HASSO – PLATTNER – INSTITUT

für Softwaresystemtechnik GmbH an der Universität Potsdam

Grid-Computing

Technische Berichte Nr. 3

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

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

Nr. 3

Grid-Computing

Potsdam 2005

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

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

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

Redaktion: Dipl. Inf. Peter Tröger, Sabine Wagner
Email: {peter.troeger, sabine.wagner}@hpi.uni-potsdam.de

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

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

3., unveränd. Nachdr., 2005
Heft 3 (März 2004)
ISBN 3-937786-28-7
ISSN 1613-5652

Grid-Computing

Seminar im Sommersemester 2003

Prof. Dr. Andreas Polze
Universität Potsdam
Hasso-Plattner-Institut für Softwaresystemtechnik
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam
eMail: andreas.polze@hpi.uni-potsdam.de

Prof. Dr. Bettina Schnor
Universität Potsdam
Institut für Informatik
August-Bebel-Strasse 89
14482 Potsdam
eMail: schnor@cs.uni-potsdam.de

Der Lehrstuhl "Betriebssysteme und Middleware" (Prof. Dr. Andreas Polze) des Hasso-Plattner-Instituts und der Lehrstuhl "Betriebssysteme und Verteilte Systeme" (Prof. Dr. Bettina Schnor) des Instituts für Informatik haben im Sommersemester 2003 gemeinsam ein Seminar zum Thema „Grid-Computing“ durchgeführt. Der vorliegende technische Bericht stellt eine Zusammenfassung der entstandenen studentischen Ausarbeitungen dar.

Als Grid bezeichnet man ein massiv verteiltes System, welches das Anbieten, Auffinden und die Nutzung von Ressourcen über verschiedene administrative Domänen hinweg ermöglicht. Dabei werden u.a. Faktoren wie Verfügbarkeit, Authentifizierung, Kosten-Abrechnung und Quality-of-Service als Nutzer-Anforderungen berücksichtigt. Somit können Ressourcen von weitverteilten, völlig verschiedenen Organisationen koordiniert genutzt werden. Als Analogie werden hier die Eigenschaften des Stromnetzes ("Power Grid") gesehen - die jeweilige Ressource ist im Idealfall unabhängig von der Quelle gleichbleibend verfügbar.

Im Kontext von Grid-Computing spielen Clustertechnologien für den Betrieb der Grid-Knoten eine entscheidende Rolle. Aus diesem Grund finden sich in diesem Bericht neben den klassischen Grid-Computing-Themen (Globus, OGSA, Scheduling, Applikationen) auch Ausarbeitungen aus dem Bereich des Parallel Computing.

Inhalt

1. Applikationen für weitverteiltes Rechnen
Dennis Klemann, Lars Schmidt-Bielicke, Philipp Seuring

2. Das Globus-Toolkit
Dietmar Bremser, Alexis Krepp, Tobias Rausch

3. Open Grid Services Architecture
Lars Trieloff

4. Condor, Condor-G, Classad
Stefan Henze, Kai Köhne

5. The Cactus Framework
Thomas Hille, Martin Karlsch

6. High Performance Scheduler mit Maui/PBS
Ole Weidner, Jörg Schummer, Benedikt Meuthrath

7. Bandbreiten-Monitoring mit NWS
Alexander Ritter, Gregor Höfert

8. The Paradyn Parallel Performance Measurement Tool
Jens Ulferts, Christian Liesegang

9. Grid-Applikationen in der Praxis
Steffen Bach, Michael Blume, Helge Issel

Applikationen für weitverteiltes Rechnen

Dennis Klemann, Lars Schmidt-Bielicke, Philipp Seuring

{dennis.klemann, lars.bielicke, philipp.seuring}@student.hpi.uni-potsdam.de

Beim „Wide Area Distributed Computing“ handelt es sich um eine Möglichkeit der Verteilung von Rechenaufwand. Jede Art des Verteilens von Anwendungen auf mehrere Prozessoren dient dazu, Probleme, die nur mit immensem Rechenaufwand gelöst werden können, zu

Inhalt

1 Paralleles Rechnen und WADC 3
1.1 Einleitung 3
1.2 Klassifikation von Rechensystemen 5
1.3 WADC und Alternativen 6
1.4 Paralleles Rechnen 7
1.5 Merkmale aktueller WADC-Projekte 9
2 WADC oder Grid Computing? 11
2.1 Wide Area Distributed Computing 11
2.2 Beispiel einer Grid Computing-Architektur 12
2.3 Abgrenzung WADC gegen Grid Computing 13
2.4 Zentrales Problem: Ressourcen-Management 15
2.5 GridBank 17
2.6 Zusammenfassung 18
3 WADC Projekte 19
3.1 SETI@home 19
3.2 distributed.net 22
3.3 United Devices 24
3.4 BOINC – Berkeley Open Infrastructure for Network Computing 26
4 Praktische Arbeit – SoBSieveServer 28
4.1 Vorstellung des Hauptprojektes – Seventeen or Bust 28
4.2 Unterprojekt: SoBSieve 28
4.3 SoBSieveServer 30

Einleitung

Die Ausarbeitung teilt sich im Wesentlichen in vier Kapitel. Im ersten wird auf paralleles und verteiltes Rechnen im Allgemeinen eingegangen. Dabei werden grundlegende Begriffe definiert und für das Verständnis vom verteiltem Rechnen wichtige Konzepte kurz erläutert wie eine Einordnung von WADC in diese Begriffswelt gegeben. Im zweiten Kapitel wird näher auf den Begriff des GRID eingegangen und die Unterschiede zum WADC werden herausgestellt. Kapitel drei widmet sich aktuell laufenden Projekten im Bereich des WADC und beleuchtet wichtige Aspekte wie Aufbau und Abläufe und Sicherheitsaspekten der prominentesten Vertreter. Im vierten Kapitel stellen wir unsere Praxisarbeit des Seminars vor: Die Sobsieve Client Server Anwendung.

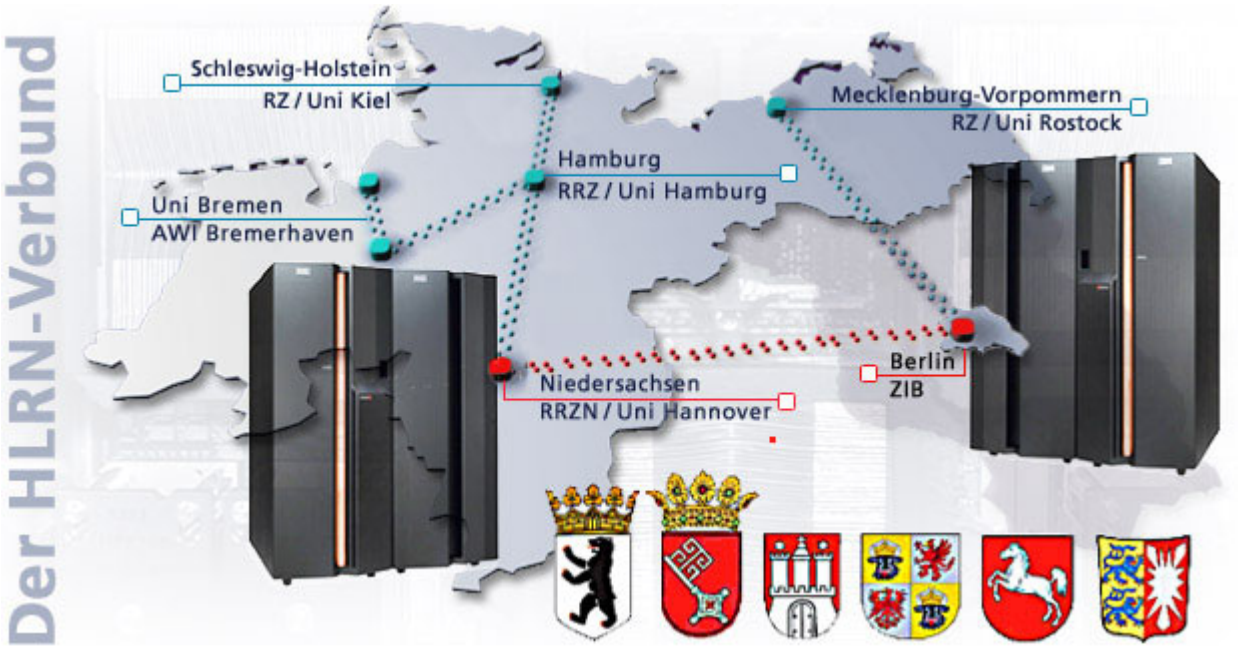
1 Paralleles Rechnen und WADC

1.1 Einleitung

Beim „Wide Area Distributed Computing“ handelt es sich um eine Möglichkeit der Verteilung von Rechenaufwand. Jede Art des Verteilens von Anwendungen auf mehrere Prozessoren dient dazu, Probleme, die nur mit immensem Rechenaufwand gelöst werden können, zu bearbeiten.

1.1.1 Wozu die große Rechenleistung?

Es gibt zahlreiche Aufgaben die ungeheure Rechenkapazitäten benötigen. Dazu zählen zum Beispiel Wettersimulationen, Berechnungen zur Krebsvorsorge, mathematische Projekte und vieles mehr. Neben Wissenschaftlern aus Chemie, Physik und Geologie hat auch in ganz erheblichem Maße das Militär ein großes Interesse an Simulationen, die riesige Mengen von Rechenkapazität benötigen, zum Beispiel zur exakten Simulation von Nuklearexplosionen, um den umstrittenen Testaufwand von realen Tests zu minimieren.



In Deutschland kann als Beispiel die Initiative vom HLRN (Norddeutscher Verbund für Hoch- und Höchstleistungsrechnen) genannt werden, die Ende letzten Jahres Deutschlands schnellsten Rechner mit 4 Terraflops in Betrieb genommen hat. Dieses Projekt, mit dem den Hochschulen der norddeutschen Bundesländer Berechnungen aus den Bereichen Astronomie, Meeres- und Küstenforschung, Schiffsbau und Baumechanik ermöglicht werden sollen, hat ein Kosten-Volumen von 20 Millionen Euro. Es besteht also enormer Bedarf an Rechenleistung, den sich die entsprechenden Einrichtungen auch einiges Kosten lassen.

1.1.2 Warum Anwendungen verteilen?

Wenn man über das verteilte Rechnen von Anwendungen spricht muss man sich zuerst einmal die Frage stellen, wozu dies überhaupt benötigt wird. Aufgrund der begrenzten Rechenkapazität einzelner Prozessoren wird seit langer Zeit an der Kopplung mehrerer Prozessoren und somit einer Vervielfachung der verfügbaren Rechenkapazität geforscht. Dabei werden die zu berechnenden Probleme in Subprobleme zerlegt, die parallel oder nebenläufig gerechnet werden können. Durch dieses Verfahren wird erreicht, dass der Steigerung der Rechenkapazität theoretisch keine Grenzen gesetzt sind.

Zurzeit laufen die wenigsten der auf der Erde verfügbaren Prozessoren unter Volllast. Die Auslastung beträgt im Schnitt nur 10% bis 30%. Um sich dieses Phänomen nützlich zu machen wird an Möglichkeiten gearbeitet, diese Prozessorleistungen untereinander zu tauschen, d.h. dass Prozessoren, die gerade vom Besitzer nicht gebraucht werden, für andere Aufgaben benutzt werden können. Der Besitzer dieser ausgeliehenen Maschine darf im Gegenzug bei Bedarf auch Prozessoren auf Maschinen benutzen, die ihm nicht gehören.

Dazu sind jedoch umfangreiche und mächtige Frameworks zum Verteilen der Kapazität, zum Scheduling der Jobs und zur Abrechung notwendig.

1.2 Klassifikation von Rechensystemen

Um über die Besonderheit von „Wide Area Distributed Computing“ sprechen zu können muss erst einmal eine generelle Klassifikation von Rechnern vorgenommen werden. Hierbei ist vor allem die Klassifikation von Flynn gebräuchlich.

1.2.1 Klassifikation von Flynn

Von Flynn werden Rechner als Operatoren auf verschiedenartigen Informationsströmen (Befehlsstrom und Datenstrom) folgendermaßen klassifiziert.

- SISD: Single instruction stream over a single data stream
- SIMD: Single instruction stream over multiple data streams
- MISD: Multiple instruction streams over a single data stream
- MIMD: Multiple instruction streams over multiple data streams

In der Klasse SISD handelt es sich um von-Neumann-Architekturen (Einzelprozessorrechner), die MISD Klasse ist leer, SIMD sind Feld- und Vektorrechner und in der MIMD Klasse finden sich die Multiprozessorsysteme, die im Folgenden weiter behandelt werden.

1.2.2 Multiprozessorsysteme

Bei Multiprozessorsystemen unterscheidet man zwischen

- Speichergekoppelte Systeme – die Kommunikation erfolgt über einen gemeinsamen Speicher, d.h. alle Prozessoren verfügen über einen gemeinsamen Adressraum und kommunizieren und synchronisieren sich über gemeinsam genutzte Variablen.
- Nachrichtengekoppelte Systeme – die Prozessoren verfügen nur über physikalisch verteilte Speicher und prozessorlokale Adressräume. Die Kommunikation erfolgt über den Austausch von Nachrichten.

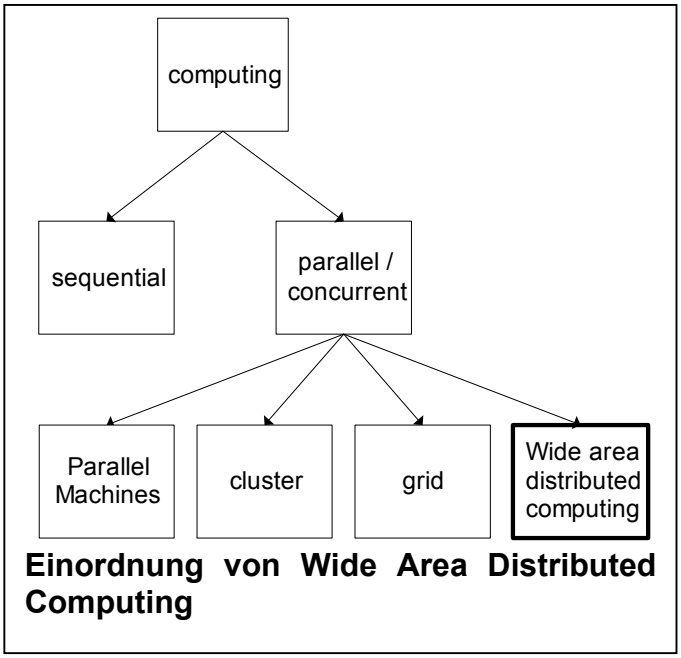
Beim WADC befinden wir uns in der Kategorie der nachrichtengekoppelten Multiprozessorsysteme. Hier gibt es eine weitere Unterscheidung zwischen UCA und NUCA. UCA (Uniform Communication Architecture) heißt, dass zwischen allen Prozessoren gleichlange Nachrichten mit einheitlicher Übertragungszeit geschickt werden können, während bei NUCA (Non Uniform Communication Architecture) die Übertragungszeiten der Nachrichtentransfers je nach Sender- und Empfängerprozessor variieren können.

In dieser Kategorisierung ist die Komplexität der Programmierung bei NUCA Systemen am höchsten. Dafür ist eine hohe Skalierbarkeit gegeben. Deshalb eignen sich nur diese Systeme für Höchstleistungsrechner im Teraflopbereich.

1.3 WADC und Alternativen

Beim Austausch von Rechenkapazität gibt es unglaublich viele Möglichkeiten, die einzelnen Prozessoren untereinander zu verbinden. Eine gemeinsame Voraussetzung müssen die zu berechnenden Programme jedoch auf jeden Fall erfüllen. Sie müssen parallelisierbar sein, d.h. es dürfen nicht alle Teile des Programms streng voneinander abhängen und damit nur jeweils ein einzelner Prozessor mit der Lösung der Aufgabe betraut sein kann.

Ist diese Vorraussetzung erfüllt, gibt es verschiedene Arten des Verteilens von Anwendungen. In Abbildung ist die Einteilung in vier Hauptmöglichkeiten gezeigt, die dabei verwendet werden.



Einordnung von Wide Area Distributed Computing

1.3.1 Parallelrechner

Parallelrechner sind Maschinen mit mehreren Prozessoren. Dabei befinden sich die Prozessoren alle in der gleichen Maschine und können auf einen gemeinsamen Speicher zugreifen. Eine vollständig getrennte Nutzung der Ressourcen ist nicht möglich.

1.3.2 Cluster

Cluster bestehen aus mehreren baulich getrennten Maschinen, die über eine Breitbandverbindung verfügen. Dabei bilden die einzelnen Maschinen eine organisatorische Einheit.

1.3.3 Grid

Ein Grid ist eine über mehrere Organisationseinheiten hinausgehende Einheit, die zur Berechnung zusammengeschaltet werden kann. Hier sind die einzelnen Komponenten des Grid nicht fest miteinander verbunden, sondern können auch einzeln und in unterschiedlichen Konfigurationen arbeiten.

1.3.4 Wide Area Distributed Computing

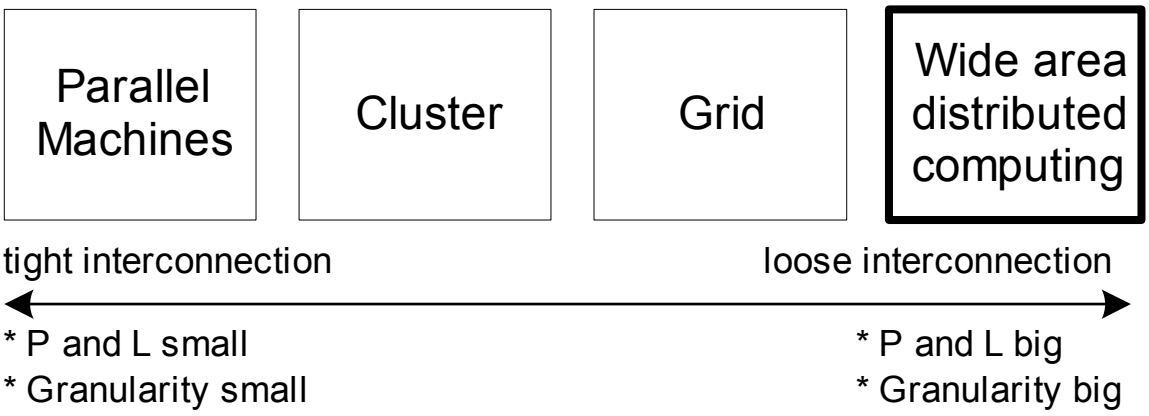
Die einzelnen Rechner sind über eine große räumliche Distanz verteilt. Die Datenverbindung zwischen den Rechnern hat oft nur eine sehr kleine Bandbreite, und die Komponenten des verbundenen Netzwerks sind relativ unzuverlässig und nicht konstant als Rechenkraft einsetzbar.

1.4 Paralleles Rechnen

Bei der parallelen Berechnung von Problemen werden verschiedene Eigenschaften der Systeme wichtig. Dabei ist vor allem die Granularität der zu verteilenden Arbeit ein Hauptunterscheidungsmerkmal bei der Wahl des zu verwendenden Paradigmas. Abhängig von der Verbindung zwischen den Prozessoren muss der Kommunikationsaufwand zwischen ihnen minimiert werden.

1.4.1 Granularität

Unter Problemen mit einer groben Granularität versteht man Anwendungen, bei denen die Verteilung über relativ große Einheiten wie Programme, Prozesse oder Blocks geregelt wird. Mit fein granular bezeichnet man die Verteilung von Operationen oder gar Suboperationen auf die unterschiedlichen Prozessoren.



1.4.2 LOGP

LOGP ist ein Modell für die Verbindung zwischen den Systemkomponenten. Es wird zur Entwicklung und Analyse von schnellen parallelen Algorithmen in Abhängigkeit von der vorhandenen Netzstruktur genutzt und geht darauf ein, wie wichtig Datenverteilung und Kommunikationsscheduling sind.

Diese Abkürzung steht für

- Latency – obere Grenze der maximalen Datenlaufzeit
- Overhead – Zeit die zum Versenden und Empfangen der Daten (Ver- und Entpacken der Nachricht) benötigt wird
- Gap – minimales Intervall innerhalb dessen 2 aufeinander folgende Nachrichte versendet oder empfangen werden können
- Processors – Anzahl der, im System vorhandenen, Prozessoren

Dabei werden L, O und G in Ticks oder Zyklen angegeben und Nachrichten werden als kleine Pakete fixierter Größe definiert. Also ergibt sich daraus, dass zwischen zwei Knoten maximal L/G Nachrichten laufen können.

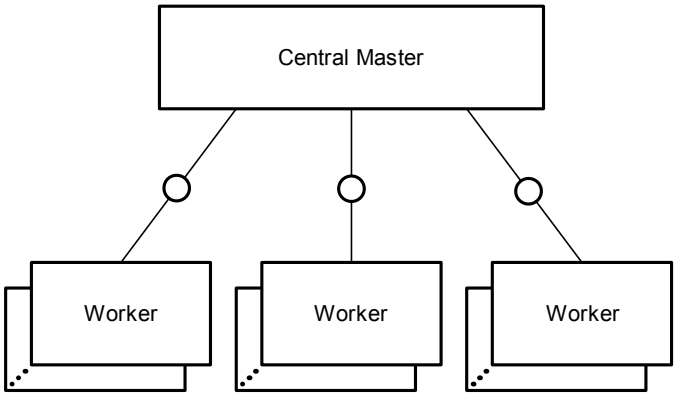
Wie in der Abbildung oben dargestellt unterscheiden sich die Arten des verteilten Rechnens ganz extrem in diesen Maßen. Von links nach rechts steigt zwar die Anzahl der potentiell verbundenen Prozessoren, es nimmt jedoch auch in ganz erheblichem Maß die Latenzzeit zu. Während man bei Parallelrechner mit einer minimalen und genau bekannten Latenzzeit rechnen kann, ist dies, bei über das Internet verbundenen Rechnern, die ein Wide Area Network formen, nicht mehr möglich.

Aufgrund dieser Überlegungen machen beim WADC nur Anwendungen Sinn, die über einen sehr geringen Kommunikationsaufwand verfügen. Das heißt, dass die zu berechnenden Probleme hochgradig parallelisierbar sein müssen und stark unabhängig voneinander gelöst werden können. Nicht nur die Häufigkeit der benötigten Kommunikation, sondern auch die Menge der zu transferierenden Daten spielen hierbei eine entscheidende Rolle.

1.5 Merkmale aktueller WADC-Projekte

Betrachtet man Projekte, die aktuell mit WADC bearbeitet werden, so fallen gravierende Gemeinsamkeiten auf. Aufgrund der geringen Bandbreite über das Internet handelt es sich meist um eine Master/Worker Architektur.

1.5.1 Ablauf



Dabei gibt es einen zentralen Master, der die Arbeit aufsplittet und verteilt. Eine große Zahl unabhängiger Worker, die vom Master unabhängige Subprobleme zum Lösen bekommen, ist über eine Internetverbindung mit ihm verbunden. Nachdem der Master die Arbeit verteilt hat ist meist bis zum Mitteilen des Ergebnisses keine weitere Kommunikation mehr nötig.

1.5.2 Einschränkungen

Ein solches System bringt eine Vielzahl von Einschränkungen für die zu berechnenden Probleme mit sich. Da bei den bekannten bestehenden Projekten jeder Internetanwender als Worker seine CPU Zeit zur Lösung der Probleme bereitstellen kann, ist es problematisch, sicherheitskritische Daten zum Worker zu übertragen. Auch muss der Master sicherstellen, dass die Ergebnisse des Clients korrekt sind. Es ist durchaus denkbar, dass der Worker gar nicht die vorgesehenen Algorithmen zur Lösung des Problems benutzt, sondern einfach zufällige Ergebnisse zurückschickt. In den meisten Projekten ist dies über die redundante Vergabe von Aufträgen geregelt. Mehrere Worker berechnen das gleiche Problem und die unterschiedlichen Ergebnisse werden vom Master abgeglichen.

