, Ausarbeitung zum Hauptseminar Internet-Technologien der nächsten Generation, WS 2002/2003

[Zeta, 2003] *Homepage von ZetaGrid*, http://zetagrid.net/, 2003

[Zhihui, 2003] Zhihui Du, Bin Du, Jiang Du, Niansheng Zhou, Xiaoge Wang, *OGSA based Grid Computing*, The First Dong-A and Tsinghua University International Technical Symposium. Korean, Busan, Dong-A University, 2003

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfasst und nur die
angegebenen Quellen benutzt zu haben.

____

(Tobias Beichter)