Eine weitere Einschränkung besteht in der Unzuverlässigkeit der Worker. Der Master kann nicht davon ausgehen, dass er überhaupt eine Antwort vom Worker erhält. Es ist deshalb unbedingt notwendig, Timeout-Mechanismen vorzusehen und nach deren Ablauf die Arbeit neu zu verteilen.

Aber auch für den Worker gibt es entscheidende Fragen. Da er seine CPU Zeit freiwillig zu Verfügung stellt, möchte er in vielen Fällen wissen wozu diese genutzt wird. Es gibt viele Interessenten, die dies gerne zur Berechnung von Primzahlen, Suche nach Außerirdischen oder der Krebsforschung tun. Handelt es sich jedoch um militärische Projekte, wie z.B. Antrax, ist dies oft nicht mehr gegeben. Es sind also Mechanismen notwendig, die es dem Worker überprüfbar machen, was er berechnet.

1.5.3 Fazit

Aufgrund dieser Überlegungen lassen sich folgende Haupteinschränkungen zusammenfassen:

- keine zeitkritischen Berechnungen, da mit einer verzögerten oder gar überhaupt keiner Antwort gerechnet werden muss
- möglichst keine privaten oder sonstigen sicherheitskritischen Daten dürfen übertragen werden

da es keine Bezahlung für den Worker gibt, muss ein öffentliches Interesse an der Lösung des Problems bestehen, und nur Leerlaufzeit des Rechners darf genutzt werden.

2 WADC oder Grid Computing?

In diesem Kapitel wird auf Gemeinsamkeiten und Unterschiede zwischen WADC und Grid Computing eingegangen und in diesem Zusammenhang werden wesentliche Merkmale des Grid Computing vorgestellt.

2.1 Wide Area Distributed Computing

Um WADC und GRID Computing vergleichen zu können, werden im Folgenden kompakt Vorzüge und Nachteile des erstgenannten erläutert.

2.1.1 Vorzüge

Der Vorteil des Wide Area Distributed Computing (WADC) Ansatzes liegt auf der Hand: Der arbeitgebende Master kann mit einer simplen und damit einfach zu realisierenden Architektur ein – je nach Projektgröße – mehr oder minder mächtiges Rechenpotential günstig bis kostenlos nutzen.

Das bekannteste und größte WADC Projekt ist SETI@home. Es erreicht, nicht zuletzt dank immer noch hoher Nutzerzuwachsraten von durchschnittlich 2500 neuen Nutzern pro Tag (!), ein Rechenpotential, dass das der schnellsten Supercomputer der Welt bei weitem übertrifft. So stellt die Community der SETI@home-Nutzer Ihrem Projekt durchschnittlich etwa 50-60 TeraFLOPs/sec unentgeltlich zur Verfügung, während der Earth Simulator, der momentan schnellste Supercomputer über eine Leistung von etwa 36 TeraFLOPs/sec verfügt. Noch deutlicher wird der Abstand, wenn man sich die Ausnahmestellung des Earth Simulators unter den Supercomputern verdeutlicht: Der zweitschnellste Supercomputer (der IBM „ASCI Q“) erreicht „nur“ etwa 14 TeraFLOPs.

Das enorme Potential der WADC-Projekte wird dabei wie bereits erwähnt einerseits durch neue Nutzer, andererseits aber auch durch immer schneller werdende Hardware auf Client-Seite weiter erhöht.

2.1.2 Nachteile

Das größte Problem des WADC liegt in der Unzuverlässigkeit der für das Projekt rechnenden Clients. Mag dies zunächst noch einfach klingen, so umfasst der Begriff Unzuverlässigkeit bei genauerer Betrachtung doch mehrere Facetten. So kann es vorkommen, dass Clientrechner fehlerhafte Rechenergebnissen zurückliefern. Dabei ist es egal ob diese Fehler aus einem Hardware- oder Softwareproblem resultieren. Als häufige Fehlerursachen zeigen sich Bedienungsfehler beim Einstellen von Hard- und Software, besonders absichtliches Overclocking ist in diesem Zusammenhang gefährlich, weil es ja das Ziel von WADC-Projekten ist, die volle Prozessorleistung auszuschöpfen und es bei diesen Dauerlastzuständen leichter zu Fehlern an getunter Hardware kommt als im normalen Teillastbetrieb. Denkbar wäre allerdings auch eine absichtliche Manipulation von Rechenergebnissen zum Zwecke der

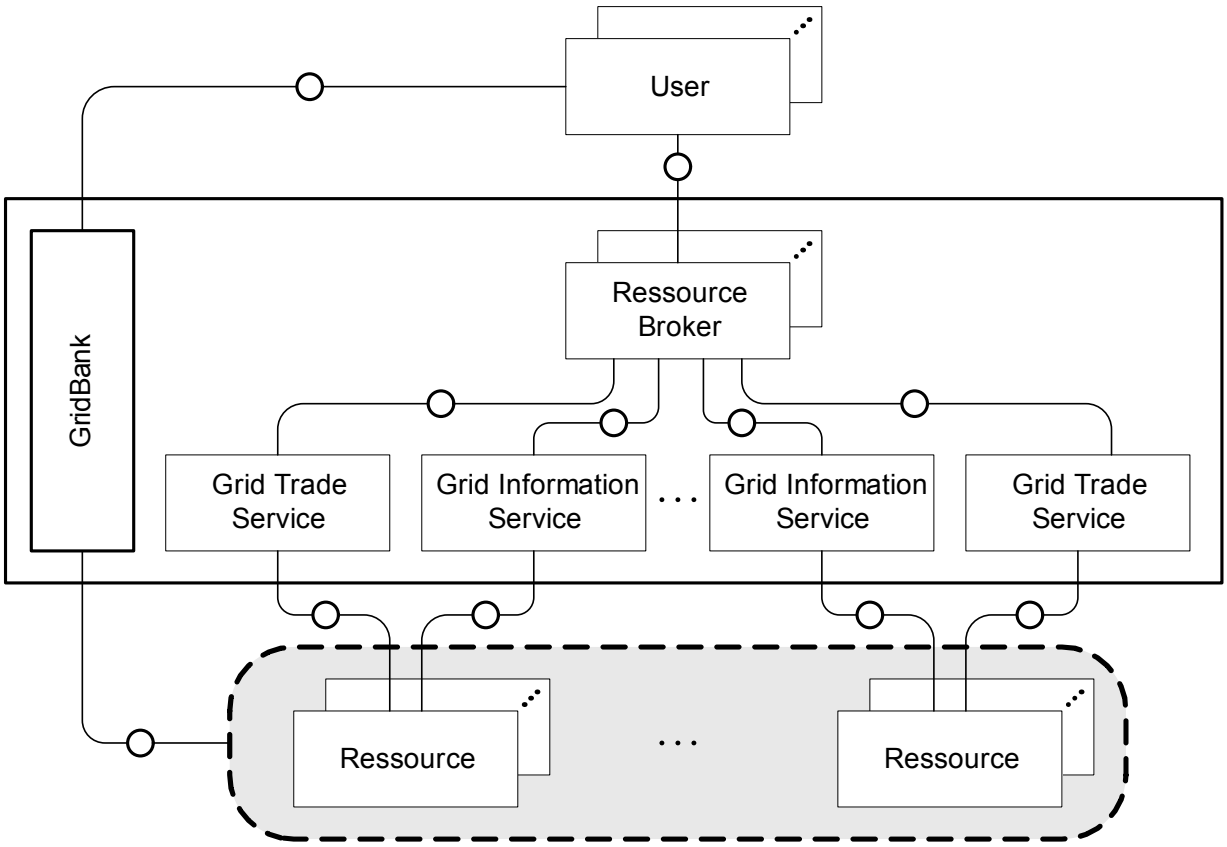
Projektsabotage. Selbst die Übermittlungs- und Steuersoftware des WADC-Projekts kann als Fehlerquelle nicht ausgeschlossen werden.

Ein weiterer Unzuverlässigkeitsfaktor liegt in der zeitlichen Dimension. So ist nicht sicher gewährleistet, dass eine zur Bearbeitung ausgegebene Teilaufgabe jemals gelöst wird. Wird die Teilaufgabe nach einem Timeout erneut vergeben, so kann es durch starke Verzögerung beim Lösen der ersten Version sein, dass sie dennoch mehrfach gelöst und beantwortet wird. Allerdings wird in der Praxis meist ohnehin mit Redundanz gearbeitet, hauptsächlich um durch Abgleichen der Antworten korrupte Lösungen zu erkennen und auszufiltern.

Ein weiterer Nachteil von WADC ist, dass eine kommerzielle Nutzung der WADC-Idee kaum möglich ist. Die Nutzer, die dem Projekt ihre freie Rechenpower kostenlos zur Verfügung stellen, möchten ja ein gewisses Interesse an der Erreichung des Projektziels haben, etwa die Erforschung von Pharmazeutika oder das Finden von extraterrestrischem Leben. Die monetäre Abrechnung des auf der Usermaschine genutzten Rechenpotentials wurde – zumindest bisher – noch von keinem WADC-Projekt erfolgreich umgesetzt. Ein Anreiz wäre es vielleicht, wenn die Prozessorleistung nicht stets in eine Richtung (vom Projekt-Teilnehmer zum Master) fließen würde, sondern wenn dem steten zur-Verfügung-stellen von Leistung auch einmal die Möglichkeit der kurzzeitigen Nutzung fremder Maschinen gegenüber stünde. Auf diese Grundidee setzt das sogenannte Grid Computing auf.

2.2 Beispiel einer Grid Computing-Architektur

Das Grid Computing selbst soll aufgrund der gegebenen Aufgabenstellung hier nicht allzu eingehend beleuchtet werden. Ein Vergleich zur WADC-Architektur ist aber durchaus interessant, weshalb folgende Architektur im folgenden als Grid Architektur bezeichnet wird.



Ziel ist die Verwaltung eines Pools zur Verfügung stehender Ressourcen, wobei die dargestellte Strukturvarianz bedeutet, dass durch Zu- und Abschalten stets neue Ressourcen hinzukommen oder bestehende wegfallen können. Stellt sich nun für einen beliebigen User eine Aufgabe, die viel Rechenaufwand benötigt, so kann dieser jederzeit zusätzlich zu seinen lokal vorhandenen Ressourcen über das Grid zusätzliche Rechenpower nutzen. Dieser Gedanke, der dem Konzept einer Steckdose ähnelt, über die theoretisch unendlich viel Rechenpower zur Verfügung steht, gab dem Grid Computing seinen Namen (engl. *power grid* = Stromnetz). Dabei wird das „woher“, wie beim Stromnetz quasi ausgeblendet und es interessiert nur, *dass* stets genug Leistung aus der Dose bezogen werden kann. Auf die im Grid vorhandenen Dienste, wie *Ressource Broker*, *Grid Trade Service* und *Grid Information Service* sowie deren Aufgaben soll erst später kurz eingegangen werden. Die Aufgaben und Arbeitsweisen der *Grid Bank* werden im weiteren Verlauf dieses Dokuments ebenfalls noch eingehender untersucht.

2.3 Abgrenzung WADC gegen Grid Computing

Das Konzept, rechenintensive Aufgaben durch Partitionierung und Verteilung zu lösen, ist beiden Ansätzen gemein. Die Wege, dies zu erreichen, sind jedoch bei WADC und Grid Computing höchst unterschiedlich. Daraus resultiert, dass die Art

der Aufgaben, die sich mit den beiden Konzepten praktikabel lösen lassen, stark unterscheiden und bei genauer Betrachtung sogar disjunkte Mengen darstellen.

Während es beim WADC einen klar zu definierenden Auftraggeber gibt, der viel Rechenleistung benötigt, so ist es das Ziel von Grid, eine Art Marktplatz für Rechenleistung zu schaffen, auf dem in einem bestimmten Rahmen – zumindest theoretisch – jeder Teilnehmer entweder Rechenleistung anbieten oder einkaufen kann. Außerdem ist es möglich, dass ein- und dieselbe Partei bzw. ein- und derselbe Nutzer sowohl Rechenleistung über das System bezieht als auch anbietet. Dabei ist es egal, ob beides gleichzeitig oder zeitlich getrennt voneinander passiert. Aus diesem Umstand resultiert auch das zweite Unterscheidungsmerkmal beider Ansätze. So sind die verteilten Aufgaben beim WADC in der Regel einem einzigen großen Problem zuordenbar (siehe auch „Wozu die große Rechenleistung?“), während es beim Grid Computing zwangsläufig keinen großen Zusammenhang zwischen den verteilten Rechenaufgaben gibt.

Bei Wide Area Distributed Computing ist es nötig, dass die verteilte Rechenaufgabe in irgendeiner Weise die Interessen der Nutzer anspricht oder für diese ein Anreiz zum Mitmachen besteht, da die freiwilligen Teilnehmer kein Geld für die bereitgestellte Rechenzeit erhalten. Dies ist bei Grid nicht nötig, da sich das „Einspeisen“ von Rechenzeit ins Grid für den User in der Regel lohnt. Wenn es keinen monetären Ausgleich gibt, so doch zumindest entsprechende Nutzungsrechte, falls der User selbst einmal Rechenzeit auf den Maschinen anderer Grid-Teilnehmer verbrauchen will.

Da es also nur ein sekundäres Interesse des WADC-Teilnehmers gibt, nutzen derartige Projekte standardmäßig auch nur die Idle Time der CPU des freiwilligen Teilnehmers, während beim Grid Computing im Allgemeinen vorgesehen ist, leistungsfähige Hardware einzig und allein dem Grid zur Verfügung zu stellen. Daraus ergeben sich – abhängig von der Größenordnung des Grids bzw. der Anzahl der Teilnehmer am WADC-Projekt – klare Performance-Vorteile für den Grid-Ansatz. Dieser Vorsprung wird noch dadurch vergrößert, dass wirkliche, intelligente Ressourcenplanung nur beim Grid möglich ist, bei dem die zur Verfügung stehenden Ressourcen an jeder Stelle zentral bekannt sind und – mehr oder weniger – zentral verwaltet werden. Beim WADC laufen hingegen identische Tasks auf jedem mitrechnenden, anonymen Client, dessen Ressourcen nicht einmal grob eingeschätzt werden können. Ob ein alter, ausrangierter PC oder eine neue, hochmoderne Maschine eingesetzt wird, ist nicht bekannt.

Neben den im folgenden Kapitel beschriebenen, noch ungelösten Problemen ist gerade auch diese komplizierte Architektur ein Grund dafür, dass Grid Computing noch nicht seine Marktreife erreicht hat, während der WADC-Ansatz bereits in vielen Projekten (siehe auch die Beispiele aus Kapitel 3) genutzt wird.

Die wichtigste Triebfeder für die Entwicklung des Grid Computing ist die – oben bereits kurz erwähnte –kommerzielle Nutzbarkeit, die dieser Ansatz dem WADC voraus hat. Dass die kommerzielle Nutzung von WADC trotz einiger Versuche bisher nicht funktioniert hat, liegt hauptsächlich an dem Umstand, dass die Maschinen, welche die Rechenleistung bereitstellen, anonym sind. Daraus ergeben sich Unsicherheiten bezüglich der Rechenzeit, die ein Arbeitspaket bis zu seiner Fertigstellung benötigen, weil nicht auszuschließen ist, das ein Nutzer seinen Rechner für längere Zeit abschaltet oder auch einfach den Client des jeweiligen

Projekts nicht startet oder gar deinstalliert. Außerdem birgt die Anonymität der Clients die Gefahr, dass zu Manipulationszwecken absichtlich falsche Daten zurückgeschickt werden. Sind vertrauliche Daten zu berechnen, so ist ohne aufwendige Verschlüsselung ebenfalls einleuchtend, dass WADC nicht die ideale Plattform darstellt.

Zusammenfassend ist festzustellen, dass Grid Computing dabei hilft, eine bedarfsgerechte Bereitstellung von Rechenleistung für ressourcenintensive wissenschaftliche oder kommerzielle Anwendungen zu gewährleisten, wobei auf viele bereits in WADC erprobte technische Lösungsansätze zurückgegriffen wird. WADC kann in gewisser Weise als Testfeld für Grid Computing gesehen werden. Die Ziele sind zu einem gewissen Grad übereinstimmend: Die schnelle Lösung rechenintensiver Aufgaben durch Verteilung. Andererseits bestehen beim WADC aufgrund des Gesamtkonzepts Einschränkungen hinsichtlich des Einsatzgebiets.

2.4 Zentrales Problem: Ressourcen-Management

Da sich die Hauptvorteile des Grid Computing Ansatzes überwiegend aus der zuverlässigen Verwaltung der zur Verfügung stehenden Ressourcen ergeben, ist offensichtlich, dass auch genau an dieser Stelle die wichtigsten Probleme dieser Technologie gelöst werden müssen. Während einige Entscheidungen bezüglich des Ressourcen-Managements bereits in der Phase der Planung und des Entwurfs des Grids getroffen werden müssen, können einige Einstellungen oder Policies noch nach der Implementierung des Systems angepasst werden.

Zunächst einmal wichtig ist die Frage, wer überhaupt als Anbieter von Rechenleistung in einem Grid zugelassen werden soll. Die Antwort ergibt sich fast zwangsläufig aus den Aufgaben, die mit Hilfe des Grids gelöst werden sollen. So spielen in diese Entscheidung natürlich Aspekte hinein wie die Vertraulichkeit der verteilten Aufgaben und Daten oder etwaige Anforderungen an die Qualität und Zuverlässigkeit der Berechnung.

Des weiteren ist es wichtig, Mechanismen zum Auffinden der benötigten Ressourcen zu spezifizieren. Dabei sind nicht nur die Kriterien, anhand derer potenziell nach Ressourcen gesucht werden soll, wichtig. Es muss auch sichergestellt werden, dass diese Mechanismen nicht an der Dynamik scheitern, denen der Pool der Ressourcen durch Hinzukommen neuer oder durch Wegfallen bestehender Ressourcen unterworfen ist. In den betrachteten Beispielarchitekturen übernehmen diese Aufgabe die *Grid Information Service* genannten Directories, in denen alle benötigten Informationen über verfügbare Ressourcen so erfasst sind, dass bei Bedarf eine Suche über alle möglichen Auswahlkriterien erfolgen kann.

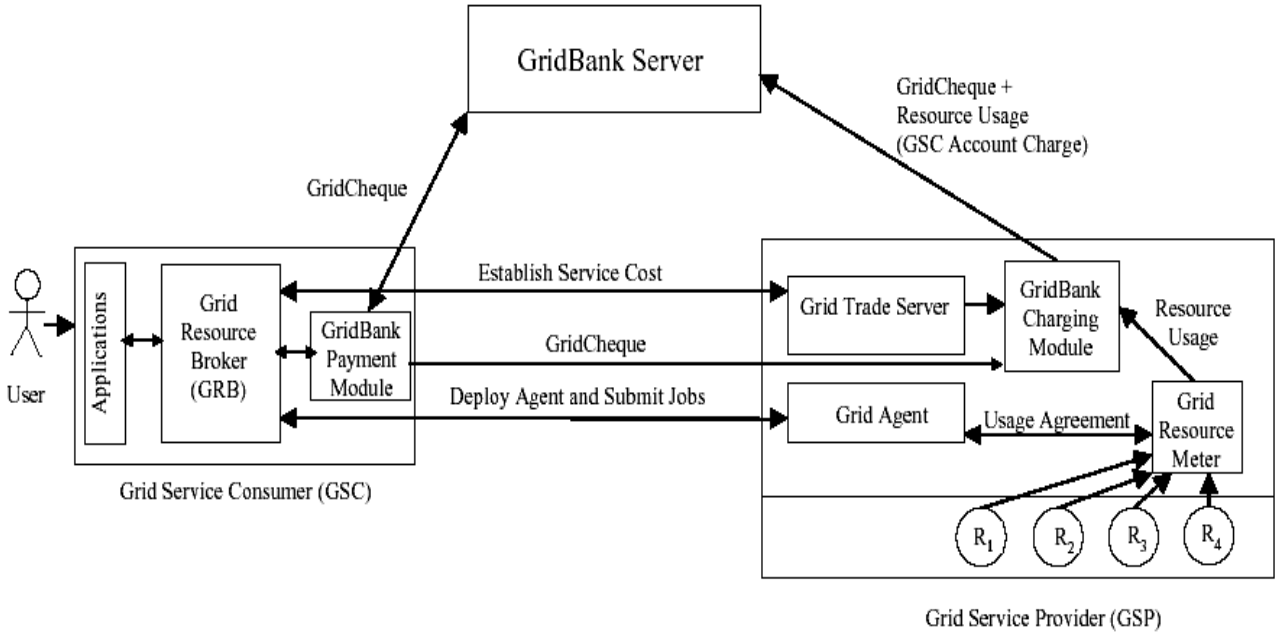
Was passiert nun im konkreten Fall einer Anforderung von zusätzlicher Rechenpower über das Grid? Zunächst wird der User seine Anforderungen an die gesuchte Dienstleistung spezifizieren müssen. Diese Spezifikation enthält Angaben wie harte Deadlines zur Lösung des Problems, Aussagen zu Quality of Service Anforderungen oder Kostenlimits. Auch weitere Einschränkungen bezüglich der Eigenschaften des Dienstleisters sind denkbar. All diese Anforderungen fließen dann zu einem Ressource Broker, der Zugriff auf das bzw. die Grid Information System(e)

hat und entsprechend eines statischen Regelwerks das beste aus den exakt zu den Suchkriterien passenden Angeboten auswählt und an den User zurückliefert. Alternativ können – je nach Konfiguration des Regelwerks – auch Alternativen zu dem bestpassenden Angebot zurückgeliefert werden oder auch nur das Angebot das, wenn nicht 100 Prozent, dann wenigstens die höchste Übereinstimmung mit den Suchkriterien bietet.

2.5 GridBank

Die kommerzielle Nutzbarkeit des Grid-Ansatzes erfordert ein Abrechnungssystem für verbrauchte bzw. erbrachte Dienstleistungen. Zu diesem Zweck wurde die GridBank entworfen, die eben diese Abrechnungen und ggf. auch eine Rechnungsstellung vornimmt.

Eine Testimplementation der [GridBank] wurde von Alexander Barmouta und Rajkumar Buyya von der Universität Melbourne entwickelt und entspricht der hier dargestellten Architektur:



Da es sich hier nur um eine Beispielimplementation handelt, soll auf die Details der Implementierung nicht näher eingegangen werden. Die Funktionsweise kann in ihrem Grobablauf aber durchaus als repräsentativ angesehen werden.

Dabei enthält diese GridBank nicht nur Mechanismen zur Bezahlung, sondern ebenfalls Tools, die es dem Grid Service Consumer (GSC) erlauben, die Abwicklung seiner Rechenaufgaben in der von Grid Service Provider (GSP) bereitgestellten Umgebung zu überwachen.

Die Schritte im Einzelnen:

Zunächst wird, wie im vorhergehenden Kapitel beschrieben, über den Grid Ressource Broker eine passende Ressource gefunden, die genutzt werden soll. Teil dieser Verhandlung ist immer auch das Entgelt, das für die Nutzung dieser Dienstleistung zu zahlen ist. Ob dieser Preis nun in monetären Einheiten gemessen wird oder in Credit Points, ist für die GridBank unerheblich.

Im nächsten Schritt fordert das GridBank Payment Modul des GSC einen GridCheque – ausgestellt auf den GSP – beim GridBank Server an. Dieser GridCheque wird ausgestellt, wenn der GSC genug Credits bzw. Geld bei der GridBank hat. Im nächsten Schritt leitet das GridBank Payment Module diesen GridCheque – noch ohne Angabe eines Betrags – direkt an das GridBank Charging

Module des GSP weiter. Nachdem damit die Bezahlung abgeschlossen ist, darf der GSC einen Grid Agenten in der vom GSP bereitgestellten Umgebung deployen. Dieser Agent hat zwei Funktionen: Einerseits dient er als Brückenkopf, über den die Daten, die zur Berechnung des Problems zum GSP übertragen werden müssen, gesendet werden. Bei geschickter Implementation können gerade hier auch Sicherheits- bzw. Vertraulichkeitsaspekte behandelt werden. Die zweite Funktion, die der Grid Agent bieten kann, ist die Überwachung der Abarbeitung der Rechenjobs.

Dabei werden auch die Daten des Grid Ressource Meter abgefragt und überwacht. Dieses zeichnet die Benutzung der Ressourcen auf, die zur Bearbeitung benötigt werden. Die entstehenden Ressource Usage Records genannten Statistiken werden auf Seite des GSP dazu verwendet, die vom GSC gebrauchten Ressourcen korrekt und genau über das GridBank Charging Modul abzurechnen. Dieses rechnet dann direkt mit dem zentralen GridBank Server ab, je nach Anwendungsfall entweder zyklisch oder am Ende des Nutzungsverhältnisses zwischen GSC und GSP.

2.6 Zusammenfassung

Was lässt sich nun als Resultat festhalten? Das Wichtigste ist vermutlich, dass das Grid Computing eine deutlich komplexere Technologie darstellt, die aber auch einen deutlich erweiterten Funktionsumfang gegenüber dem Wide Area Distributed Computing bietet. Dass sich WADC bereits im Praxiseinsatz befindet, unterstreicht die Feststellung, dass in WADC bereits viele Lösungen zu Problemen enthalten sind, die sich gleich oder ähnlich auch beim Grid Computing stellen. Die bei der Lösung gesammelter Erfahrungen helfen bei der Entwicklung der Grid Technologie, daher kann WADC in einigen Aspekten durchaus als Wegbereiter für Grid Computing gesehen werden. Bei diesen Gemeinsamkeiten darf jedoch nicht vergessen werden, dass Grid Computing und WADC auf sehr unterschiedliche Anwendungsbereiche zielen, obwohl beide die Idee teilen, komplexe Rechenaufgaben durch Verteilung schnell und effizient zu lösen. Beide versuchen dem Effekt der Verschwendung von Ressourcen entgegenzuwirken, auch wenn sich die Wege diesen Effekt zu erreichen (power-on-demand vs. vorhandene Leistung voll nutzen) unterscheiden.

3 WADC Projekte

Aus dem Bereich des Wide Area Distributed Computings (WADC) sind mittlerweile über 100 Projekte bekannt [1]. Die Projektziele sind unterschiedlichster Natur (Atom-/Elementarphysik, Krankheitsbekämpfung, Kryptographie, Mathematik, Suche nach Außerirdischen, ...). Auch durch andere Projektattribute wie Art der Daten, Größe des Projekts, Anzahl der unterstützten Computer-Architekturen oder das User-Interface des Clients sind die Projekte äußerst individuell.

Die nachfolgend näher betrachteten Projekte zählen zu den größten überhaupt und zeigen auf, was für unterschiedliche Ansätze für weitverteiltes Rechnen verwendet werden.

3.1 SETI@home

3.1.1 Projektbeschreibung

SETI@home [2] ist das größte und gleichzeitig das bekannteste Projekt für weitverteiltes Rechnen. Es startete Mitte 1999 und hat in den vergangenen 4 Jahren über 4,6 Millionen registrierte Benutzer gewinnen können. Täglich kommen 2500 neue Benutzer hinzu. Die durchschnittliche Gesamt-Rechenleistung liegt bei 50-60 TeraFlops, also deutlich über der von HPs Supercomputer ASCI Q (14 TeraFlops) und sogar vom „Earth Simulator“ (36 TeraFlops), dem aktuell mit Abstand schnellsten Supercomputer der Welt. [3]

Das Ziel dieses Projektes ist die Suche nach außerirdischer Intelligenz (SETI – Search for ExtraTerestrial Intelligence). Dazu werden Radiosignale aus dem All vom Observatorium in Arecibo aufgezeichnet und nach Signalen durchsucht, die nicht natürlichen Ursprungs sind.

Radio-Observatorium in Arecibo

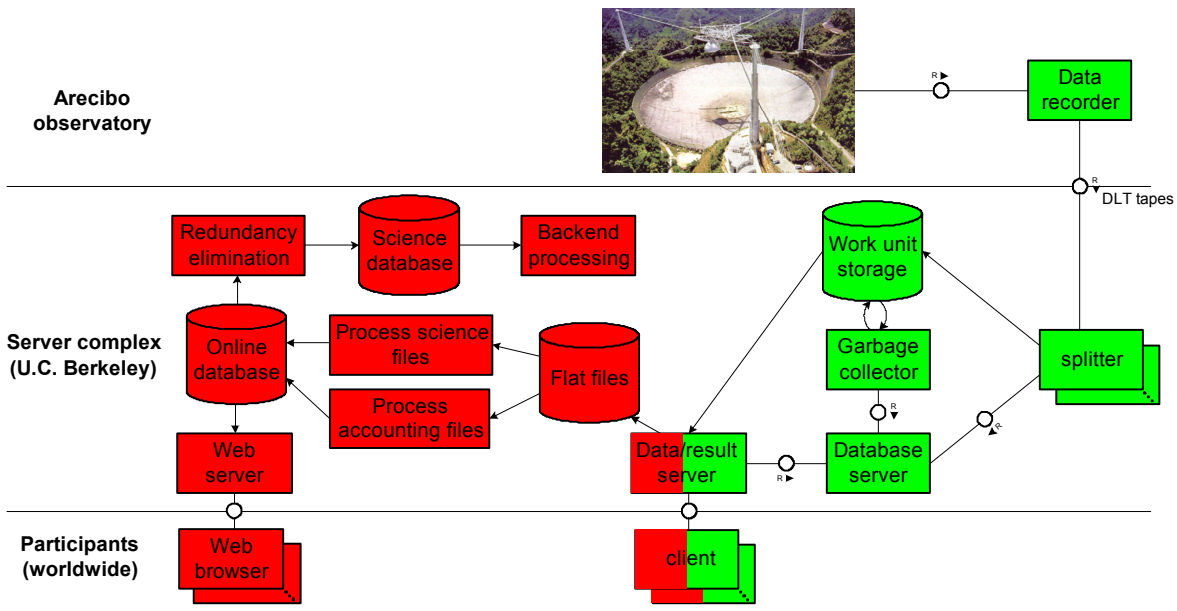
Die Kosten für diese Auswertung würde allerdings die dem SETI-Projekt zur Verfügung stehenden Mittel überschreiten, so dass an der University of California, Berkeley eine Software entwickelt wurde, die diese Radiodaten in Pakete unterteilt und sie an verschiedene Computer verteilen kann, die voneinander unabhängig auf den jeweiligen Daten Berechnungen durchführen können.

Durch die große Anzahl an freiwilligen Mitgliedern, welche die freie Rechenzeit ihres Computers zur Verfügung stellen, beschränken sich die Kosten auf Seiten von SETI@home auf die Aufrechterhaltung der Infrastruktur, Bandbreitenkosten und ähnliches.

3.1.2 Die Clientsoftware

Die einzelnen Datenpakete, die von den Clients vom Server abgerufen werden, enthalten die zu untersuchenden Radiodaten. Nach der Berechnung solch einer „Work Unit“, die durchschnittlich 10 Stunden dauert, werden die Ergebnisse wieder an den Server in Berkeley geschickt und ein neues Datenpaket abgeholt. Die Client-Software ist für eine Vielzahl an CPU-Typen (x86, PowerPC, MIPS, ...) und Betriebssystemen erhältlich. Es gibt sowohl grafische Versionen als auch solche für die Kommandozeile. Auch ist es möglich, SETI@home als Bildschirmschoner laufen zu lassen. Des weiteren gibt es eine Menge an Programmen, die zusätzliche Funktionalität anbieten, wie z.B. das Zwischenspeichern von mehreren Datenpaketen, damit man am Ende der Berechnung nicht online sein muss, um neue Arbeit zu erhalten. Andere Utilities erlauben das Verwalten einer großen Anzahl an Rechnersystemen in einem Netzwerk. Da gelegentlich ganze Firmen auf ihren Rechnern SETI@home laufen lassen, finden auch diese Programme ihre Nutzer.

Während die Datenpakete ca. 350 KByte groß sind, belegen die Resultate nur etwa 1 KByte. Die Bandbreitennutzung ist folglich sehr asymmetrisch. Insgesamt werden ca. 400 GByte pro Tag an Daten transferiert. Diese Bandbreite wird von der Universität zur Verfügung gestellt. Auch sonst ist SETI@home auf Sponsoring angewiesen, zum Beispiel für die benötigten Server.



Struktureller Aufbau des SETI@home - Netzwerks

3.1.3 Zuverlässigkeit bei Projekten weitverteilten Rechnens

Die Zuverlässigkeit einzelner Clients ist recht unkritisch, so werden viele angemeldete Benutzer sich schon nach kurzer Zeit aus dem Projekt zurückgezogen haben, teilweise schon während der ersten work unit. Sie brachten damit zwar keinen Nutzen für das Projekt, aber auch keinen bedeutsamen Schaden, da nach einer

gewissen Frist ein anderer Client mit der Berechnung des betreffenden Datenpakets beauftragt wird.

Die Erreichbarkeit des Servers hingegen ist von kritischer Wichtigkeit für das Projekt. Ein Ausfall bedeutet, dass keine Resultate mehr entgegengenommen und vor allem keine neuen Datenpakete mehr verschickt werden. Somit nimmt die Gesamtleistung im Laufe des Ausfalls rapide ab. Durch die Anforderung an eine hohe Verfügbarkeit ist es eine schwierige Aufgabe, das System aufzurüsten. Dies gilt insbesondere für Hardware, doch auch die Software muss so entwickelt worden sein, dass Updates im laufenden Betrieb bzw. mit möglichst kurzer Unterbrechung möglich sind.

Gerade am Anfang des Projektes gab es eine Vielzahl an Ausfällen, die größtenteils auf die Überlastung des Servers zurückzuführen sind, der die Masse an Abfragen nicht bewältigen konnte. Dies ist ein Problem, dass viele WADC-Projekte beim Start betrifft, insbesondere wenn sie auf einen Schlag bekannt werden und die bereitgestellten Ressourcen nicht ausreichen und die Systeme noch nicht optimiert werden konnten, da die dazu nötige Erfahrung sich erst im Laufe der Zeit einstellt.

3.1.4 Sicherheitsaspekte

Neben der Verfügbarkeit muss auch die Sicherheit des Systems bedacht werden. Da für die meisten Nutzer der Hauptreiz des Projektes in den Statistiken besteht, animiert dies auch zum Schummeln, um in den Bewertungen möglichst weit oben zu stehen. Man muss also dafür sorgen, dass Betrugsversuche weitgehend ausgeschlossen oder zumindest leicht aufzudecken sind. Dazu wird unter anderem eine work unit an drei bis fünf Benutzer versendet. Abweichende Resultate werden nicht weiterverwertet. Neben gewollter Verfälschung der Ergebnisse (z.B. um die Berechnung zu beschleunigen) kann dies auch einfach durch einen Anwendungsfehler geschehen. Dies ist in den meisten Fällen zurückzuführen auf einen Hardwaredefekt, da ein nicht unwesentlicher Teil der zur Berechnung verwendeten Systeme deutlich außerhalb ihrer Spezifikation betrieben werden („overclocking“), um eine höhere Performance zu erzielen.

Aber auch die Clients müssen gegen Angriffe abgesichert werden, da die große Anzahl an Benutzern eine attraktive Plattform zur Verteilung von Viren und anderen Schädlingen darstellt. So wurde kürzlich eine neue Version des Clients bereitgestellt, die eine Sicherheitslücke, ausgelöst durch einen buffer overflow, behebt. Würde jemand diese Schwachstelle ausnutzen, könnte er zum Beispiel die Millionen von Systemen dafür benutzen, eine verteilte Denial-of-Service-Attacke zu starten oder Spam zu versenden. Dies ist natürlich keineswegs im Interesse der Projektanbieter.

3.2 distributed.net

3.2.1 Projektbeschreibung

Bei distributed.net [4] kann man an mehreren Projekten für weitverteiltes Rechnen teilnehmen. Es besteht schon seit 1997 und ist das zweitfrüheste Projekt überhaupt – nach GIMPS [5]. Obwohl es demnach 2 Jahre älter ist als SETI@home, verfügt es nur über vergleichsweise wenige Benutzer. So waren beim Projekt RC5-64 etwas über 300.000 Benutzer angemeldet. Dies mag im eher rustikalen Aussehen des Clients begründet sein. Es ist lediglich eine Kommandozeilen-Version erhältlich. Allgemein spricht der Client eher „Computer-Enthusiasten“ an, da es im Gegenzug eine große Anzahl an Einstellungsmöglichkeiten gibt, um die Konfiguration den eigenen Bedürfnissen anzupassen. Des weiteren liegt der Quellcode offen (bis auf den Code für den Zugriff auf Dateien und Sockets). Der Client kann also auf weitere Plattformen portiert werden. Erhältlich ist er wie bei SETI@home für eine Vielzahl an CPU/Betriebssystem – Kombinationen.

Alle Projekte sind innerhalb eines Clients auswählbar, es handelt sich hierbei durchweg um stark mathematiklastige Projekte.

Die bei distributed.net bereitgestellten Projekte haben den Vorteil, dass dem Client nur ein Bereich übermittelt werden muss, in dem er suchen soll. Dadurch liegt der Bandbreitenbedarf erheblich unter dem vieler anderer Projekte.

Für Intranets wird eine Personal Proxy – Software bereitgestellt, mit deren Hilfe man die Work Units auf einen einzelnen Computer laden kann, zu dem dann alle teilnehmenden Rechner innerhalb des Netzwerkes Verbindung aufnehmen, um Arbeit abzuholen und Ergebnisse zurückzumelden. Dies erleichtert die Administration.

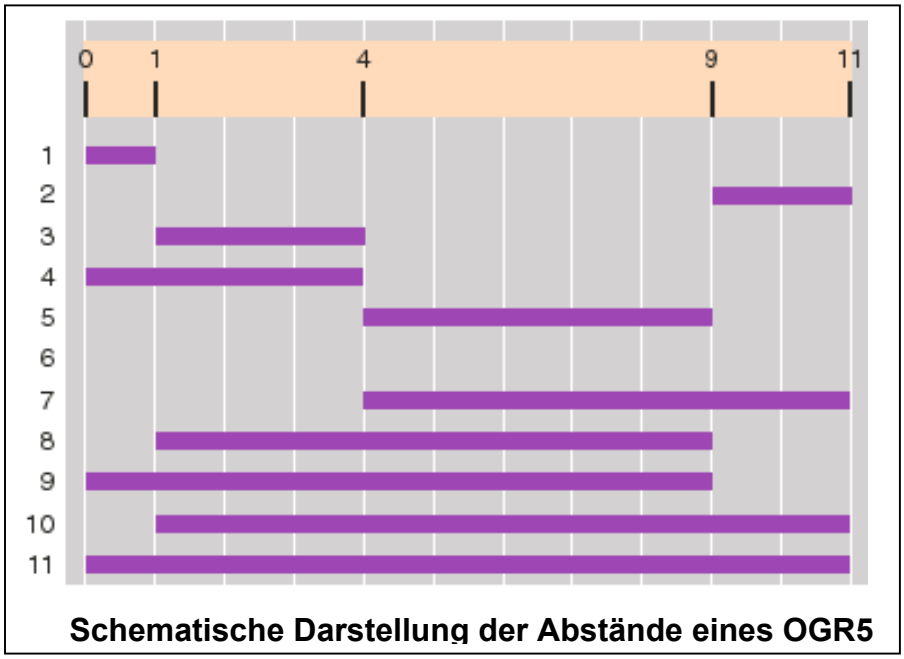
3.2.2 RC5

Bekanntestes Projekt ist RC5, ein Verschlüsselungsalgorithmus von RSA. Von dieser Firma stammt auch der Wettbewerb, Nachrichten mit verschieden starker Verschlüsselung zu entschlüsseln, um zu Werbezwecken die Zuverlässigkeit der eingesetzten Techniken zu demonstrieren. Die Verschlüsselungsstärke steigt jeweils in 8-Bit-Schritten von 40 Bit auf 128 Bit [6]. Für jede dekodierte Nachricht ist ein Preisgeld von 10000$ ausgesetzt.

Zur Zeit wird bei distributed.net ein Angriff auf den 72-Bit-Schlüssel durchgeführt, nachdem in der Vergangenheit schon RC5-56 und RC5-64 erfolgreich beendet wurden. Hierzu wird der Brute Force – Ansatz gewählt, also jeder mögliche Schlüssel durchprobiert. Es hängt somit größtenteils vom Zufall ab, wie lange die Entschlüsselung dauert. So wurde bei einem früheren Projekt (56-Bit-Schlüssel) der richtige Schlüssel nach 23 Stunden gefunden, RC5-64 hingegen dauerte knapp 4,5 Jahre, da der richtige Schlüssel erst gefunden wurde, nachdem 82% des gesamten Schlüsselraums durchsucht wurden.

Weitere bereits beendete Entschlüsselungsprojekte sind DES II-1, DES III und die CS-Cipher Challenge [7].

3.2.3 OGR



Schematische Darstellung der Abstände eines OGR5

Ein Projekt anderer Natur ist OGR24 bzw. OGR25. OGR ist die Abkürzung für Optimal Golomb Ruler. Ein Golomb Ruler stellt eine bestimmte Anzahl von Markierungen auf einem (ganzzahligen) Lineal dar, bei welchem der Abstand zwischen zwei Markierungen sich von allen anderen Abständen unterscheidet. Das Lineal heißt optimal, wenn es so kurz wie möglich ist. Während man bei einer Handvoll Markierungen auch noch ohne Computer die optimale(n) Lösung(en) finden kann, wächst der Aufwand, das OGR zu bestimmen, mit zunehmender Markierungsanzahl exponentiell, so dass es bei 24 bzw. 25 Markierungen bereits derart viele mögliche Kombinationen gibt, dass nur noch mit Hilfe der Rechenkraft von Tausenden von Computern Ergebnisse im Laufe der nächsten Jahre zu erwarten sind.

Golomb Ruler finden u.a. Anwendung in der Astronomie, Kombinatorik, Kodierungstheorie und Kommunikation.

3.3 United Devices

3.3.1 Projektbeschreibung

Bei United Devices [8] handelt es sich nach eigenen Angaben um eine „Serious commercial enterprise“. Beworben wird der sogenannte „MetaProcessor“. Als Grid-Technologie bezeichnet – obwohl es sich genaugenommen um Wide Area Distributed Computing handelt – kommt er in mehreren Ausführungen daher. Zunächst gibt es den „Enterprise MetaProcessor“, der firmenintern arbeitet und die firmenweiten Ressourcen (Rechenkraft, Anwendungen, Daten, Speicherkapazität, Netzwerk) verwalten.

Weiterhin wird „MetaProcessor On Demand“ angeboten. Dieser stellt einen Verbund von 8000 Computern mit einer Rechenkraft von 14 TeraFlops/sec dar, der von Unternehmen zur Erledigung rechenintensiver Aufgaben genutzt werden kann.

Der „Global MetaProcessor“ ist die 3. Variante. Dieser wird beworben als 2 Millionen Systeme in über 225 Ländern, die ihre ungenutzte Rechenkraft „qualifizierten Projekten“ zur Verfügung stellen. Aus einem Vergleich der angegebenen Informationen mit den Statistiken des Projektes für weitverteiltes Rechnen – am bekanntesten unter der Bezeichnung „Intel-United Devices Cancer Research Project“ – ergibt, dass es sich hierbei um ein und dasselbe handelt. Folglich bietet United Devices anderen Unternehmen an, deren Rechenjobs auf von Privatpersonen zur Verfügung gestellten Rechnern auszuführen. Derartige Projekte werden bislang allerdings nur selten in Anspruch genommen, und auch in diesem Fall gibt es nur 4 verschiedene Projekte, von denen nicht bekannt ist, inwieweit sie von einem anderen Unternehmen in Auftrag gegeben wurden. Häufig ist allerdings von beteiligten Unternehmen oder Institutionen die Rede.

3.3.2 Die Clientsoftware

United Devices wirbt für die Teilnahme am Projekt mit Schlagwörtern wie z.B. „PatriotGrid“ für das Anthrax-Projekt oder „combat bioterrorism“ beim Smallpox-Projekt. Der Client selbst ist nur für die Windows-Plattform erhältlich. Um den Anforderungen an kommerziellen Projekten gerecht zu werden, ist ein starke Verschlüsselung (128 Bit) integriert. Des weiteren kann sich der Client selbständig updaten, um etwa Sicherheitslücken zu schließen oder neue Projekte nachzuladen. Die Oberfläche selbst ist, wie bei SETI@home, farbenfroh gestaltet, der Client bietet auch eine ungefähr gleiche Einstellungsmöglichkeit. Zudem besteht die Möglichkeit, im Internet ein paar weiterführende Konfigurationen durchzuführen, um zum Beispiel die Berechnung auf einen gewählten Tageszeitraum zu beschränken. Bei der nächsten Verbindung des Clients mit dem Server wird diese Einstellung dann übertragen.

3.3.3 Interaktion mit anderen WADC-Projekten

United Devices war distributed.net im Jahr 2000 dabei behilflich, finanzielle Mittel zu erhalten und stellt darüber hinaus die Netzwerkverbindungen und das Hosting zur Verfügung. Im Gegenzug arbeiten 14 Mitarbeiter von distributed.net jetzt für United Devices, ihr ehemaliges Projekt führen die meisten von ihnen in ihrer Freizeit weiter. Da es bisher insgesamt 45 Mitarbeiter bei distributed.net gab, kann dies als weitreichende Übernahme der Entwickler angesehen werden, einschließlich deren langjähriger Erfahrung im Bereich des weitverteilten Rechnens. [9]

Des weiteren handelt es sich bei dem Chief Technology Officer (CTO) von United Devices um den ehemaligen Direktor von SETI@home.

3.3.4 Cancer Research Project

Bei dem ersten Projekt von United Devices in Zusammenarbeit mit der National Foundation for Cancer Research (NFCR) und dem Department of Chemistry der University of Oxford wird versucht, effektive Gegenmittel gegen vermutlich krebsverursachende Proteine zu finden. Dazu werden für jedes Protein mehrere Milliarden Moleküle virtuell auf Wechselwirkungen getestet.

Die ermittelten Ergebnisse werden zurück an den Server gemeldet und eine neue Work Unit besorgt, welche eine Größe von knapp über 50 KByte besitzt. Mittlerweile ist der erste Teil des Projektes abgeschlossen. Die vielversprechenden Ergebnisse aus dem ersten Teil werden jetzt mit einer anderen Anwendung ein weiteres Mal untersucht. Allerdings ist die letzte, das Projekt betreffende, Meldung schon über ein Jahr alt.

3.3.5 Web Performance Testing

Dieses Projekt verwendet nicht die Rechenkraft des Rechners, sondern dessen Internetanbindung, um die Erreichbarkeit diverser Webseiten zu testen. Durch die weite Verbreitung des Clients kommt es dadurch zu einer weltweiten Prüfung, die aufschlussreiche Resultate erbringt, um die Webanbindung zu optimieren. Hier ist ein direkter kommerzieller Zweck ersichtlich, da eine Überprüfung des Internetauftritts ohne großen Aufwand seitens des Auftraggebers durchgeführt werden kann und das Projekt nicht als Forschungsprojekt deklariert ist.

3.3.6 Anthrax Research Project

Das Anthrax Research Project ähnelt dem Cancer Research Project. Hier wird anstelle eines Gegenmittels für Krebs eines für Anthrax gesucht, ausgelöst durch die mit Milzbranderregern versehenen Briefe, die Anfang 2002 in den USA und anderen Ländern aufgetaucht sind.

Besonders brisant war in diesem Zusammenhang die Art der Projektverteilung, die automatisch an alle Clients erfolgte, die sich beim Server meldeten. Zudem wurde

das opt-out-Verfahren angewandt, man musste also explizit sagen, dass man an diesem Projekt nicht teilzunehmen wünscht. Allerdings wurde in den Projektbestimmungen festgelegt, dass die ermittelten Ergebnisse den USA sowie „allen befreundeten Nationen“ zur Verfügung gestellt werden sollten. Diesem Verwendungszweck hätten gerade viele nicht-amerikanische Teilnehmer nicht zugestimmt. Dieser Vorfall hat deutlich gemacht, dass es auch auf der Teilnehmerseite Sicherheitsbedenken gibt, dem das serverseitige Nachladen von Projekten entgegensteht.

Die Anthrax-Untersuchung wurde nach 24 Tagen abgeschlossen. Mit herkömmlichen Methoden hätte dieser Forschungsabschnitt mehrere Jahre gedauert.

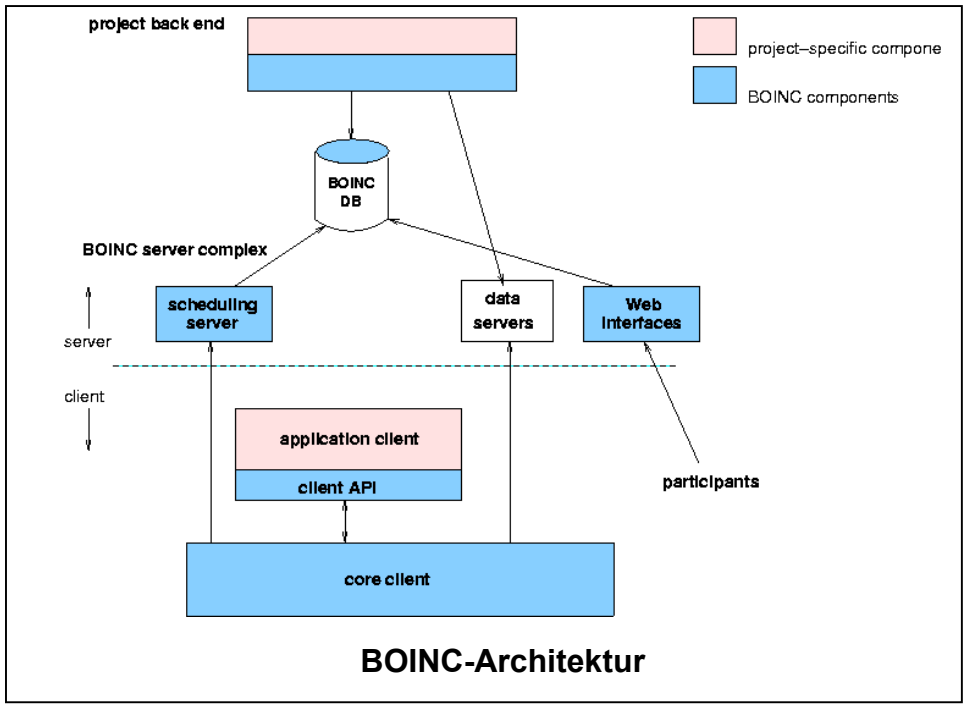
3.3.7 Smallpox Research Grid

Dies ist das 2. aktuell laufende Projekt von United Devices. Es wird versucht, ein Gegenmittel gegen Pocken zu finden. Es existiert zwar ein Impfstoff, dieser soll allerdings Nebenwirkungen haben. Außerdem lässt der Schutz mit der Zeit nach und braucht nach Verabreichung der Impfung einige Zeit, um das Immunsystem auf die Abwehr von Pockenerregern vorzubereiten. Für die Erforschung eines Gegenmittels sollen 35 Millionen Moleküle auf Wechselwirkungen getestet werden.

3.4 BOINC – Berkeley Open Infrastructure for Network Computing

Bei BOINC [10] handelt es sich um keinen WADC-Client, sondern um eine Softwareplattform für Projekte für weitverteiltes Rechnen, welche die Infrastruktur für Projekte bereitstellt. Für die Erstellung von Anwendungen wird ein Framework bereitgestellt, auf das mit den Sprachen C, C++ und Fortran zugegriffen werden kann. Bereits bestehende Projekte sollen sich mit nur geringen Modifikationen auf BOINC umstellen lassen. Die Verteilung von Updates wird ebenso übernommen wie die Authentifizierung mittels digitaler Signaturen. Des weiteren ist es möglich, mehrere Server aufzusetzen, sodass im Falle des Ausfalls bzw. Upgrades eines Servers die Kommunikation mit den Clients weiter aufrecht erhalten werden kann. Auch werden eine Vielzahl an Statistiken wie z.B. CPU-Last, Netzwerkbelastung und Größe der Datenbanktabellen zur Verfügung gestellt, um Flaschenhälse aufzuspüren.

Wie die nebenstehende Grafik zeigt, wird ein Großteil der erforderlichen Komponenten von BOINC bereitgestellt, das Projektteam muss nur noch den projektspezifischen Teil hinzufügen.



BOINC-Architektur

Ursprünglich wurde BOINC unter die Mozilla Public License 1.0 gestellt. Diese BSD-artige Lizenz erlaubt die Benutzung der Infrastruktur für private, öffentliche und kommerzielle Zwecke. Außerdem muss der Quellcode der Anwendungen nicht offengelegt werden. Als Ergebnis eines Rechtsstreits mit United Devices wird nun allerdings eine BOINC Public License verwendet, die als größten Unterschied die Verwendung zu kommerziellen Zwecken verbietet.

Nach langer Alpha-Phase ist Anfang September die Beta-Version von BOINC freigegeben worden. Das momentan einzige Projekt „Astropulse“ [11], das aufgefangene Radiosignale aus dem Weltall nach Pulsaren und schwarzen Löchern durchsucht, befindet sich allerdings in einem geschlossenen Beta-Test, so dass man zwar den BOINC-Client herunterladen kann, eine Erprobung von BOINC aber derzeit noch nicht möglich ist.

Im Oktober wird BOINC voraussichtlich offiziell starten. Das erste Projekt zu dem Zeitpunkt soll SETI@home sein, Astropulse folgt einige Monate später. Weitere Projekte, von denen bereits bekannt ist, dass sie BOINC nutzen werden, ist die SETI@home-Suche von der Südhalbkugel aus sowie Folding@home, welches zu den bekanntesten Projekten für weitverteiltes Rechnen gehört und eine Viertelmillion Nutzer zählt.

Nach dem offiziellen Start kann man davon ausgehen, dass umgehend weitere Projekte verfügbar sind, da BOINC die Erstellung und Wartung eines Projektes sowie die Suche nach Mitgliedern enorm erleichtert.

4 Praktische Arbeit – SoBSieveServer

4.1 Vorstellung des Hauptprojektes – Seventeen or Bust

Seventeen or Bust [12] startete im März 2002 als Zusammenarbeit von Louis Helm und David Norris, Studenten der Universitäten von Michigan respektive Illinois. Zusätzlich zu dem Projekt beigetragen hat insbesondere George Woltman, der das artverwandte GIMPS-Projekt ins Leben gerufen hat, mit der Bereitstellung dessen Codes.

Das Projekt, das mittlerweile fast 4500 Mitglieder zählt, hat es sich zur Aufgabe gemacht, das Sierpinski-Problem zu lösen. Dieses behandelt Zahlen der Form N = k * 2^n + 1, wobei k eine ungerade natürliche Zahl ist und n jede natürliche Zahl sein kann. N ist in diesem Fall eine Proth-Zahl. Sollte für ein gegebenes k die Zahl N für jedes beliebige n nicht-prim (also teilbar) sein, so handelt es sich bei dem k um eine Sierpinski-Zahl. Das Sierpinski-Problem besteht darin, herauszufinden, welches die kleinste Sierpinski-Zahl ist. Vor 40 Jahren wurde die Sierpinski-Zahl k = 78.557 gefunden, die kleinste bisher bekannte Sierpinski-Zahl. Um zu klären, ob es auch die absolut kleinste ist, muss entweder eine noch kleinere gefunden oder aber gezeigt werden, dass alle kleineren Zahlen das Sierpinski-Kriterium nicht erfüllen, es also ein n gibt, so dass N eine Primzahl ist.

Die 2. Möglichkeit wurde für Seventeen or Bust gewählt. Der Projektname rührt daher, dass es zu Beginn noch 17 Kandidaten gab. Über ein halbes Jahr war die Suche ergebnislos, dann wurden innerhalb von weniger als einem Monat 5 Primzahlen gefunden, die sich mit 200.000 bis knapp über 400.000 Dezimalstellen momentan auf Plätzen von 15 bis 52 der größten bekannten Primzahlen befinden. Seitdem gab es allerdings keine neuen Funde. Die nächste Primzahl wird mindestens 1,1 Millionen Dezimalstellen besitzen und damit Platz 3 einnehmen. Sollte es sich bei einer der verbleibenden Zahlen um eine Sierpinski-Zahl handeln, so ist dies wiederum nicht durch Seventeen or Bust bestimmbar. Vielmehr würde dies bedeuten, dass das Projekt ewig rechnen könnte, ohne abgeschlossen zu werden.

4.2 Unterprojekt: SoBSieve

Da der Aufwand für die Überprüfung auf Primalität von Zahlen dieser Größe auch für aktuelle Computer noch immens ist (Dauer: auf schnellen Systemen ca. 4 Tage, Aufwand steigt exponentiell zu n), kann man einen Teil der zu überprüfenden Zahlen durch ein Zahlensieb herausfiltern, indem man sie versucht durch kleinere Primzahlen zu teilen. Ist dies erfolgreich, kann es folglich keine Primzahl gewesen sein. Dies hat zusätzlich noch den Vorteil, dass dieses Ergebnis im Gegensatz zur Primalität in Sekundenbruchteilen nachgewiesen werden kann.

Aktuell werden n-Werte im Bereich von 300.000 bis 20.000.000 (entspricht ca. 100.000 bis fast 7.000.000 Dezimalstellen) überprüft, die verwendeten Teiler sind mittlerweile im Bereich von 65 Billionen. Durch Verbesserung der verwendeten Algorithmen werden auf schnellen Computern über 500.000 Teiler pro Sekunde getestet.

4.3 SoBSieveServer

4.3.1 Motivation

Obwohl das Sieving-Unterprojekt schon etliche Monate alt ist, muss immer noch ein nicht unwesentlicher Teil der Verwaltungsarbeit manuell erledigt werden. Der momentane Ablauf besteht aus

- zu untersuchenden Bereich in einem Forum per Posting reservieren,
- den Client starten und diesen Bereich eingeben und
- nach der Berechnung auf eine Webseite gehen und die Resultate dort in ein Formular einfügen.

Durch den großen Anteil an menschlichem Zutun ist es nicht möglich, den Client auf einem Computersystem laufen zu lassen, ohne sich regelmäßig um neue Arbeit zu kümmern. Bei einem einzelnen Arbeitsplatzrechner ist dies noch kein Problem, bei der Koordination einer Vielzahl von Systemen wächst der Aufwand hingegen schnell.

Des Weiteren kann es leicht zu Irrtümern kommen, wenn zum Beispiel ein bereits reservierter Bereich versehentlich erneut reserviert wird oder die Datei mit den gefundenen Faktoren gelöscht wird, bevor die Ergebnisse übertragen wurden. Beide Missgeschicke sind bereits mehrmals vorgekommen.

4.3.2 Aufbau des SoBSieveServers

Aufgrund dieser umständlichen und fehleranfälligen Implementierung haben wir es uns zur Aufgabe gemacht, die Verteilung von Bereichen und das Rücksenden von Resultaten zu automatisieren. Hierzu war auf der Clientseite ein Wrapper nötig, der die entsprechenden Variablen für den SoBSieve-Client setzt, ihn anschließend startet und darauf wartet, dass er seine Arbeit beendet. Daraufhin nimmt er die gefundenen Faktoren und übermittelt sie an den Server-Teil unseres Projekts. Dieser speichert die Faktoren sowie die Information, dass der Bereich bereits durchsucht wurde, in einer Datenbank und weist dem Client-Wrapper einen neuen, bislang noch nicht abgesuchten Bereich zu.

Da der SoBSieve-Client sowohl für Windows als auch für Linux erhältlich ist und andere Programme für denselben Zweck weitere Systeme wie z.B. Sun Solaris unterstützen, haben wir uns für Java als Programmiersprache entschieden. Somit reicht ein Wrapper für alle vorhandenen Programmversionen aus.

Der Server ist ebenfalls in Java geschrieben und greift auf eine MySQL-Datenbank zu. Diese Datenbank ist zentral installiert und verwaltet drei Tabellen:

- die Bereiche,
- die gefundenen Faktoren und

- zum Mappen des Benutzernamens auf projektintern verwendete User- und TeamIDs.

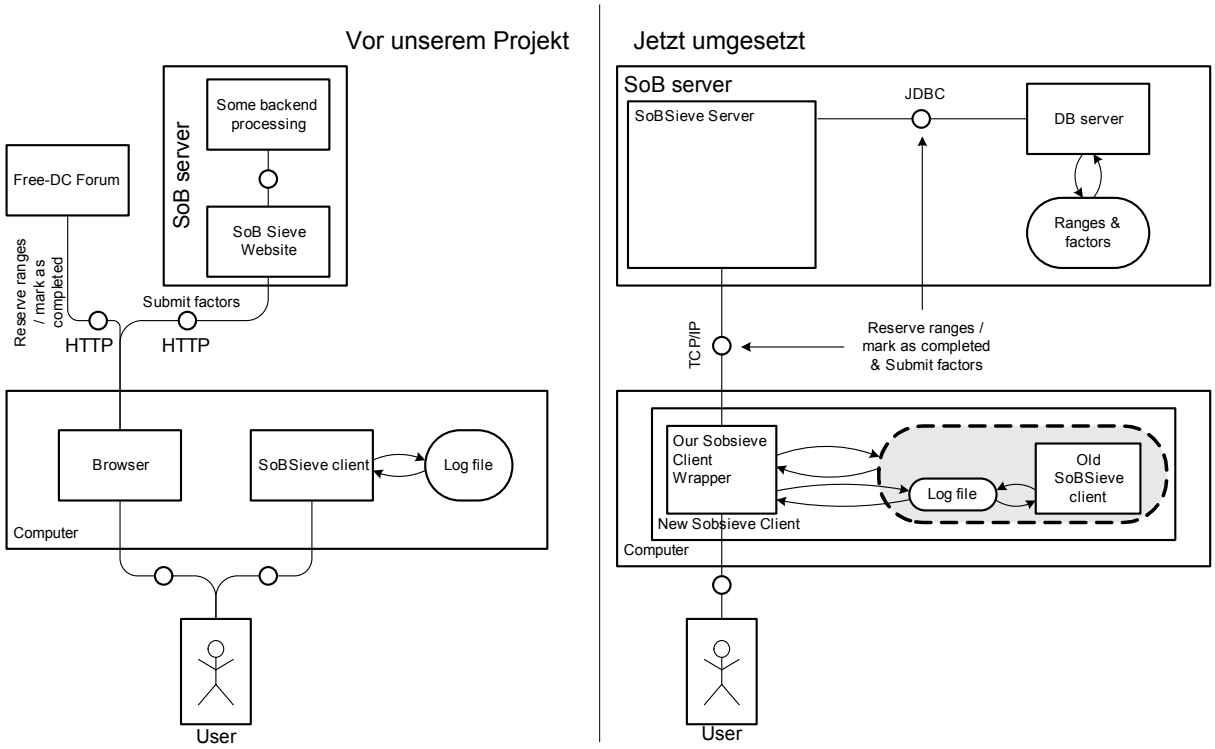


Abbildung: Aufbau vorher und aktuell

Server und Client kommunizieren über eine TCP/IP Verbindung. Wir haben auf diese Verbindung aufsetzend ein proprietäres Protokoll entwickelt, welches im Diagramm Ablauf im Gesamtsystem beschrieben wird. Sowohl Server als auch Client bestehen jeweils aus zwei Klassen, wobei in einer die Programmlogik/algorithmik und in der anderen die Behandlung der TCP/IP Schnittstelle und das Lesen und Schreiben des Logfiles implementiert ist.

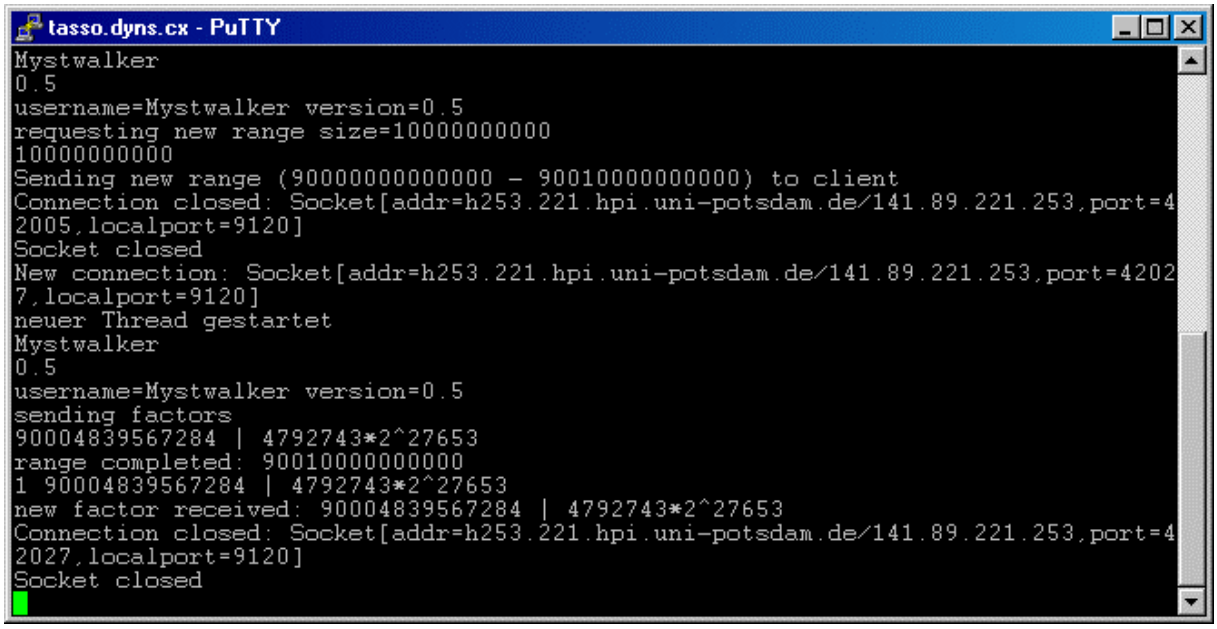


Abbildung: Ausgabe der Konsolenversion des Servers

4.3.3 Aktueller Stand

Bislang läuft das Projekt in einer Beta-Phase und nur mit einem der 3 vorhandenen Clients, mit einem öffentlichen Betrieb und der Erweiterung der Client-Unterstützung kann in den nächsten Wochen gerechnet werden.

Weiterhin besteht seitens des Projektleiters von Seventeen or Bust Interesse daran, das Projekt auch auf ein anderes Unterprojekt namens SBfactor auszudehnen, welches ebenfalls eine Faktorisierung versucht, allerdings nicht mit Hinsicht auf ansteigende Teilerwerte, sondern bezogen auf die zu faktorisierenden Zahlen.

Durch die relativ kleinen Ausmaße von Seventeen or Bust war es problemlos möglich, direkt mit den verantwortlichen Personen Kontakt aufzunehmen, die uns auch soweit wie möglich unterstützt haben, zum Beispiel durch leichte Anpassungen des SoBSieve-Clients, damit sich dieser von unserem Wrapper steuern lässt.

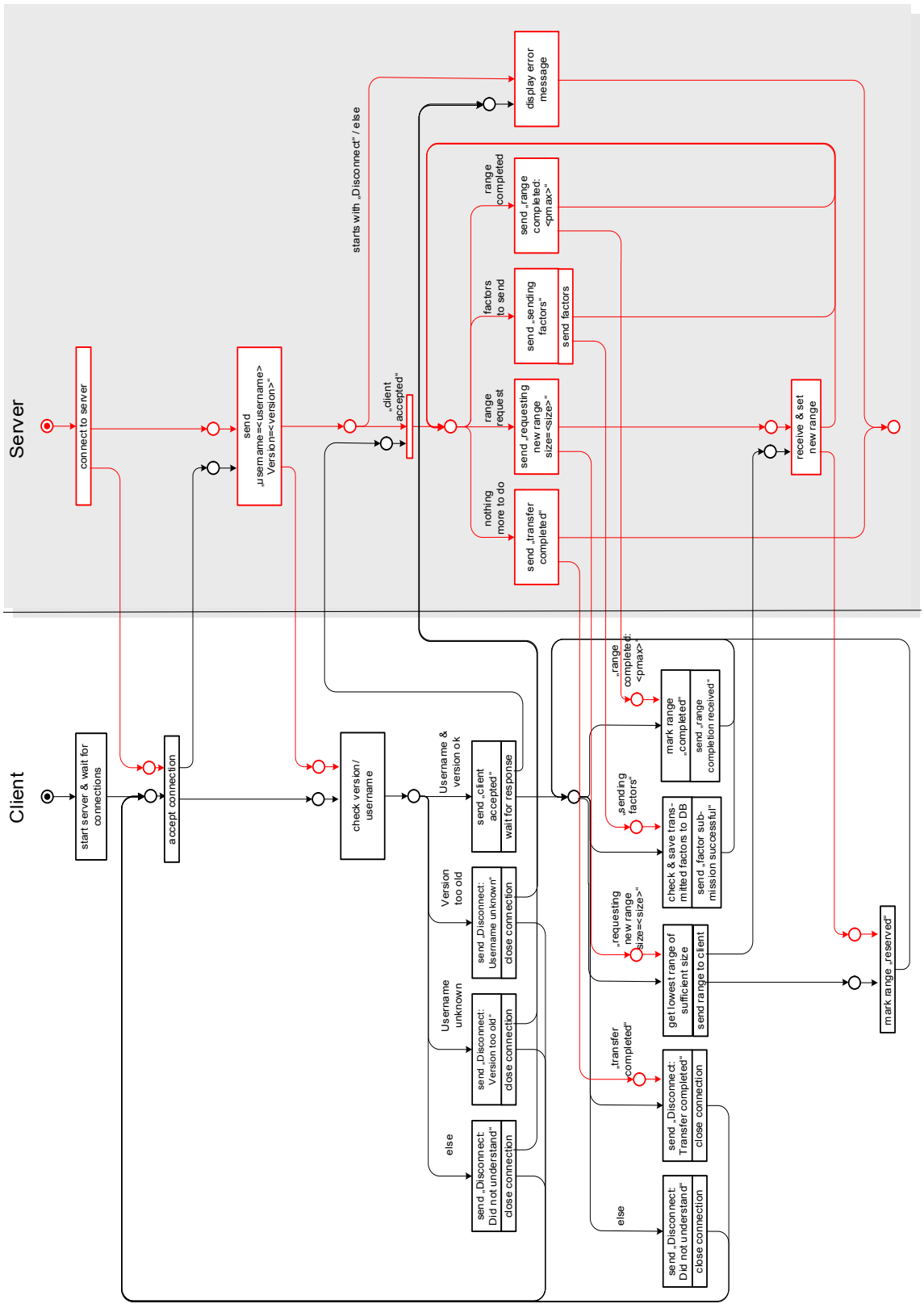


Abbildung: Ablauf im Gesamtsystem

Quellen

Die Quellenangaben sind nach Kapiteln unterteil – die Kapitelzahl wurde vorangestellt.

[1] Theo Ungerer et al 2002 - Systemnahe Informatik - Parallelrechner
[2] IX und Ct' Archiv 2001 – 2003 - diverse Artikel
[3] http://www.csse.uwa.edu.au/~barmouta/gbinstallation.html
[4] Rechenkraft.net, http://www.rechenkraft.net
[5] SETI@home, http://setiathome.ssl.berkeley.edu
[6] Top500 supercomputer sites, www.top500.org
[7] distributed.net, http://distributed.net
[8] Great Internet Mersenne Prime Search, http://www.mersenne.org
[9] RSA Laboratories | Secret-Key Challenge, http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html
[10] distributed.net: Projects, http://distributed.net/projects.php
[11] United Devices, http://www.ud.com
[12] Press release "distributed.net and united devices join forces", http://www.distributed.net/pressroom/news-20001127.html
[13] BOINC homepage, http://boinc.berkeley.edu
[14] Astropulse, http://maggie.ssl.berkeley.edu/ap/
[15] Seventeen or Bust homepage, http://www.seventeenorbust.com

Das Globus-Toolkit

Dietmar Bremser, Alexis Krepp, Tobias Rausch
dietmar.bremser@hpi.uni-potsdam.de, alexis.krepp@hpi.uni-potsdam.de,
tobias.rausch@hpi.uni-potsdam.de

Diese Ausarbeitung entstand im Rahmen des Seminares ”Grid-Computing”, gehalten im Sommersemester 2003 am Hasso-Plattner-Institut an der Universität Potsdam.
Das Globus-Toolkit ist nur eine von mannigfaltigen Möglichkeiten zum Aufbau eines Grids für kleine und große Netzwerke, beschreibt aber von der Gestaltung her eine grundlegende Methode zur Modellierung, dem Entwurf und der Implementierung einer Grid-Architektur.
Deshalb präsentieren wir hier zum besseren Verständnis zuerst den generellen Aufbau von Grid-Architekturen und deren kennzeichnende Merkmale.
Anschließend erfolgt eine genauere Betrachtung der Globus Toolkit Komponenten.
Den letzten Teil bilden Betrachtungen und einige praktische Hinweise zur Installation des Globus-Toolkits.
Für eine kurze Einführung des Globus-Toolkits und grafischen Darstellungen der Komponenten sei auf die begleitenden Folien verwiesen.

Einige Anmerkungen zur Notation:

- <**Term**> hebt einen wichtigen Begriff hervor
- <Term> stellt einen Zusammenhang dar
- <*Term*> hebt den Namen einer Datei oder eines Verzeichnisses hervor
- <`Term`> stellt einen Befehl in einem Kommandointerpreter dar, hier der `GNU Bash`

Inhaltsverzeichnis

1 Grundlagen **4**
1.1 Definitionen . . . 4
1.2 Ursprünge des Gridgedanken . . . 4
1.2.1 Teleologische Herkunft . . . 4
1.2.2 Wissenschaftliche Herkunft . . . 4
1.3 Unterscheidung von Grids . . . 4
1.3.1 Data Grids . . . 5
1.3.2 Computational Grids . . . 5
1.3.3 Provisioning Grids . . . 5
1.4 Applikationstypen für Grids . . . 5
1.4.1 Distributed Supercomputing . . . 5
1.4.2 High Throughput . . . 5
1.4.3 On-Demand Computing . . . 6
1.4.4 Data-Intensive Computing . . . 6
1.4.5 Collaborative Computing . . . 6
1.5 Grid-Architektur . . . 7

2 Einführung in das Globus-Toolkit **8**
2.1 Ein kurzer Überblick . . . 8
2.2 Lizenzmodell des Globus-Toolkit . . . 9

3 Komponenten des Globus-Toolkit **11**
3.1 Entwurfsprinzipien . . . 11
3.2 Aufbau und Komponenten . . . 12
3.2.1 Komponenten im Überblick . . . 12
3.2.2 Ressourcenverwaltung . . . 13
3.2.3 Kommunikation . . . 17
3.2.4 Information . . . 18
3.2.5 Sicherheit . . . 21
3.2.6 Fehlertoleranz-Dienst . . . 24
3.2.7 Remote-Data-Access . . . 25
3.2.8 Replica . . . 28
3.3 Ausblick . . . 29

4 Installation des Globus Toolkits **32**
4.1 Systemanforderungen . . . 32
4.2 Installationsablauf . . . 32
4.3 Installationsschritte zum Einrichten eines Servers . . . 33
4.3.1 Erstellen eines neuen Benutzers ”globus” . . . 33
4.3.2 Erstellen der Installationsverzeichnisse für GPT und Globus . . . 33
4.3.3 Einrichten der Toolkitumgebungsvariablen . . . 34
4.3.4 Installation des Grid Packaging Toolkits (GPT) . . . 34
4.3.5 Installation der Globus Distribution . . . 34

4.3.6 Verifikation der Installation 35
4.3.7 Installation der Simple Certificate Authority (SimpleCA) 35
4.3.8 Beantragen eines Benutzerzertifikates 36
4.3.9 Testen der Installation 37
4.3.10 Beantragen eines Host-Zertifikates 37
4.3.11 Starten der Dämonen 38
4.3.12 Test der Dämonen 39
4.4 Installation eines weiteren Clients 40
4.4.1 Installation eines zweiten Servers 40

Abbildungsverzeichnis

1 Der GRAM im Überblick als FMC-Aufbau-Modell 15
2 Der Aufbau der MDS-Teilkomponenten im Überblick als FMC-Aufbau-Modell 20
3 Der Aufbau der Sicherheitskomponente im Überblick als FMC-Aufbau-Modell 23
4 Der Aufbau des HeartBeat-Monitor im Überblick als FMC-Aufbau-Modell 25
5 Der Aufbau der GASS-Komponente in der gegenwärtigen Realisierung im Überblick als FMC-Aufbau-Modell 27
6 Der Aufbau der GASS-Komponente in der gewünschten Realisierung im Überblick als FMC-Aufbau-Modell 27
7 Der Aufbau des Replica-Dienstes im Überblick als FMC-Aufbau-Modell 29

Tabellenverzeichnis

1 Open-Source-Lizenzmodelle 10
2 Komponenten des Globus Toolkit 12

1 Grundlagen

In diesem Abschnitt geben wir einen kurzen Überblick über die Herkunft und den Aufbau von Grids.

1.1 Definitionen

Zu dem Thema Grid Computing existieren in der Literatur verschiedene Definitionen. Eine unserer Meinung nach sehr passende sei hier aufgeführt.

Definition (Übersetzung):

> Grid Computing ist das koordinierte, transparente und sichere gemeinsame Nutzen von IT Ressourcen geographisch verteilter Systeme.

Originaltext:

> ”Grid computing is the coordinated, transparent and secure sharing of IT resources across geographically distributed sites.” [10].

1.2 Ursprünge des Gridgedanken

Die Herkunft des Begriffes Grid lässt sich sowohl aus teleologischer als auch wissenschaftlicher Sicht beschreiben.

1.2.1 Teleologische Herkunft

Als viel zitiertes Vorbild für die Konstruktion von Grids wird immer wieder die Stromindustrie als Beispiel herangezogen. Hier mussten heterogene Stromerzeugungsgeräte zum Zwecke der Ausfallsicherheit und der Kapazitätsauslastung miteinander verbunden werden, um die Erfüllung von Stromnachfrage über die eigene Kapazität hinaus oder die Weitergabe von Überkapazitäten zu ermöglichen.

1.2.2 Wissenschaftliche Herkunft

Aus wissenschaftlicher Perspektive ist ein Grid die Verbindung des Meta-Computing und der Netzwerkinfrastruktur.

1.3 Unterscheidung von Grids

Es werden je nach Anwendungsschwerpunkten verschiedene Unterscheidungen von Grids getroffen:[1]

[1] angelehnt an [9]

1.3.1 Data Grids

Diese dienen überwiegend der Speicherung und Verwaltung von ”Informationen” auf einer Vielzahl von lose gekoppelten Netzwerken. Insbesondere das Data-Mining, bei dem es darum geht, Informationen zu jedem Zeitpunkt von jedem Zugangspunkt aus unmittelbar zur Verfügung zu stellen, spielt dabei eine wichtige Rolle.

1.3.2 Computational Grids

Sie dienen der Nutzung von lokalen und/oder entfernten ”Netzwerkinseln”, um deren CPU- und Speicherkapazitäten, Eingabe-/Ausgabegeräte, sowie Dienste und Anwendungen lokal zur Verfügung zu stellen. Ein Beispiel sind rechenintensive Aufgaben wie seismische Ereignisse. Hierbei ist die Stückelung der Gesamtaufgabe in Teilaufgaben und die Verteilung dieser Teilaufgaben auf das Grid ein wesentlicher Punkt.

1.3.3 Provisioning Grids

Diese bieten die allgegenwärtige Möglichkeit, Netzwerk-, Speicher-, Rechenressourcen und Applikationen gemeinsam zu nutzen. Sie stellen eine Kombination aus den beiden obigen Varianten dar.

1.4 Applikationstypen für Grids

Ähnlich wie die Grids selbst können auch Applikationen für Grids differenzierter betrachtet werden. Grid-Applikationen werden in folgende Bereiche eingeteilt:

1.4.1 Distributed Supercomputing

Hier liegt der Fokus auf Rechenressourcen, bzw. der Ballung von Rechenleistung für Probleme, die nicht auf einem einzelnen Rechner gelöst werden können. Zumeist handelt es sich um seltene und teure Ressourcen mit tausenden von Knoten in einem heterogenen Netzwerk, in dem Co-Scheduling und latenztolerante Algorithmen eine wichtige Rolle spielen.
Beispiele:

- DIS (Distributed Interactive Simulation): Planungs- und Trainingstechnik des amerikanischen Militärs
- Simulation komplexer physikalischer Prozesse (z.B. stellare Berechnungen)

1.4.2 High Throughput

Hier liegt der Fokus auf der Zuweisung nicht genutzter Prozessorzeit. Es wird versucht, einen möglichst hohen Auslastungsgrad der sich im Grid befindenden Prozessoren zu erreichen. Meist werden eine hohe Anzahl lose oder gar unabhängig gekoppelter

Aufgaben verteilt.
Beispiele:

- Advanced Micro Devices (AMD): Nutzung in der Hochphase (”peak-design-phasis”) der Entwicklung der K6/7-Prozessoren
- Condor System (Universität Wisconsin): Verwaltung von hunderten von Rechnern über den Erdball, für Projekte wie beispielsweise der Simulation von Flüssigkristallen auf molekularer Ebene.

1.4.3 On-Demand Computing

Bei dieser Gruppe von Applikationen liegt das Hauptaugenmerk auf der zeitlich punktuellen Verfügbarkeit von Rechenleistung. Dies bezieht sich auf die Nutzung von benutzerseitig lokal nicht verfügbaren Rechenleistungen, Software, Datenquellen oder speziellen Messanlagen. Häufig stehen hier Kostenaspekte im Vordergrund.
Beispiele:

- NEOS, NetSolve: numerische Lösungssysteme zur Integration auf Rechnern von Endnutzern, um komplexe Berechnungen durchzuführen
- Aerospace Corp.: Datenverarbeitung von meteorologischen Satellitenaufnahmen

1.4.4 Data-Intensive Computing

Charakteristisch für diese Gruppe ist die Synthese (Neuverknüpfung) von Daten, welche lokal und inhaltlich verteilt sind.
Beispiele:

- Die Hochenergie-Physik generiert Terabytes an Daten, die nach elementaren oder interessierenden Ereignissen durchsucht werden sollen
- ”Digital Sky Survey”: Untersuchung von astronomischen Daten

1.4.5 Collaborative Computing

Hier liegt der Fokus auf dem Zeitverhalten, sowie der Erweiterung der ”human-to-human interaction”, im Sinne der gemeinsamen Nutzung von Rechenressourcen wie beispielsweise Datenarchiven und Simulationen.
Beispiele:

- BoilerMaker (Argonne National Library): Entwurf eines Systems zur Emissionskontrolle von industriellen Verbrennungsanlagen.
- NICE (Universität Chicago): Erzeugung und Erweiterung realistischer virtueller Welten von Kindern zum Zwecke der Unterhaltung und Bildung

1.5 Grid-Architektur

Ein Grid existiert in einer homogenen oder heterogenen technischen Umgebung, wobei sich die Umgebung in verschiedenen politischen und wirtschaftlichen Rahmenbedingungen befinden kann. In der Literatur wird folgende Unterscheidung getroffen:

1. ”End-Systeme”:
   Im Sinne der Rechnerarchitektur geringe Skalierungsmöglichkeiten und hohe Homogenität der Bestandteile. Signifikant ist hier der Aspekt der Rechenleistung. Zukünftige Rechenanforderungen bedeuten die Veränderung der Rechnerarchitekturen hin zu parallelen Architekturen und die Integration dieser Systeme in Netzwerke.

2. ”Clusters”:
   Im Sinne einer ”großen Einheit”, also einer Sammlung von Rechnern mit überwiegend gleicher Architektur und verschiedener Konfiguration, verbunden über ein lokales Netzwerk, welches von einer zentralen Instanz administriert wird. Signifikant sind hier Parallelismus und verteiltes Management der verschiedenen Ressourcen.

3. ”Intranet”:
   Im Sinne einer hohen Anzahl an Ressourcen, die sich im Regelfall im Besitz einer Organisation befinden, einer administrativen Kontrolle unterworfen sind und daher eine gute Abstimmung untereinander aufweisen. Signifikant sind hier die Aspekte der Systemheterogenität und der geographischen Verteilung der Teilnetze und Endsysteme und die damit verbundene Administration.

4. ”Internet”:
   Im Sinne eines sehr lose verkoppelten, überregionalen und heterogenen Systems. Signifikant ist hier der Aspekt des Mangels an zentralisierter Kontrolle aufgrund loser und dynamischer Bindung der Teilnehmer an das Netz. Dies stellt zugleich auch das interessanteste und komplexeste Einsatzgebiet für Grids dar.

2 Einführung in das Globus-Toolkit

2.1 Ein kurzer Überblick

Das Globus-Toolkit stellt ”Werkzeuge und Programmbibliotheken zur Zugriffssteuerung auf Supercomputer, enthält Komponenten für Sicherheit, Kommunikation, Informationsinfrastruktur, Fehlererkennung, Ressourcen-Management, Portabilität und Datenmanagement” zur Verfügung [2].
Eine wesentlich spezifischere Definition liefern die Entwickler des Globus-Toolkits selbst: ”Grids are persistent environments that enable software applications to integrate instruments, displays, and computational and information resources that are managed by diverse organizations in widespread locations. Grid applications often involve large amounts of data and/or computing and are not easily handled by today’s Internet and web infrastructures.” [6]

Zielsetzungen:

1. ”bag of services”: die Grid-Architektur definiert Basis-Dienste, einen Werkzeugsatz an ”low-level”-Diensten für Sicherheit, Kommunikation, Ressourcen-Ortung sowie -anforderung, Prozess-Verwaltung und Datenzugriff, welch wiederum der Implementierung von ”higher-level”-Diensten, Werkzeugen und Applikationen dienen, aber nicht einzelne Programmiermodelle vorschreiben
2. Grid-Applikationen benötigen Dienste jenseits der aktuell verfügbaren Technologien, also führt Globus Informationsdienste ein, welche die potentiell hohen Leistungsanforderungen unterstützen
3. Globus implementiert spezielle Mechanismen, die mit ”globus-fremden” Mechanismen koexistieren oder sie ersetzen können

Einordnung:
Grid innerhalb einer Internet-Architektur, d.h. eines großen Netzwerkes mit einer hohen Heterogenität und einer losen, dynamischen Verbindung der Teilnehmer

Beteiligte:

- Argonne National Laboratory
- Universität Chicago
- Universität Southern California
- Illinois Urbana Champaign
- NASA
- Information Power Grid Team (Entwickler)

Aktivitäten:

- Forschung:
  Untersuchung grundlegender Aspekte des Ressourcen-Managements, Datenmanagements und der Sicherheit sowie von Informationsdiensten
- Software-Werkzeuge:
  Produktion von Software-Prototypen für die Forschung bezogen auf die unterstützten Rechnerplattformen (Hardware und Betriebssystem)
- Testbeds:
  Planung und Aufbau großer Versuchsanordnungen für die eigene Forschung und den Einsatz der Wissenschaftler und Ingenieure
- Anwendungen:
  Produktion von Applikationen für das Grid

unterstützte Betriebssysteme:

- Linux/ FreeBSD (Intel, AMD)
- AIX, HP-UX (Sparc, IRIX, SGI)
- sonstige Unix-Derivate
- Windows (nur client-seitig)

Programmierplattform: Perl, C

2.2 Lizenzmodell des Globus-Toolkit

Eine Besonderheit des Globus Toolkits ist sein Lizenzmodell, weshalb es hier etwas ausführlicher beschrieben wird.
Diese Lizenz, genannt ”Globus Toolkit Public License” (GTPL), leitet sich aus der ”Netscape Public Licence” (NPL) ab, welche wiederum ein Derivat der ”GNU Public Licence” (GPL) ist, allerdings mit der Ausnahme, dass sie den Lizenzgebern andere Rechte als anderen Nutzern gewähren.
Die GPL hat sozusagen politische Ziele, da sie, neben dem üblichen Ausschluss der Haftung für Konsequenzen aus der Nutzung der Software, zusätzlich die Forderung enthält, dass alle Weiterentwicklungen und alle Programme, die in irgendeiner Form GPL-lizensierten Code enthalten, wiederum unter der GPL veröffentlicht werden müssen. Diese Einschränkung macht GNU-Software, insbesondere Bibliotheken, im wesentlichen für eine Verwendung im kommerziellen Umfeld unbrauchbar.

Die GTPL erlaubt hingegen die kommerzielle Nutzung der Software, so dass externe Entwickler auf dem Globus Toolkit basierende, kommerzielle Programme entwickeln, ohne eine Lizenzgebühr zu zahlen.
Allerdings behalten sich die Entwickler, also das Argonne National Laboratory, die Universität Chicago, die University of Southern California Information Sciences Institute sowie der US-Regierung, welche die Entwicklung massgeblich finanziell unterstützt hat, die intellektuellen und materiellen Eigentums- und Urheberrechte vor.

Trotz dieser Einschränkungen verweisen die Entwickler auf den oben benannten Haftungsausschluss.

Die Veröffentlichung der Software unter einer anderen Lizenz ist allerdings ebenso untersagt. Das Lizensierungsmodell zielte deshalb mit Erfolg auf die Heranbildung einer Entwicklergemeinde ab. Eine weitere Eigenart ist, dass Modifikationen an der Software des Globus-Toolkit lediglich in Form von Patches bereitgestellt werden dürfen, um eine unerwünschte Aufspaltung des Kernproduktes in Derivate zu unterbinden.

Die einzelnen, hier angesprochenen Lizenzmodelle sind hier in Anlehnung an das ”Open Source”-Handbuch [13] dargestellt:

Tabelle 1: Open-Source-Lizenzmodelle

Lizenz	**Verwendung in kommerzieller Software**	**Veränderungen frei**	**Veröffentlichung unter anderen Bedingungen**	**besondere Rechte für Lizenzinhaber**
GPL	Nein	Ja	Nein	Nein
LGPL	Ja	Ja	Nein	Nein
BSD	Ja	Nein	Nein	Nein
NPL	Ja	Nein	Nein	Ja
Public Domain	Ja	Nein	Nein	Nein

3 Komponenten des Globus-Toolkit

3.1 Entwurfsprinzipien

Das Globus-Toolkit unterscheidet zwischen

1. einem **lokalen Dienst**, der sehr einfach und systemspezifisch gehalten und an homogene Plattformen gerichtet ist, d.h. entweder auf einer speziellen Hardware-Plattform oder einem Pool an homogenen Systemen wie einem Cluster aufsetzt, sowie

2. einem **globalen Dienst**, welcher auf einem lokalen Dienst aufsetzt und damit die Interaktion der Komponenten des Globus-Toolkit auf lokalen und entfernten Ressourcen befördert.
   Dies bedeutet, analog zum lokalen Dienst, dass ein globaler Dienst auf die Heterogenität der Ressourcen und seine Portabilität gerichtet ist. Damit unterstützt er einerseits eine variable Bandbreite an heterogenen Systemen, ist aber auch unempfindlich gegenüber Veränderungen der darunter liegenden Infrastruktur.
   Ferner ist ein globaler Dienst deshalb komplexer strukturiert als ein lokaler.

Grundsätzlich können Dienste sowohl lokal als auch global auftreten. Eine bekannte Umschreibung dieser Unterscheidungen ist der Begriff einer Client-Server- Architektur, allerdings mit der Einschränkung versehen, dass Klienten auch als Server auftreten können.

Aufgrund der Integration von Globus in die Internet-Architektur wurde dessen Schichten-Modell adaptiert, d.h. das Toolkit ist softwaretechnisch logisch in eine Komponenten-Architektur unterteilt.
Beispiele:

1. für das ”Globus Resource Management” (GRAM)

   - auf der tiefsten Ebene sind die lokalen Dienste wie Condor, LSF oder LoadLeveler angesiedelt,
   - auf der höchsten Ebene sind die globalen Dienste des ”Resource Brokers” und der ”Resource co-allocation” angesiedelt und
   - die Vermittlung zwischen ihnen wird über das GRAM-Protokoll geführt

2. für die ”Globus Communication Services”

   - auf der tiefsten Ebene existieren die lokalen Dienste wie ”IP Message Passing” und ”Shared Memory ATM”,
   - auf der höchsten Ebene sind die globalen Dienste des ”Message Passing Interface (MPI)”, von CC++, HPC++ sowie CORBA angesiedelt und
   - die Vermittlung zwischen ihnen wird über das Nexus-Protokoll geführt

Die Schnittstellen zwischen den höheren und den niederen Diensten werden als durchlässig (”translucent”) bezeichnet.
Die horizontale Integration der Dienste bedingt aber nicht eine vertikale Integration der Dienste, d.h. dass Globus-Applikationen zwar auf höhere oder niedere Dienste zugreifen können, dann allerdings die benötigte Funktionalität selbst implementieren müssen. Dies resultiert aus dem Prinzip des **”bag of services”** und wird ferner damit begründet, dass Anwendungen, welche auf Leistungsfähigkeit angewiesen sind, eher auf die Dienste der lokalen Ebene zugreifen werden, auch bekannt als ”low-level-programming”. Dementgegen wird bei Anwendungen, welche auf Portabilität fokussieren und ”high-level-tools” genannt werden, die Frage nach der Leistungsausbeute bzw. -effizienz eher nachrangig sein. Dies impliziert auch die Anwendung von hochsprachlichen Programmiermodellen, das sogenannte ”high-level-programming”.

3.2 Aufbau und Komponenten

3.2.1 Komponenten im Überblick

Das Globus-Toolkit besteht im wesentlichen aus sieben Komponenten, welche hier tabellarisch mit einer kurzen Erläuterung ihrer Funktionen dargestellt sind.

Tabelle 2: Komponenten des Globus Toolkit

Dienst	**Akronym**	**Bedeutung**	**Beschreibung**
Ressourcenverwaltung	GRAM	Globus Resource Allocation Management	Ressourcen-Allokation und Prozessverwaltung
Kommunikation	Nexus	(Protokoll, Bibliothek)	Unicast- und Multicast- Kommunikationsdienste
Information	MDS	Metacomputing Directory Services	Verteilter Zugang zu Informationen über Struktur und Status der Rechner im Grid
Sicherheit	GSI	Globus Security Infrastructure	Authentifikation und damit verbundene Sicherheitsdienste
Fehlertoleranz-Dienst	HBM	HeartBeat Monitor	Überwachung des Zustandes der Systemkomponenten inklusive Ausnahmebehandlung bei der Prozessabwicklung
Fortsetzung auf nächster Seite			

Fortsetzung: Komponenten des Globus Toolkit			
Dienst	**Akronym**	**Bedeutung**	**Beschreibung**
Ferndatenzugang	GASS	Globus Access to Secondary Storage	Ferndatenzugang zu entferntem Hintergrundspeicher via sequentieller und paralleler Schnittstellen
”Executable Management”	GEM	Globus Executable Management	Konstruktion, Zwischenspeicherung (caching) und Ortung von ausführbaren Dateien (executables)

Das Globus-Toolkit verwendet und implementiert mit Erweiterungen im wesentlichen vier Protokolle ([10], Seite 10):

1. **HTTP** für die Ressourcenverwaltung, im Globus Toolkit als **GRAM-Protokoll (GRAM-P)** implementiert
2. **FTP** für den Datentransfer zwischen Sekundärspeichern, im Globus Toolkit als **GridFTP** implementiert
3. **LDAP** für die Informationsdienste, im Globus Toolkit als **Globus Resource Information Protocol (GRIP)** implementiert
4. **Nexus** für die Kommunikation bzw. Interaktion aller Komponenten des Globus-Toolkits auf der Transportebene der Netzwerkgeräte

Die ersten drei genannten Protokolle wurden gewählt, weil es sich bei ihnen um einen anerkannten und nicht proprietären Industriestandard handelt.
Die Autoren erlauben sich in diesem Zusammenhang und in Anbetracht der Komponenten die Vermutung, dass HTTP im wesentlichen für die rechenintensiven Leistungen eines Grids eingesetzt wird.
Dementgegen wurde FTP eher den datenintensiven Leistungen eines Grids zugeordnet.

In Folge werden nun der Aufbau und die Funktionsweise der Komponenten detaillierter beschrieben.

3.2.2 Ressourcenverwaltung

Die Ressourcenverwaltung, kurz der **GRAM**, fokussiert auf die Verwaltung der Ressourcen. Dabei ist der Term Ressource ein Sammelbegriff für Rechenmaschinen, Speicher, Sensoren und Netzwerke. Demzufolge stellt der GRAM einen Verwalter für

die lokal an einem Ort bzw. zu einer Domäne zusammengeschlossenen Ressourcen und kann weiterhin als Komponente auf der ”niedrigsten Ebene” (”low-level”) bzw. dem ”fabric-layer”[2] am nächsten angesiedelte Komponente verstanden werden. Der GRAM interagiert auf diese Weise direkt mit dem Scheduler der ihm zugeordneten Ressource, also entweder einem einzelnen Rechner oder auch einem Pool von Rechenmaschinen einer Organisation bzw. (Sub-)Domäne.

Bei dem GRAM wird der

- lokale Dienst vom **Job-Manager, GRAM-Reporter und Gatekeeper**,
- globale Dienst vom **Gatekeeper** mit Unterstützung durch den **Resource Broker** und dem **Resource Co-Allocator**

zur Verfügung gestellt.

Wie schon bemerkt, stellt die GRAM-Komponente eine einheitliche, netzwerkfähige Schnittstelle (API) auf Basis von HTTP bzw. GRAM-P zur Verfügung, um Zugriff auf die Werkzeuge der lokalen Ressourcen-Verwaltung und die verschiedenartigen Scheduler zu ermöglichen.
Deshalb implementiert der GRAM eine Spezifikationssprache zur Ressourcen-Verwaltung, genannt **Resource Specification Language (RSL)**. Die RSL zeichnet folgende Eigenschaften aus:

1. sie ist erweiterbar und einfach gehalten
2. sie ermöglicht generische Anfragen für die Ausführung eines Programmes im Grid, welche über die konkreten Exekutivbefehle des ausführenden Schedulers abstrahieren
3. sie ermöglicht die Anforderung von Ressourcen durch Angabe expliziter Kriterien für die Auswahl der benötigten Ressourcen durch den Benutzer, bspw. die gewünschte lokale Maschine im Grid, genauso wie
4. die Anforderung von Ressourcen mittels Angabe (impliziter) Rahmenbedingungen für die Auswahl der benötigten Ressourcen, bspw. der minimal benötigte Speicherplatz für die Ausführung eines Programmes oder die Anzahl der Ressourcen, auf denen die Kopien eines Programmes ausgeführt werden sollen;
   damit ist eine automatisierte Selektion der Ressourcen möglich, welche von den globalen Diensten des GRAM abgewickelt wird

Die RSL wird von den globalen Diensten des GRAM genutzt, welche die Übersetzung der eben genannten generischen Ressourcen-Anfragen in die konkreten Befehle der

[2] Der ”fabric layer” repräsentiert entweder die Betriebssystemschicht, welche den Zugang zur Hardware eines Rechners ermöglicht, beispielsweise der Scheduler oder der für den Zugriff auf den Hintergrundspeicher zuständige Betriebssystemdienst. Oder diese Schicht repräsentiert eine Abstraktion eines Betriebssystemdienstes für mehrere Rechenmaschinen, wie bspw. dem LoadLeveler, einem kommerziellen Job-Scheduler für ein Cluster an IBM-Rechenmaschinen. Siehe dazu [10], Seite 7

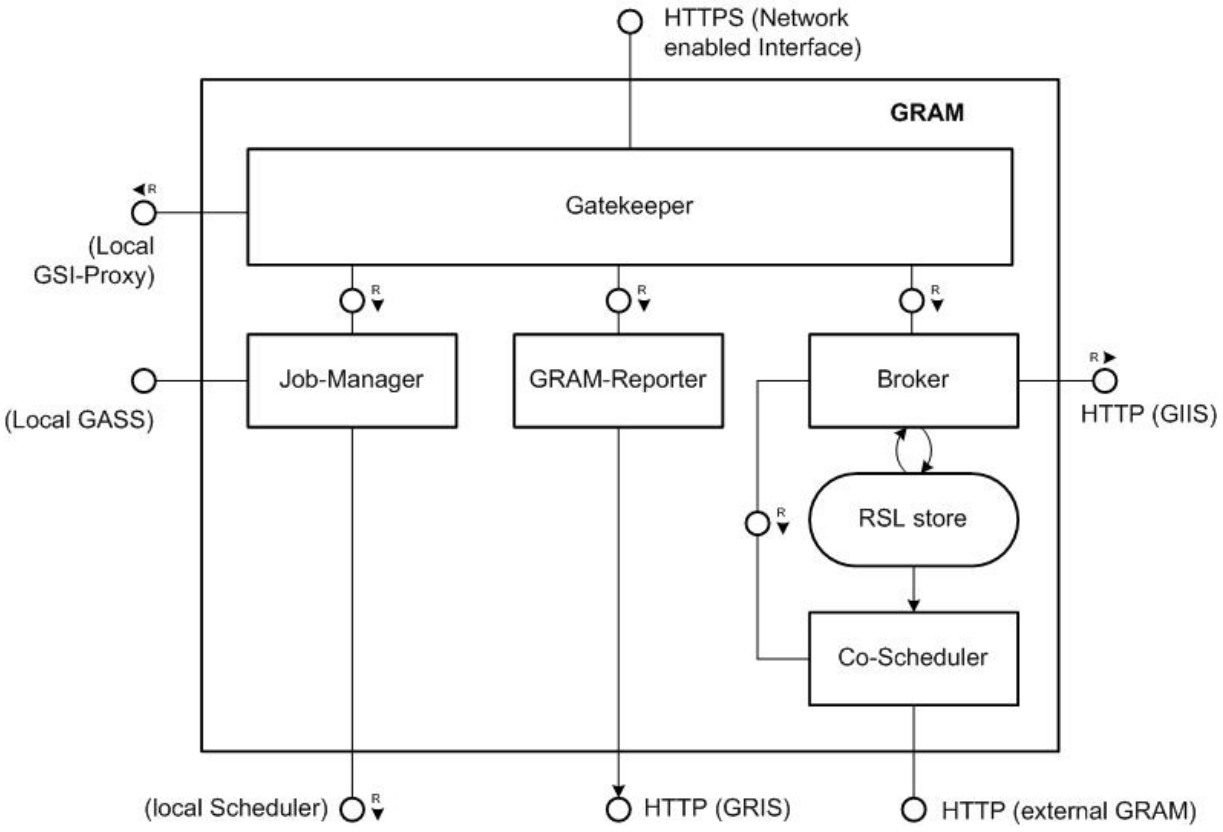


Abbildung 1: Der GRAM im Überblick als FMC-Aufbau-Modell

darunterliegenden Schichten übernehmen.
Eine generische Ressourcen-Anfrage, gehalten in RSL, hat beispielsweise folgende Erscheinung:

```
&(executable=ftp://host:1234/dir/myfile)
```

Der GRAM unterstützt die Erstellung und Verwaltung eines Satzes an Prozessen, sogenannten **Globus-Jobs**, auf einem Pool von lokalen Ressourcen.
Außerdem ist der GRAM verantwortlich für einen Satz bzw. Pool an Ressourcen, welche der gleichen (Sub-)Domäne und Allokationspolitik (”site-allocation-policy”) unterworfen ist. Das bedeutet, dass ein einzelner GRAM den Zugang zu einem Knoten an Parallel-Rechnern, einem Cluster von Workstations oder einer Sammlung an Rechnern innerhalb eines Condor-Pools zur Verfügung stellt.
Da der GRAM die kleinste, atomare Entität eines Grids darstellt, ermöglicht sie die Strukturierung der verbundenen Systeme. Ein Globus-Grid enthält damit mehrere GRAM, die wiederum einen lokalen Pool oder eine lokale Ressource verwalten.
Letztlich ist der GRAM auch für die Sammlung und Verwaltung von Informationen über die lokale Ressource zuständig.

Die eben beschriebenen Dienste werden von Teilkomponenten des GRAM übernommen, in Abbildung 1 veranschaulicht und im weiteren textuell beschrieben:

1. Der **GRAM-Reporter** ist die zuständige Teilkomponente für die Sammlung und Verwaltung von Daten und gibt diese Daten mittels des auf HTTP basierenden GRAM-Protokolles an den Informationsdienst des Globus-Toolkit, den MDS, weiter.

2. Der **Gatekeeper** ist die Benutzerschnittstelle des GRAM und läuft auf jeder Globus-Maschine.

   - Als lokaler Dienst (`globus-personal-gatekeeper`) dient er der
     i) Authentifizierung von lokalen Benutzern und
     ii) Aufrechterhaltung von Verbindungen des lokalen Benutzers zu entfernten Ressourcen für eine bestimmte Zeitdauer.
   - Als globaler Dienst (`gsigatekeeper`) ist er für
     i) die Akzeptierung von Verbindungen entfernter Benutzer zu der ihm zugeordneten lokalen Ressource,
     ii) die Authentifizierung entfernter Benutzer mittels einer Anfrage beim lokalen Sicherheitsdienst,
     iii) die Abbildung des entfernten Benutzers auf ein lokales Benutzerkonto mittels des *grid-mapfile* und
     iv) die Entgegennahme von Anfragen für einen Globus-Job

     zuständig.

3. Der **Job-Manager** interagiert mit lokalen Schedulern, d.h. startet auf Veranlassung des Gatekeepers einen Prozess (Globus-Job) und ist verantwortlich für die Allokation und Deallokation der Ressourcen für diesen Job. Standardmäßig startet der Job-Manager den `fork()`-Scheduler auf der lokalen Ressource, was die Ausrichtung des Globus-Toolkits auf UNIX-Systeme unterstreicht.
   In der Version $2.x$ des Globus-Toolkit dient der Job-Manager noch der Kommunikation mit dem Benutzer und ersetzt dabei den später eingeführten HeartBeat-Dämonen.

4. Der **Resource Broker**

   - implementiert die domänen-spezifische Suche nach Ressourcen und Auswahlstrategien in lokalen und entfernten Grid-Clustern mittels des globalen Informationsdienstes GIIS,
   - ist ein Dienst der höheren Ebene (”high-level-service”) und übersetzt daher die abstrakten bzw. generischen (RSL-)Anfragen der Applikationen in systemspezifische Aufträge, indem er iterativ die durch ihn verwalteten verwalteten Ressourcen abfragt, bis eine geeignete Ressource gefunden ist und
   - übergibt die übersetzte Anfrage an den Resource Co-Allocator.

5. Der **Resource Co-Allocator**

- ist verantwortlich für die Koordination von Ressourcen-Anfragen an entfernte Maschinen an verschiedenen Orten und verschiedenen Organisationen (”sites”) und damit letztlich für die Synchronisation der Programmausführung,
- implementiert verschiedene Ansätze zur Allokation und Verwaltung von ”Ensembles” an Ressourcen, also einer Sammlung heterogener Systeme, und
- übernimmt den Versand von in RSL formulierten Anfragen an entfernte GRAM.

3.2.3 Kommunikation

Der Kommunikationsdienst wird im Globus-Toolkit auf Basis von **Nexus** implementiert. Damit ist Nexus kein Dienst, sondern eine Kommunikationsbibliothek.
Nexus wurde eingesetzt, weil TCP/IP die Bedürfnisse einer Grid-Architektur nicht hinreichend unterstützt, also zu viele Verwaltungsdaten erzeugt werden. Ein weiterer Grund war die zu geringe Kontrolle des Verhaltens der physischen Netzwerkgeräte vermittels der rezent verfügbaren Technologien [9].
Zu bemerken ist, dass Nexus gemäß des Prinzipes der Globus-Mechanismen, die herkömmlichen Netzwerkprotokolle sowohl ersetzen, aber auch mit ihnen koexistieren kann.

Nexus definiert eine Programmierschnittstelle (API) zur ”low-level”-Kommunikation, die wiederum Hochsprachkonstrukte wie ”message passing”, ”remote procedure call” (RPC) oder ”remote I/O” ermöglichen. Nexus ist demzufolge kein Benutzerprotokoll.
Die Komponenten des Globus-Toolkit implementieren das Nexus-Protokoll deshalb selbst.

Die Stärke von Nexus liegt im wesentlichen in der Fähigkeit zur multi-mode-conversation, also der zeitgleichen Kommunikation - bspw. uni- oder multicast - über verschiedene Netzwerkgeräte und Protokolle wie PPP, TCP, ATM oder andere.
Zu diesem Zweck bietet Nexus fünf Abstraktionen für Kommunikationsobjekte:

1. ein **context** ist der (zusammenhängende oder partitionierte) Adressraum eines Prozesses, der wiederum einen oder mehrere . . .

2. **threads of execution** enthält,
   threads tauschen Informationen (Botschaften oder ”remote service requests”) aus → über einen . . .

3. **link**, welcher ein Kommunikationskanal der ”threads” ist,
   der Aufbau des Kommunikationskanal kann im Modus ”one-way”, ”two-way”, ”broadcast” oder ”gather” erfolgen → über einen . . .

4. **communication node**, welcher eine territoriale Abstraktionen der threads darstellt, da es sich in Grids um einen globalen Adressraum handelt, also eine Kollektion von mehreren lokalen Adressräumen, in denen zwei verschiedene Pro-

zesse auf jeweils verschiedenen lokalen Maschinen einen gleichen Adressraum bzw. eine gleiche Anfangsadresse haben können

5. **remote service request** ist ein einseitiger, asynchroner Funktionsaufruf an entfernte Prozesse (RPC), der im wesentlichen einen Auftrag zur Ausführung von Funktionen in ”prozessfremden” Adressräumen bedeutet;
   dagegen ist eine Botschaft nur auf den Austausch von Kontrollinformationen und/ oder globaler Adresszeiger auf ”context”-invariante Datenstrukturen gerichtet

Diese Abstraktionen können auf verschiedene Kommunikationsmethoden abgebildet werden, die jeweils verschiedene Leistungscharakteristiken, wie etwa Zuverlässigkeit, Sicherheit, ”Quality of Service” oder Kompression, zur Folge haben.

Ein Beispiel:
Eine Prozess bzw. Thread kann Verbindungen zu zwei anderen Prozessen bzw. Threads aufbauen, bei denen der eine Kommunikationskanal zuverlässig sein soll und der andere eine geringe Latenzzeit haben soll.
Dies bedeutet für die Frage nach der Leistungscharakteristik, dass bei einer geringen Latenzzeit eine unzuverlässige Kommunikation toleriert werden muss, wohingegen eine zuverlässige Kommunikation steigende Latenzzeiten bedingt.

3.2.4 Information

Die informationelle Infrastruktur wird vom **Metacomputing Directory Service (MDS)** realisiert.
Der MDS abstrahiert die physische Struktur und die virtuelle Organisation des Grid mittels einer Datenbank an sogenannten ”records of information”. Da aber überwiegend nur ein lesender Zugriff auf die Informationen stattfindet, genügte die Implementation eines Protokolles anstatt eines komplexen Datenbank-Verwaltungs-Systemes.
Dieses Protokoll basiert auf dem **LDAP**-Protokoll der Version 3. Bekanntermaßen hat LDAP seine Wurzeln im ”Directory Access Protocol” der Universität Michigan, das im Protokoll X.500 beschrieben wurde.
Zudem ist das ”Lightweight Directory Access Protocol” (LDAP) seit der Verabschiedung im ”RFC 2252” ein De-Facto-Standard zur Repräsentation von Informationen jeglicher Art, sofern sich diese hierarchisch darstellen lassen.
LDAP spezifiziert demzufolge eine Repräsentation dieser Informationen in Form eines hierarchisch baumstrukturierten Namensraumes (”directory-information tree”) und definiert ein Netzwerk-Protokoll für den Zugriff auf die in den Verzeichnissen verfügbaren Informationen.
Zudem eignet ist LDAP wegen der möglichen Assoziation der Teilbäume des Verzeichnisses mit verschiedenen Servern für verteilte Dienste besonders gut geeignet.
LDAP ermöglicht Anfragen an den Verzeichnisbaum auf zwei Arten[3]:

[3]Die Anfragen haben ihre teleologische Herkunft aus den Telefonbüchern, den ”weißen” und den ”gelben” Seiten.

1. white-pages-request,
   eine Anfrage, für welche der Inhalt eines Blattes des Verzeichnisses, also die Information über ein konkretes Verzeichnisobjekt, zurückgeliefert wird, wie z.B. die IP-Adresse einer Ressource[4]

2. yellow-pages-request,
   eine Anfrage, welche mit Informationen über den Knoten und damit einen Teilbaum des Verzeichnisbaumes beantwortet wird[5], wie z.B. die Liste aller Maschinen mit mehr als 42 Prozessoren[6]

Das Globus-Toolkit erweitert LDAP vor allem um Sicherheitsmechanismen und wird dann **Globus Resource Information Protocol (GRIP)** genannt.
Auf diese Weise beinhaltet der MDS Werkzeuge und eine API zur Sammlung, Verarbeitung und Veröffentlichung statischer sowie dynamischer Metadaten über die Struktur und den Status des Grid. Zudem ermöglicht der Dienst korrekte Ressourcenanfragen, die für die oben besprochene Komponente GRAM von elementarer Bedeutung sind.

1. Statische Daten sind weitestgehend unveränderliche Informationen wie der Typ der Prozessor-Architektur, das Betriebssystem, die Größe des primären oder sekundären Speichers, die Netzwerkbandbreite, die Version des eingesetzten Globus-Toolkit, der lokale Scheduler oder Kontaktdaten der Organisation.

2. Dynamische Daten betreffen dagegen Latenzzeiten, Auslastung der Rechner, alloziierte Prozessoren, verfügbare Kommunikationsprotokolle oder die Abbildung der IP-Adressen auf die Netzwerkgeräte.

Die MDS-Infrastruktur wird in der Version $2.x$ des Globus-Toolkit auf

- globaler Ebene mittels des **Globus Institution Indexing Service (GIIS)**,
- lokaler Ebene mittels des **Globus Resource Information Service (GRIS)** und dem GRAM-Reporter

bereitgestellt.

Das Globus-Toolkit der Version $2.x$ koppelt die Mechanismen der Sammlung und Veröffentlichung von Daten gänzlich voneinander ab und realisiert diese in den oben genannten Komponenten [4].
Auf diese Weise ist der **GRIS** nur für die Sammlung der Informationen über die verfügbaren Ressourcen, wenn sich diese im Grid anmelden, zuständig.
Diese Informationen erhält der GRIS mittels eines ”push” vom GRAM-Reporter via HTTP.

[4] Eine gängige Erklärung ist die Suche einer Person in einem Telefonbuch mit ”weißen Seiten”.
[5] Eine gängige Erklärung ist die Suche nach Anbietern einer bestimmten Leistung in einem Telefonbuch mit ”gelben Seiten”.
[6] Die Autoren wissen, dass derartige Maschinen wohl kaum existieren dürften. Aber einige Fragen sterben wohl nie.

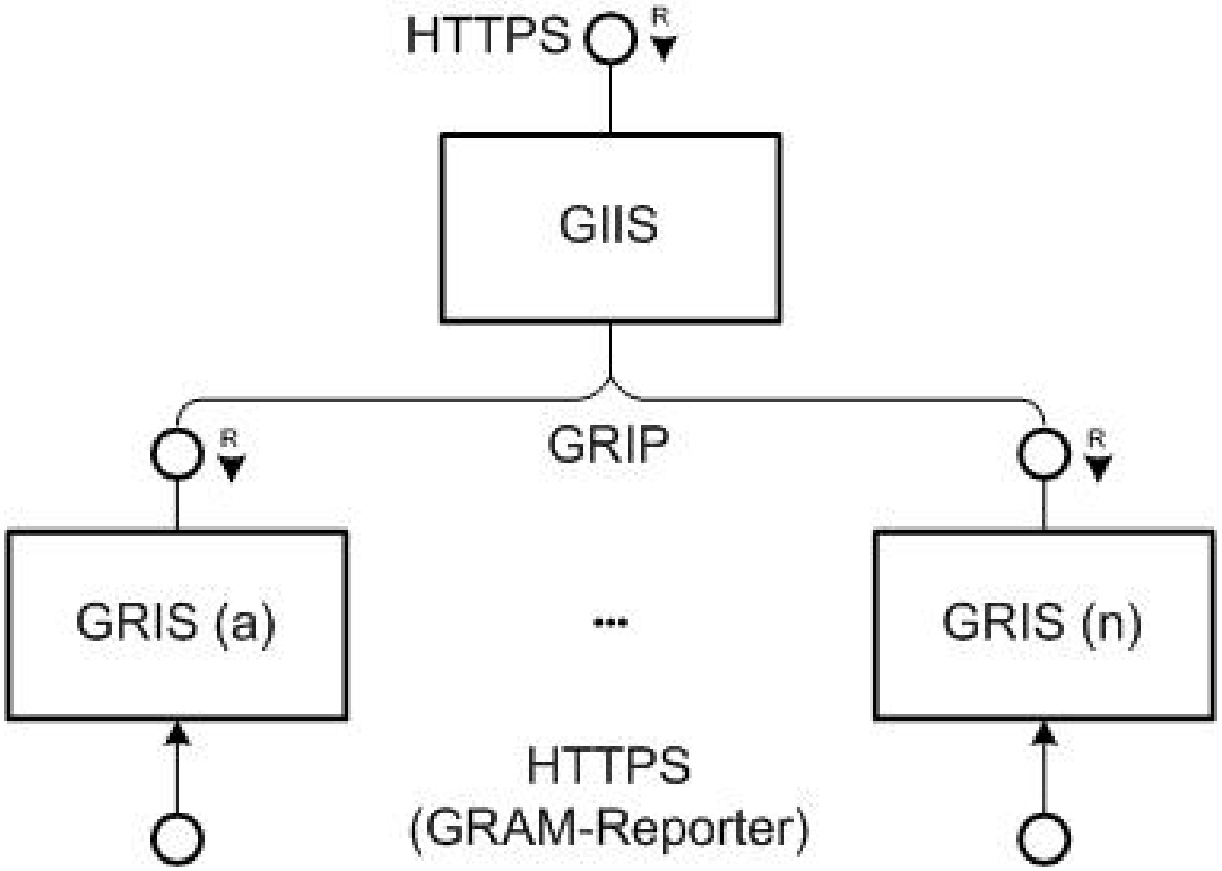


Abbildung 2: Der Aufbau der MDS-Teilkomponenten im Überblick als FMC-Aufbau-Modell

Neben den Informationen über die Ressourcen, die ein GRAM verwaltet, erhält der GRIS auch die Zeitstempel des Gatekeepers des entsprechenden GRAM, also den Zeitpunkt der Initialisierung und der voraussichtlichen Terminierung der Ressource bzw. des Gatekeepers.

Der **GIIS** aggregiert die Informationen aus den ihm untergeordneten bzw. bei ihm angemeldeten GRIS, erzeugt den Verzeichnisbaum und erlaubt Suchanfragen auf diesem.
Genaugenommen ist der GIIS ein GRIS, welcher sich bei sich selbst registriert, was mittels bestimmter Einträge in der Konfigurationsdatei *grid-info-slapd.conf* explizit durch bestimmte Parameter festgelegt wird.
Im Unterschied dazu verfügt ein GRIS ”auf lokaler Ebene” vermittels einer gleichnamigen Datei in seinem lokalen Konfigurationsverzeichnis über das Wissen, welchem GIIS er zugeordnet ist([3], Seite 2-3).

Für den Mechanismus der Aktualisierung des LDAP-Verzeichnisbaumes wurde im Vergleich zur Version $1.1.3$ des Globus-Toolkit sogar eine äußerst intelligente Lösung gefunden.

In der hier besprochenen Version des Globus-Toolkit registrieren sich die GRIS mittels des **Grid Resource Registration Protocol (GRRP)**[7] bei den ihnen zugeordneten GIIS.
Der GIIS wiederum holt sich erst bei einer Anfrage bezüglich des von ihm verwalteten Verzeichnisbaumes die benötigten Informationen vom betreffenden GRIS.
Eine Aktualisierung der Informationen in dem Verzeichnisbaum findet erst statt, wenn eine MDS-Komponente sich anmeldet oder der Cache eines GRIS invalidiert wurde. Der Cache des GRIS enthält die oben beschriebenen Zeitstempel der ihm zugeordneten GRAM.

In der Version $1.1.3$ des Globus-Toolkit waren die MD-Server sowohl für die Sammlung der Informationen als auch für die Beantwortung von Anfragen bezüglich des von ihnen verwalteten Verzeichnisbaumes zuständig. Dabei ergaben mehrere MD-Server den globalen Dienst. Allerdings konnten die Informationen nur irregulär in einem Zeitrahmen von Stunden mittels Anfragen an die GRAM der anderen Maschinen aktualisiert werden, was im wesentlichen eine Konsequenz der seinerzeit möglichen Latenzzeiten war [8].

Zum Abschluss dieser Ausführungen über den MDS sei noch bemerkt, dass ein MDS in einer sogenannten Tier-Umgebung agieren kann, was bedeutet, dass ein GIIS einer regionalen Organisation als GRIS in einer übergeordneten Organisation auftreten kann.

3.2.5 Sicherheit

Die Sicherheitskomponente des Globus-Toolkit, der **"Generic Security Service" (GSS)**, hat ihren Fokus auf das Problem der Authentifizierung in einer komplexen Umgebung von dynamisch verbundenen Nutzern.
Damit ergeben sich nach Foster und Kesselman [9] zwei Problembereiche:

1. N-Way security context:
   im Gegensatz zu klassischen Client-Server-Architekturen, in welcher die Authentifizierung zwischen einem Klienten und einem Server stattfindet, wird in einem Grid wegen der Dynamik der Kommunikationskanäle die Fähigkeit benötigt, Prozesse dynamisch auf vielen Ressourcen zu starten und zu beenden und zwischen ihnen eine vielfältige Kommunikation im unicast- oder multicast-Modus zu ermöglichen,

2. lokale Heterogenität bzw. verschiedene administrative Domänen:
   die Ressourcen halten verschiedene Authentifizierungsmechanismen und -regeln wie Kerberos, die Secure Shell oder ein Passwort vor, so dass ein Sicherheitsmechanismus in einem Grid für einen Benutzer auf verschiedenen Ressourcen verschiedene Identitäten ermöglichen sollte

Das Globus-Toolkit der Version $2.x$ bietet dafür folgenden Lösungen:

[7] Das GRRP wird in dem Programm `grid-info-soft-register` implementiert.

- für den ”N-Way security context” bedeutet dies die einmalige Authentifizierung des Benutzer bei seinem lokalen Sicherheitsdienst und die weitere Authentifikation des Benutzers bei entfernten Ressourcen durch diesen Sicherheitsdienst ohne weitere Interaktion mit dem Benutzer:
  dieses Prinzip des **”single sign on”** wird beim Globus-Toolkit durch den `grid-proxy` realisiert

- für die ”lokale-Heterogenität” wurde ein Mechanismus zur Abbildung der durch ein ITU-X.509 Zertifikat garantierten Identität des entfernten Benutzers auf ein lokal verfügbares Benutzerkonto mittels des *grid-mapfile* und `gsi-proxy` gewählt.

Die Sicherheitsinfrastruktur des Globus-Toolkit (**”Globus Security Infrastructure” (GSI)**) basiert in den gegenwärtig verfügbaren Versionen auf der **Public Key Infrastructure (PKI)**. Diese lehnt sich an die Idee des Tickets der Kerberos-Sicherheitsmechanismen an. Zwar ist die Implementation von Kerberos geplant, aber für den Programmierer von Sicherheitsmechanismen für das Globus-Toolkit wird dies eine binäre Entscheidung zwischen Kerberos und PKI sein.

Die PKI des Globus-Toolkit baut wie erwähnt auf dem ITU-X.509-Protokoll der ”International Telecommunication Union” auf und zeichnet sich durch folgende Eigenschaften aus:

- das Zertifikat bzw. ”credential” bindet den Namen des Benutzers an seinen öffentlichen Schlüssel

- dieses ”Bündel” wird von einer dritten vertrauenswürdigen Instanz - im Idealfall nach Vorlage des Personalausweises - unterzeichnet

- ein Zertifikat besteht demzufolge aus vier Entitäten:

  1. der Name des Benutzers
  2. der Name des Unterzeichners des Zertifikates, genannt Zertifikaten-Autorität oder CA
  3. der öffentliche Schlüssel des Benutzers
  4. die Signatur der Zertifikaten-Autorität

- die drei erstgenannten Entitäten eines Zertifikates werden mittels eines HASH-Wertes[8] eindeutig repräsentiert

- der eben genannte HASH-Wert wird mit dem privaten Schlüssel der Zertifikaten-Autorität chiffriert und bildet damit die Signatur des Zertifikates

[8]”Hashing” bedeutet die eindeutige Einbildung einer Menge an Zeichenfolgen auf eine Chiffre fester Länge. Diese Chiffre ist eindeutig für jede mögliche Kombination an Zeichenfolgen. Ferner ist die Abbildung ist eine Einwegfunktion, d.h. es gibt keine Umkehrfunktion von der Chiffre auf die Ausgangsmenge.

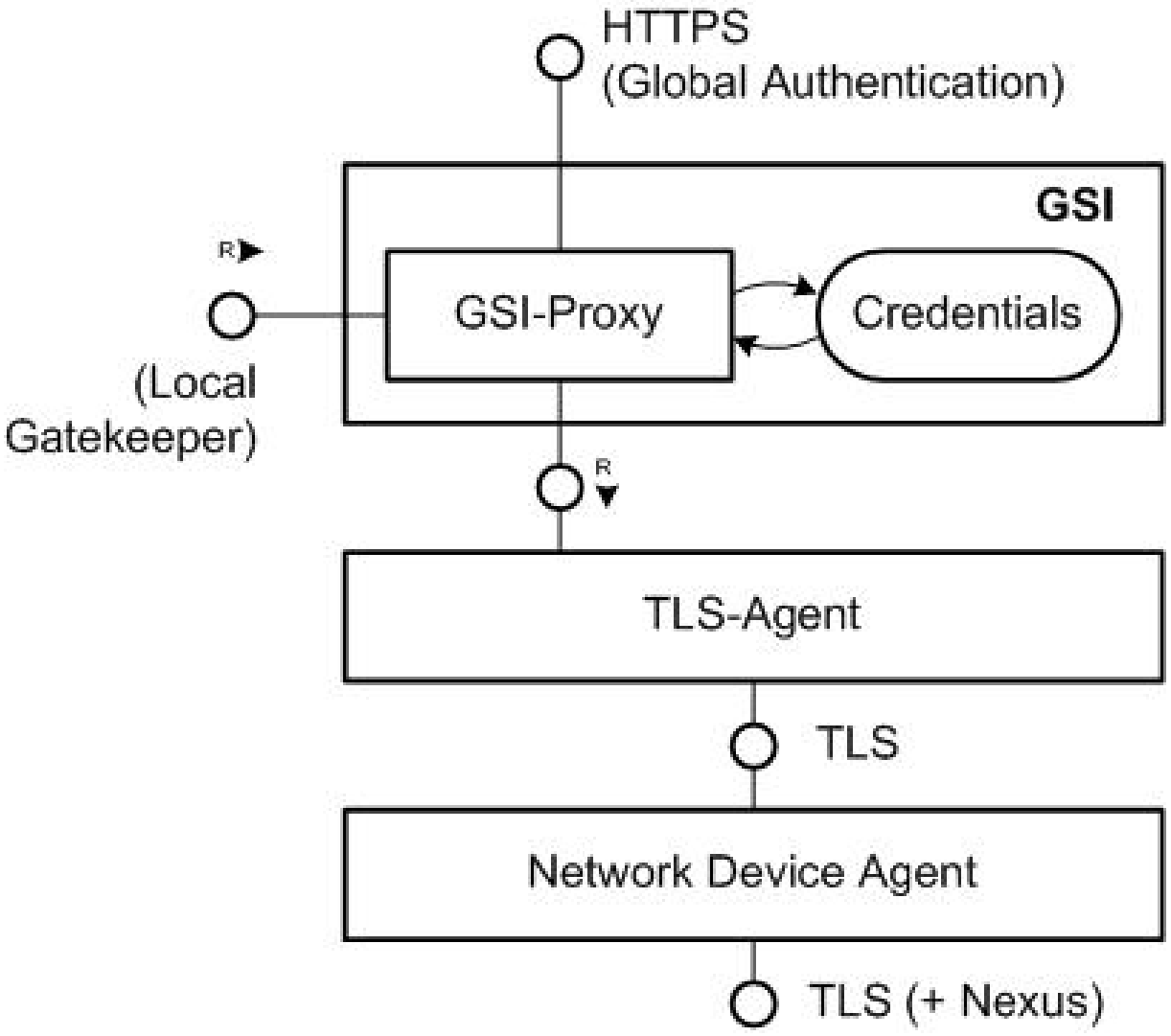


Abbildung 3: Der Aufbau der Sicherheitskomponente im Überblick als FMC-Aufbau-Modell

- beide Schlüssel können von jedem Programm, das die ITU-X.509-Sicherheitsmechanismen implementiert, auf Identität überprüft werden und im Fall des positiven Ausganges der Prüfung gilt der Inhaber des Zertifikates als eindeutig identifiziert
- das Protokoll ermöglicht aufgrund der Einbeziehung eines öffentlichen und eines separaten privaten Schlüssels die asymmetrische Verschlüsselung für die Kommunikation

Mit diesen Vorbemerkungen können nun die Komponenten des Sicherheitsdienstes des Globus-Toolkit charakterisiert werden.
Auf der

1. lokalen Ebene agieren die verschiedenen Sicherheitsdienste wie Kerberos oder

PKI, wobei die Wahl des lokalen Sicherheitsmechanismus beim Globus-Toolkit wie schon bemerkt auf die PKI beschränkt bleibt

2. globalen Ebene agiert der **GSI-Proxy** bzw. `grid-proxy` als Implementation der GSS, indem er

   - das beschriebene ”single sign-on”,
   - die Abbildung des X.509-Zertifikates des Benutzers auf ein lokales Benutzerkonto mittels des *grid-mapfile*,
   - die Ausführung verschobener Globus-Jobs zu einem späteren Zeitpunkt,
   - die Delegation der Zertifikate an andere Proxies zum Zwecke der dynamischen Allokation der Ressourcen, genannt ”delegated proxy”, und
   - die verschlüsselte Kommunikation vermittels eines OpenSSL-Clients (TLS-Benutzerprogrammes[9]) nach der Authentifizierung des Benutzers

   unterstützt.

Zusätzlich zu den Diensten existiert noch die **GSSAPI**, welche eine generische Programmierschnittstelle zur Entwicklung von Werkzeugen für die Administration und Verwaltung der Globus-weiten Sicherheitsmechanismen offeriert. Außerdem stellt die GSSAPI eine Schnittstelle für die Interaktion mit den speziellen, lokalen Sicherheitsmechanismen, allerdings kein Benutzerprogramm, zu Verfügung.

3.2.6 Fehlertoleranz-Dienst

Wie schon im Kapitel 3.2.2 im Zusammenhang mit dem Job-Manager des GRAM erwähnt, ist der Fehlertoleranz-Dienst (”Fault-Tolerance-Service”) bis zur Version 3.0 des Globus-Toolkit nicht implementiert worden.
Ebenso verhält es sich mit dem ”Executable Management”.

Die wesentliche Aufgabe des **Globus Heartbeat-Monitor (HBM)** ist die Überwachung und Verifikation von Prozessabläufen auf entfernten Ressourcen.
Es handelt sich hier nur um Prozess- und nicht um Systeminformationen, die über HTTP ausgetauscht werden.

Die Gestaltung des Fehlertoleranz-Dienstes ist nach Foster und Kesselman [9] unterteilt in einen:

- lokalen Dienst, welcher durch den **HeartBeat Client (HBC)** zur Verfügung gestellt wird;
  diese Komponente holt auf Anfrage eines Benutzer Informationen über den (die) betreffenden Prozess(e) ein

[9] TLS := Transport Layer Security, eine aktuellere Definition des bekannten SSL-Standard

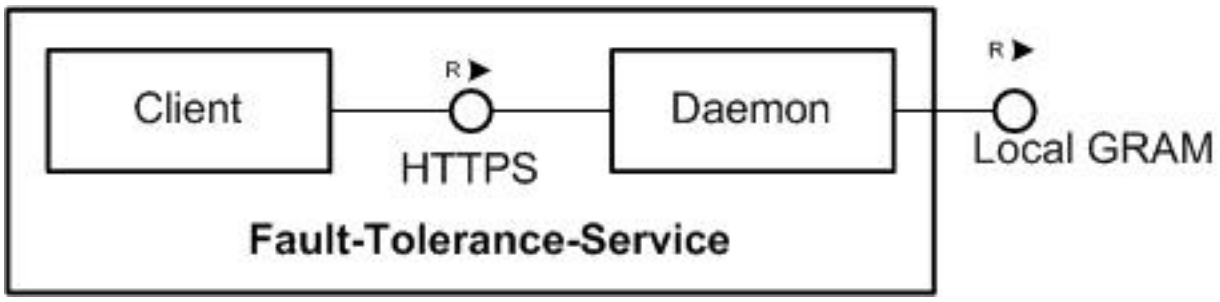


Abbildung 4: Der Aufbau des HeartBeat-Monitor im Überblick als FMC-Aufbau-Modell

- globalen Dienst, den der **HeartBeat Daemon (HBD)** bereitstellt;
  dieser agiert auf der entfernten Maschine und reagiert auf den Trigger des HBC, indem er den betreffenden Prozess in einem selbstgewählten Intervall testet, die Ergebnisse in einem alloziierten Datenbehälter sammelt und an den anfragenden Klienten zurückgibt.

3.2.7 Remote-Data-Access

Es ist im voraus zu bemerken, dass die Gestaltungsprinzipien, welche schon für den MDS in Kapitel 3.2.4 beschrieben wurden, auch während der Gestaltung der Komponenten für den Datenzugriff angewandt wurden.
Dies bedeutet eine Trennung der Komponente in einen Dienst zur Datenmanipulation und einen zur Sammlung der Informationen bzw. Metadaten über die verfügbaren Daten. Der letztgenannte Dienst wird durch die später beschriebene Komponente ”Replica” realisiert.

Die Komponente, welche den Zugriff auf den Sekundärspeicher ermöglicht, besteht wiederum aus zwei Teilkomponenten:

1. der ausschließlich lokal agierende Dienst zur Zwischenspeicherung der ausführbaren Dateien eines Globus-Job, der **GASS-Cache-Manager**

2. der Dienst zur Datenmanipulation via **GridFTP**

   - als lokaler Dienst: erweiterter ncftp
   - als globaler Dienst: erweiterter wu-ftp

GridFTP ist - wie schon in den einleitenden Bemerkungen beschrieben - das zweite wichtige Protokoll, welches im Globus-Toolkit implementiert wurde [1].
GridFTP basiert auf dem FTP-Protokoll, das in der ”IETF RFC 959”[10], formuliert wurde und durch die ”IETF RFC 2228” um Sicherheitsanforderungen erweitert wurde. Allerdings wurden diese Sicherheitsanforderungen von den Anbietern der FTP-Implementationen nicht realisiert. Deshalb haben die Entwickler des Globus-Toolkit,

[10] IETF := Internet Engineering TaskForce, RFC := Request for Comment

auf Basis der RFC 2228 und den spezifischen Erfordernissen der Grid-Technologie an den Datentransfer, GridFTP entwickelt und implementiert [11].
Die Charakteristika dieses Protokolles lassen sich wie folgt beschreiben:

- Einführung einer Schicht für authentifizierte und gesicherte Verbindungen (RFC 2228),
- Kontrolle des Transfers zwischen zwei Servern durch eine dritte Partei,
- ”striped file access”, also Partitionen bzw. Blöcke einer Datei werden über mehrere Server verschickt und von der Empfangsstelle wieder zusammengefügt, um Bandbreite zu aggregieren,
- ”partial file access”, d.h. Teile bzw. bestimmte Blöcke einer Datei werden übertragen,
- Verwaltung von Parallelismus für Hochgeschwindigkeitstransfers, indem mehrere TCP-Streams zu einem TCP-Stream vereinigt werden können,
- Datenübertragungen können nach einer Unterbrechung wieder aufgenommen werden,
- generische Regeln für die Manipulation von Dateien
    1. ”shared-read-only”: Zugriff auf eine Datei im Lesemodus
    2. ”shared-write”: Zugriff auf eine Datei im Schreibmodus, wobei der letzte Schreibzugriff dominiert
    3. ”shared append-only”: Zugriff auf eine Datei im Schreibmodus, wobei die einzufügenden Daten mit der betreffenden Datei konkateniert werden, was sich für die Verteilung von threads auf Prozessoren und der Ergebnisspeicherung durch die threads als sinnvoll erweist
    4. ”unshared unrestricted read/write”

Die Mächtigkeit von GridFTP wurde dann mittels der oben benannten FTP-Dienste auf der lokalen und globalen Ebene implementiert.

Die Funktionen des GASS-Cache-Manager sind dann folgende:

- Verwaltung der Datentransfers zu entfernten Ressourcen für Scheduling und Ressourcenallokation und Angebot einer Schnittstelle für den Replica mittels `globus_gass_copy`,
- Synchronisation der Datentransfers,
- Integration von GridFTP, HTTP und lokaler Datenein- und -ausgabe, um einen sicheren Datentransfer zu ermöglichen.

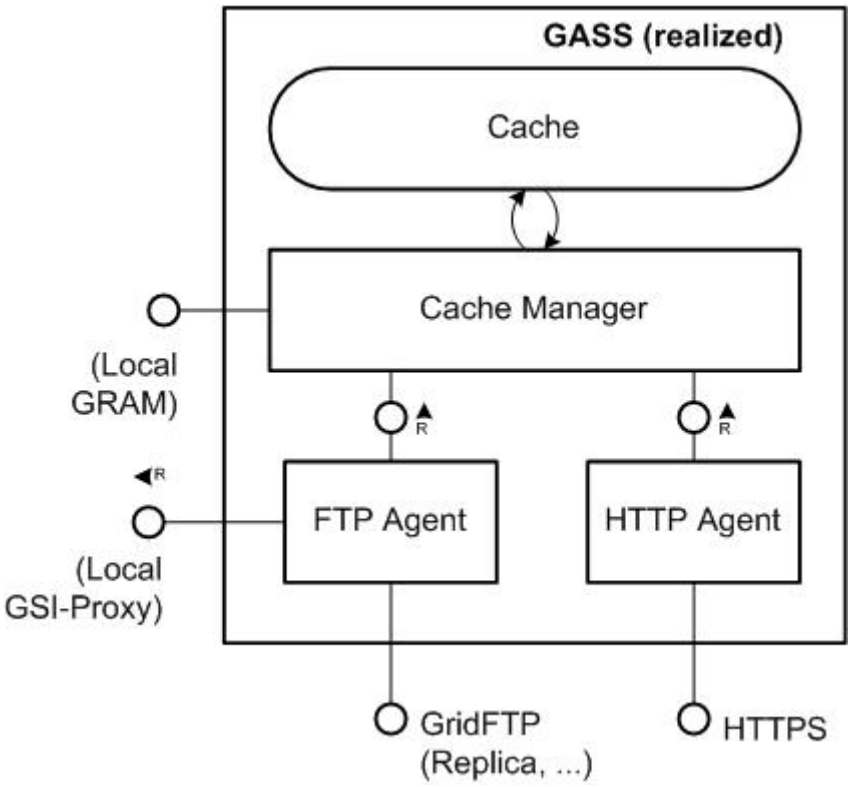


Abbildung 5: Der Aufbau der GASS-Komponente in der gegenwärtigen Realisierung im Überblick als FMC-Aufbau-Modell

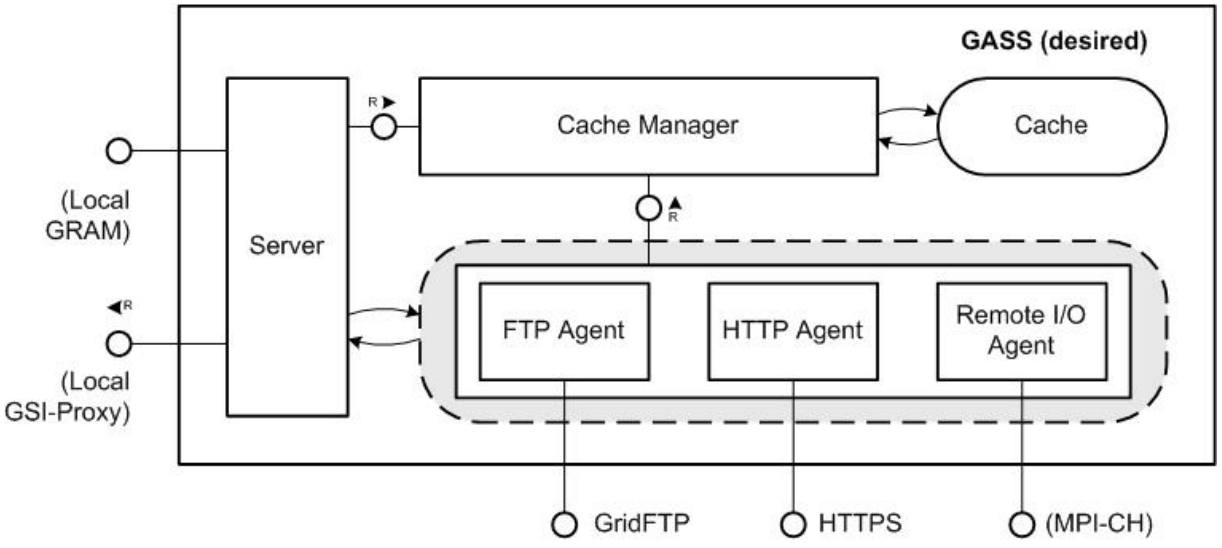


Abbildung 6: Der Aufbau der GASS-Komponente in der gewünschten Realisierung im Überblick als FMC-Aufbau-Modell

Ein Beispiel: jeder in der RSL beschriebene Job wird in den lokalen GASS-Cache auf der Maschine des Ressourcen-fordernden Benutzers geladen und mit dem entfernten GASS-Cache abgeglichen. *stdout* und *stderr* werden dann via `GASS_append-mode` gehändelt und der Cache wird nach Beendigung des Jobs geleert.

Ein weiterer wichtiger Aspekt der gegenwärtigen Realisierung des Globus-Toolkit ist - wie in Abbildung 5 veranschaulicht - der Fakt, dass die GASS-Komponente noch nicht ihrem Entwurf entspricht.
Beim Entwurf wurde noch eine Teilkomponente mit Namen **Remote I/O** aufgeführt, welche die Umleitung der Ausgabe- oder Eingabeströme der Globus-Jobs wie *stderr* oder *stdout* auf die verschiedenen, im Grid verfügbaren Ressourcen übernimmt.
Derzeit wird diese Komponente nur durch Werkzeuge der höheren Ebene realisiert, und zwar durch **MPI-CH2**.

Ferner verhält es sich so, dass die Vereinigung der drei Dienste bzw. Akteure für die jeweils verschiedenen Protokollformen des Datentransfers einen weiteren Akteur bedingen. Dieser sollte dann mittels eines strukturvarianten Speichers in der Lage sein, den gewünschten Akteur zur Ausführung eines Datentransfers dynamisch zu laden.

3.2.8 Replica

Diese Komponente ist, wie im vorherigen Kapitel beschrieben, einerseits für die Verwaltung der Metadaten über die verfügbaren Dateien zuständig [11]. Andererseits dient sie auch, wie der Name schon offenbart, der Spiegelung der in einem Grid verfügbaren Dateien, basierend auf der Annahme, das Globus-Toolkit in einer Umgebung mit loser Kopplung einzusetzen. Das impliziert damit letztlich die Notwendigkeit einer Absicherung vor Ausfällen der teilnehmenden Ressourcen im Grid.

Der Replica-Service ist deshalb nur global verfügbar, agiert also über die lokalen Ressourcen hinweg.
Die Komponenten von Replica sind damit folgende:

1. **Metadata Service**: er abstrahiert den physischen Ort der Dateien mittels einer URL, indem er einen Katalog mit Metadaten über die verfügbaren Dateien im Grid zur Verfügung stellt;
   ferner erlaubt er das Durchsuchen von Datensätzen nach einem Inhalt und ist dem UNIX-Befehl `apropos` vergleichbar

2. **Replica Management**: er findet mögliche Speicherorte für/von Datensätze/n und führt die Anfragen zur Spiegelung mit Hilfe der GridFTP-Teilkomponente aus,
   er ist damit für die Erzeugung, Ortung und Verwaltung von Repliken bzw. Kopien an Datensätzen verantwortlich und einer Kombination der UNIX-Befehle `locate` und `cp` vergleichbar

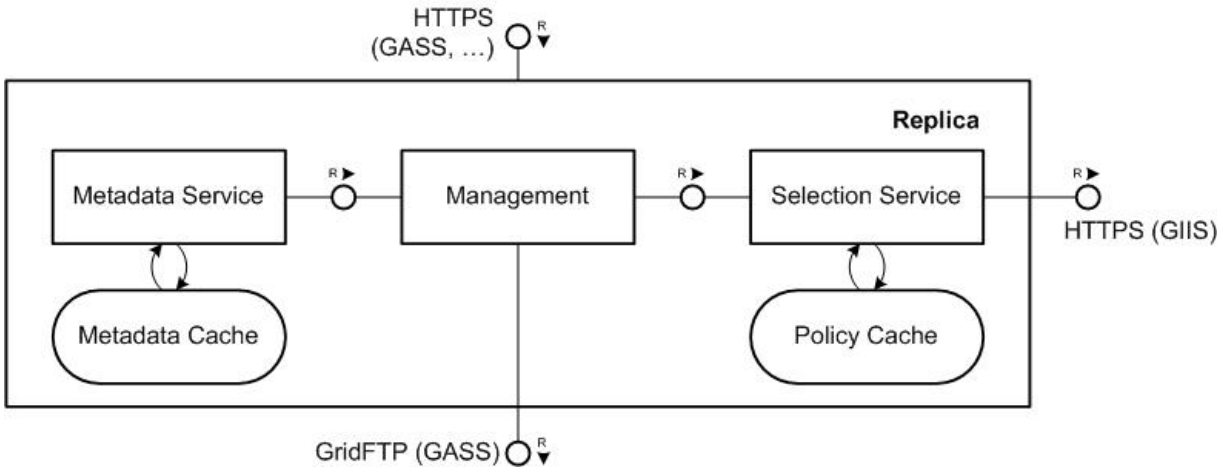


Abbildung 7: Der Aufbau des Replica-Dienstes im Überblick als FMC-Aufbau-Modell

3. **Replica Selection Service**: er implementiert Algorithmen für die Abschätzung von Datentransfers nach dem Aspekt der Zuverlässigkeit sowie Schnelligkeit und wird dabei auch von GIIS-Komponenten des Globus-Toolkits unterstützt

3.3 Ausblick

In diesem Abschnitt sollen noch einige Gedanken zur Zukunft und dem Status des Globus-Toolkit geäußert werden.
Wie schon erwähnt, wurde die Komponente des **Executable Management** genauso wie die Komponente des **HeartBeat-Monitors** in den bisherigen Versionen des Globus-Toolkit noch nicht implementiert. Diese werden aber den Ankündigungen der Entwickler zufolge in der Version 3.0 verfügbar sein, die im Juni 2003 herausgegeben wird [7].
Diese neue Version des Globus-Toolkit wird aber noch weitere Fortschritte in der Grid-Technologie hervorbringen. Dabei wären zu erwähnen:

- die Umsetzung eines einheitlichen Standards für Grid-Technologien, genannt Open Grid Services Infrastructure (OGSI), da trotz der Idee der Bündelung von heterogenen Ressourcen in einem Grid mittels einer Technologie, die Interoperabilität zwischen den verschiedenen Grid-Technologien noch nicht vollständig gewährleistet ist

- Fokus auf ”higher-level”-Umgebungen zur Implementation von Applikationen wie CORBA [12]

- mit dem Fokus auf den eben genannten Punkt ergab sich die Notwendigkeit zur Implementation einer Schnittstelle für die objektorientierte Programmierung, wobei die Wahl auf das Simple Object Access Protocol (SOAP) fiel und damit eine breite Palette zur Realisierung von ”Web-Services”, letztlich auch die kommerzielle Nutzung von Grids, ermöglicht wird

Aus diesen Ausführungen kann man schließen, dass das Globus-Toolkit eine ”Entwicklung in Arbeit” (”evolving work”) ist. Dies hat zu Folge, dass beispielsweise die Programmierschnittstellen der Sicherheitskomponente des Globus-Toolkit fast gar nicht dokumentiert ist.
Auch der Komponentenaufbau des Globus-Toolkits war nur zu ergründen, weil die Informationen sich in verschiedenen Dokumenten verbargen und sich teilweise widersprachen.
Zudem wurden häufig nur einige technische und den Entwurf rechtfertigende Aspekte erläutert.

Der Aspekt der Sicherheit fokussierte in Grids bisher nur auf Fragen der Authentifikation und Kommunikation.
Das bedeutet, dass die einzelnen Ressourcen im Globus-Toolkit zwar mittels asymmetrischer Verschlüsselung kommunizieren und sich mittels strenger Zertifizierungsvorschriften identifizieren können. Allerdings ergibt sich durch die transitive Authentifizierung und die Abbildung von zertifizierten, entfernten Benutzern auf ein Benutzerkonto der lokalen Ressource ein mögliches Sicherheitsloch.
Mit der transitiven Authentifikation ist gemeint, dass ein Benutzer A bei einer Ressource B zertifiziert und anerkannt ist. Wenn allerdings die Ressource B bei einer weiteren Ressource C zertifiziert und anerkannt ist, dann ist dies auch der Benutzer A. Dieser Mechanimus wurde eingerichtet, um das Prinzip des ”single-sign on” zu unterstützen, so dass ein ressourcen-fordernder Benutzer nicht genötigt ist, sich bei ihm unbekannten, aber zertifizierten Ressourcen gesondert auszuweisen.
Die Problematik tritt aber im Zusammenhang mit der Abbildung entfernter und zertifizierter Benutzer auf ein lokales Benutzerkonto einer Ressource auf. Denn die Rechte des lokalen Benutzerkontos auf bzw. an der lokalen Ressource entsprechen dann den Rechten des entfernten Benutzers.
Diese Problematik betrifft allerdings die Sicherheitsmechanismen der lokalen Ressource bspw. des Betriebssystemes und kann mit den gegenwärtig verfügbaren Sicherheitsinstrumenten nicht hinreichend gelöst werden.

Beispielsweise stand die Grid-Technologie des, zu wissenschaftlichen Zwecken und auf den Altruismis privater Rechnerbesitzer angewiesene, ”Distributed.Net” in den Jahren 2001/2002 in der Kritik, weil die Rechner der privaten Nutzer mit Paketen zur Analyse von Anthrax-Sporen beschickt wurden.
An dieser Stelle lässt sich die mögliche Dimension der Verletzung von Privatrechten von Benutzern erahnen, bspw. eine Ausspähung von Paketen, welche gerade in einem Grid berechnet oder gespeichert werden. Da Grids überwiegend für rechen- und datenintensive Anwendungen, wie dem BoilerMaker-Projekt zum Entwurf von Systemen für die Emmissionskontrolle industrieller Verbrennungsanlagen, eingesetzt werden, ergibt sich nicht nur ein Interesse an Daten, sondern es bietet sich hier ein erster Ansatzpunkt.

Doch bisher wurden Grid-Technologien überwiegend im akademischen Umfeld

eingesetzt, was auch die Menge der Benutzer, denen zu vertrauen war, eindeutig einschränkte.
Doch wenn Grid-Technologien sich ebenso wie das ARPA-Net aus dem wissenschaftlichen Umfeld heraus zu einer Technologie von breitem öffentlichen und damit auch kommerziellem Interesse entwickeln, werden die Fragen der Sicherheit wie beim Internet sehr viel immanenter werden. An dieser Stelle sei der interessierte Leser aber auf das erste grundlegende, aber nicht minder unterhaltsame Werk zu Fragen der Sicherheit und Regierbarkeit des Internets durch staatliche Organisationen von Lawrence Lessig, seines Zeichens Professor für Verfassungsrecht an der Universität Stanford und Hilfsrichter im AntiTrust-Verfahren gegen Microsoft im Jahre 1998, verwiesen [5].

Diese Betrachtungen werden komplettiert durch eine Bewertung der Lizenz.
Zwar bemühten sich die Entwickler des Globus-Toolkit, vornehmlich die libertäre Universität Chicago und die OpenSource-erfahrenen Kalifornier, um einen Kontrakt, welcher der Öffentlichkeit im Sinne des ”public domain” zugute kommt. Allerdings entstand das Globus-Toolkit im Rahmen eines wissenschaftlichen Projektes und damit auch unter der Mitwirkung öffentlicher Gelder. Damit unterliegt dieses Projekt auch dem Eigentumsvorbehalt der Regierung der USA.
Hinzu kommen noch Beschränkungen durch US-Export-Gesetze, welche sich vorwiegend auf die Nutzung von SSL-Technologien durch nicht in der USA residente Benutzer beziehen. Zwar sind diese Gesetze gelockert wurden, aber sie unterliegen bis auf weiteres den politischen Rahmenbedingungen.
Es sei in diesem Zusammenhang erwähnt, dass das Globus-Toolkit auch intensiv für das militärische Projekt ”Distributed Interactive Simulation” (DIS) als Planungs- und Trainingstechnik des US-Militärs genutzt wird. Inwieweit Schnittstellen zwischen Grid-Technologie und dem Projekt ”Falcon View” zur Planung von Militäreinsätzen vorliegen, läßt sich nur erahnen.
In diesem Kontext wird aber die Folge der US-Export-Kontrolle deutlich. Das Globus-Toolkit dürfte der Lizenz zufolge nicht von Wissenschaftlern in Ländern eingesetzt werden, die ein schwieriges aussenpolitisches Verhältnis zur USA haben. Dies kommt ansatzweise einer Einschränkung der wissenschaftlichen Freiheit bzw. eines Verbotes der wissenschaftlichen Weiterentwicklung nahe.
Denn das Globus-Toolkit kann nicht nur für die Approximation von Militärstrategien sondern auch für die Datenverarbeitung von metereologischen Satellitenaufnahmen verwendet werden.
Und klimatische Erscheinungen treten auch außerhalb der USA auf.

4 Installation des Globus Toolkits

Das Globus Toolkit benutzt zur Installation die Grid Packaging Technology (GPT). Das GPT Softwarepaket erlaubt die einfache Installation von speziell erstellten GPT-Paketen, den sogenannten ”bundles”. GPT installiert wahlweise entweder vorkompilierte Pakete, die ”binary bundles”, oder noch zu übersetzende Pakete, die ”source bundles”. Im Folgenden werden wir uns auf die Installation von einer binären Version des Globus Toolkits konzentrieren, da diese bereits für Linux, Solaris und weitere UNIX-Derivate verfügbar sind.

Unabhängig von der Wahl des ”bundles” (binary oder source) findet man in der Regel zu jeder Toolkit-Komponente eine Client- und eine Serverversion. Befindet sich auf einem Rechner lediglich die Clientversion, so kann dieser Rechner keine Ressourcen für das Grid zur Verfügung stellen, sondern lediglich ”jobs” im Grid ausführen. Analog verhindert eine reine Serverversion die Nutzung des Grids für eigene ”jobs”, erlaubt aber das Teilen von Ressourcen mit dem Grid. Für Entwickler gibt es zu jeder Komponente noch ein SDK, um eigene Programme gegen die benötigten Headers und Libraries kompilieren und linken zu können.

4.1 Systemanforderungen

Die Hardwareanforderungen der Globus-Software selbst sind eher moderat.
Jeder handelsübliche PC reicht bereits aus, um eine Globus-Distribution zu installieren und zu betreiben. Entscheidend für die Wahl einer geeigneten Hardware sind eher die ”remote jobs”, d.h. Programme die durch Globus auf der eigenen Maschine ausgeführt werden sollen.
In Bezug auf die benötigte Software zur Installation einer Globus-Distribution sind folgende Programme zu nennen. Um GPT nutzen zu können wird mindestens `Perl 5.005` benötigt [11]. Ferner werden die GNU-Programme `tar` und `make` benötigt und bei einer Installation von einer ”source distribution” natürlich auch ein aktueller C-Compiler, beispielsweise `gcc`.

4.2 Installationsablauf

Eine typische Installation folgt den folgenden Schritten:

1. Erstellen eines neuen Benutzers ”globus”
2. Erstellen der Installationsverzeichnisse für GPT und Globus
3. Einrichten der Toolkitumgebungsvariablen
4. Installation des Grid Packaging Toolkits (GPT)
5. Installation der Globus Distribution

[11] Auf www.perl.com zum Herunterladen verfügbar.

6. Verifizieren der Installation
7. Installation der Simple Certificate Authority (SimpleCA)
8. Beantragen eines Benutzerzertifikates
9. Testen der Installation
10. Beantragen eines Host-Zertifikates
11. Starten der Dämonen
12. Test der Dämonen

Für eine reine Clientinstallation sind lediglich die Schritte 1-9 erforderlich. Da unserer Meinung nach eine Installation anhand eines Beispiels am leichtesten nachzuvollziehen ist, beschreiben wir im Folgenden die Installation eines 2 Maschinen Grids auf Basis des Globus Toolkits. Auf jeder dieser Maschinen soll Linux laufen. Für die Installation verwenden wir die ”binary distribution” mit allen Komponenten [12]. Maschine A soll in diesem Grid sowohl Client als auch Server sein und Maschine B vorerst nur Client. Ziel ist das Starten eines ”jobs” von B auf A.

4.3 Installationsschritte zum Einrichten eines Servers

Wir starten mit unserer Installation auf Maschine A.

4.3.1 Erstellen eines neuen Benutzers ”globus”

Das Erstellen eines ”globus” Accounts ist optional. Man kann das Toolkit auch ohne Probleme als root oder normaler Benutzer installieren. Jedoch impliziert letztere Variante, daß ausschließlich die gewählte Person Konfigurationen am Grid vornehmen kann. Dennoch sind für bestimmte Installationvorgänge root-Rechte notwendig. Immer wenn dies der Fall ist, so wird gesondert darauf hingewiesen durch ”als root”.

4.3.2 Erstellen der Installationsverzeichnisse für GPT und Globus

In unserem Beispiel führen wir dafür die folgenden Befehle aus:

```
mkdir /usr/globus
mkdir /usr/globus/gpt
chmod 755 /usr/globus /usr/globus/gpt
```

[12] Eine aktuelle Binär-Distribution findet man unter `www.globus.org`.

4.3.3 Einrichten der Toolkitumgebungsvariablen

Als nächstes müssen wir eine Reihe von Umgebungsvariablen setzen. Für den Fall das alle Benutzer der Maschine das Globus Toolkit nutzen, bietet sich das Eintragen der folgenden Zeilen in die Datei */etc/profile* als Benutzer 'root' an:

```
GPT_LOCATION=/usr/globus/gpt/
GLOBUS_LOCATION=/usr/globus/
GLOBUS_HOSTNAME=hpi.uni-potsdam.de
export GPT_LOCATION GLOBUS_LOCATION GLOBUS_HOSTNAME
```

Will man das Toolkit nur für bestimmte Benutzer installieren, so werden deren Konfigurationsdateien geändert, also entweder die Datei *.profile, .bashrc* oder *.bash_profile*. Um die neuen Einstellungen zu laden, muss man sich erst ab- und dann wieder anmelden oder den Befehl `source` *geänderte Konfigurationsdatei* ausführen.

4.3.4 Installation des Grid Packaging Toolkits (GPT)

Nach dem Herunterladen des GPT-Pakets von der Globus-Webseite [13] kann die Installation des GPTs erfolgen:

`tar xzvf gpt-2.2.5-src.tar.gz` (zum Entpacken)
`cd gpt-2.2.5`
`./build_gpt` (zum Installieren)

4.3.5 Installation der Globus Distribution

Zur Installation der Globus Distribution benutzen wir das gerade installierte GPT. Wie anfangs erwähnt wählen wir zum Download das *all-bundle*, also das Paket mit allen Komponenten des Globus Toolkits. Dieses Paket bzw. "bundle" legen wir in ein temporäres Verzeichnis, zum Beispiel */tmp/gptbuild*. Die Installation erfolgt dann mittels

```
/usr/globus/gpt/sbin/gpt-install
/tmp/gptbuild/globus-all-2.2.4-i686-pc-linux-gnu-bin.tar.gz
```

Nach der Installation müssen wieder Umgebungsvariablen gesetzt werden. Wahlweise können die folgenden Zeilen wieder entweder als 'root' in die `/etc/profile` oder in die oben genannten Konfigurationsdateien der einzelnen Grid-Benutzer eingetragen werden:

```
if [ -e $GLOBUS_LOCATION/etc/globus-user-env.sh ]; then
    . $GLOBUS_LOCATION/etc/globus-user-env.sh
fi
```

Nun führt man noch den folgenden Befehl aus, um die Installation abzuschließen.

```
$GPT_LOCATION/sbin/gpt-postinstall
```

Dieses Programm weist während der Ausführung darauf hin, dass noch `setup-gsi`

[13] www.globus.org

auszuführen ist.
Dieses Programm werden wir im übernächsten Schritt auch ausführen, zuvor generieren wir aber noch die *"Header files"* für die Nutzung der Entwicklerbibliotheken (SDK). Dabei ist `gcc32dbg` als Parameter für den "flavour" die geeignete Wahl für Linux:

```
$GPT_LOCATION/sbin/gpt-build gcc32dbg -nosrc
```

4.3.6 Verifikation der Installation

Um zu überprüfen, ob alle Paketabhängigkeiten bei der Installation berücksichtigt wurden, dient das folgende Kommando:

```
$GPT_LOCATION/sbin/gpt-verify
```

Die Ausgabe des Programms sollte bestätigen, daß alle Pakete kohärent sind.

4.3.7 Installation der Simple Certificate Authority (SimpleCA)

Jeder Anwender der Globus benutzen möchte, benötigt ein *user certificate*.
Dieses Zertifikat kann man entweder direkt von der Zertifikatsautorität von Globus (Globus Certificate Authority) in den USA[14] ausstellen lassen oder aber mit einer eigenen Zertifizierungsstelle generieren.
Da wir in unserem Beispiel ein geschlossenes Grid aufbauen wollen, bietet sich eine eigene Zertifizierungsstelle (CA) an. Zu diesem Zweck bietet Globus auch das geeignete Paket, das *SimpleCA-bundle*, welches eine einfache lokale Zertifikatsautorität implementiert.
Da die CA das Herzstück des Sicherheitskonzeptes von Globus ist, sollte eventuell die Erstellung eines separaten Benutzerkontos in Betracht gezogen werden. Damit könnten dann Zertifikate nur von diesem Account signiert werden. Unter diesem Benutzer muss dann die Installation des Globus SimpleCA-Pakets mit folgenden Befehlen erfolgen:

```
$GPT_LOCATION/sbin/gpt-build
/tmp/GPTbuild/globus_simple_ca_bundle-latest.tar.gz gcc32dbg
```

Ausgaben der folgenden Art kann man dabei ignorieren:

```
make: *** No rule to make target 'distclean'. Stop.
```

Während der Ausführung des letzten Befehls:

```
$GPT_LOCATION /sbin/gpt-postinstall
```

wird ein eindeutiger "subject name" erfragt. Dieser "subject name" ist eine Art CA spezifischer Namensraum. Eine mögliche Antwort wäre zum Beispiel:

```
CN=HPI CA, OU=Globus, O=HPI
```

Dabei gibt `O=HPI` den höchstgelegenen Teil des Namensraumes an, zum Beispiel eine Schule, Firma oder Universität.
`OU=Globus` verengt den Namensraum auf eine irgendwie geartete Untergruppe, hier

[14] via e-mail an ca@globus.org

alle Globus Benutzer.
Hinter `CN` gehört der Name der Zertifizierungsstelle.
Da Namensräume eindeutig sein sollen, wird von den Entwicklern des Globus-Toolkit darauf hingewiesen, dass `/O=Grid/O=Globus` nicht verwendet werden darf, da dieser Namensraum für die Globus CA schon reserviert ist[15].
Als nächstes gibt man noch die Existenzdauer der hier individuell eingerichteten Zertifierungsstelle und damit die Gültigkeitsdauer der von ihr herausgegebenen Zertifikate an.
Abschließend wird das Skript `setup-gsi` ausgeführt, um die Globus Security Infrastructure (GSI) zu konfigurieren:

```
cd $GLOBUS_LOCATION/setup/globus_simple_ca_<HASH>_setup/[16]
./setup-gsi -default
```

Die Einstellung `default` gibt an, daß die gerade installierte ”SimpleCA” die standardmäßige CA ist.

4.3.8 Beantragen eines Benutzerzertifikates

Diesen Schritt sollte man nicht als ’root’ ausführen, da wir jetzt ein erstes Zertifikat ausstellen, um einem Benutzer die Teilnahme am Grid zu ermöglichen. Folgender Befehl muss dafür ausgeführt werden:

```
grid-cert-request -cn ''Muster Mustermann''
```

Nun wird man wieder nach einem eindeutigen ”subject name” gefragt. Angebracht wäre in unserem Fall zum Beispiel der Folgende:

```
/O=HPI/OU=Globus/OU=hpi.uni-potsdam.de/CN=Muster Mustermann
```

Auch hier ist wieder:

`/O=HPI` der höchstgelegene Namensraum

`/OU=Globus` eine Gruppe in der oberen Gruppe

`/OU=hpi.uni-potsdam.de` der vollständige Domänenname[17]

`/CN` ist der Name des zukünftigen Inhabers von diesem Zertifikat

Obiger Befehlsaufruf generiert eine Datei namens *usercert_request.pem* und einen privaten Schlüssel in der Datei *userkey.pem*. Beide Dateien findet man in `$HOME/.globus`. Ferner wird eine leere Datei `usercert.pem` generiert, die später durch das - von der Zertifikaten-Autorität signierte - Zertifikat ersetzt werden muss. Deshalb sendet man die Datei `usercert_request.pem` an die zuständige Zertifikaten-Autorität. Da unsere ”SimpleCA” auf dem selben Rechner läuft reicht der folgende Befehlsaufruf[18]:

```
grid-ca-sign -in /user_home/.globus/usercert_request.pem
-out /user_home/.globus/usercert.pem
```

[15] Mehr dazu im ”Admin Guide” unter www.globus.org, Unterpunkt ”Documentation”.
[16] HASH ist dabei eine rechnerspezifische Zufallssequenz.
[17] FQDN - Fully Qualified Domain Name
[18] Dieser Befehl muss natürlich von dem ausgeführt werden, der die ”SimpleCA” installiert hat.

4.3.9 Testen der Installation

Zum Testen der Installation wird über
`grid-proxy-init`
ein eigener proxy gestartet. Nun wird man aufgefordert das Passwort, welches zum Erstellen des Zertifikates benutzt wurde, einzugeben. Dieses wird gegen das Zertifikat verifiziert. Bei erfolgreicher Anmeldung hat man sich Zugang zum Grid verschafft. Der `grid-proxy` ermöglicht den ”single-sign-on” für das Grid, da für jede weitere nötige Authentifizierung im Grid der proxy uns identifiziert.
Da bisher noch keine Dämonen laufen, müssen wir unseren eigenen `gatekeeper` starten. Wir verschaffen uns damit im Prinzip nur wieder Zugang zu unserem eigenen Rechner, jetzt aber über die GRAM-Komponente des Toolkits:

```
globus-personal-gatekeeper start
```

Die Ausgabe dieses Kommandos ist:

```
GRAM contact: machine:port:/O=HPI/OU=Globus/CN=Muster Mustermann
```

Dabei ist:

- `Machine` der Name für die lokale Maschine.
- `Port` ist die gatekeeper ID.
- `Muster Mustermann` ist der ”Common Name” (CN) des Benutzers

Durch folgenden Befehl können wir dann, zugegebenermaßen etwas kompliziert, unseren Rechner dazu bewegen das Datum auszugeben, wobei `machine:port:''SUBJECTSTRING''` durch obigen `GRAM contact` ersetzt werden muss[19]:

```
globusrun -o -r machine:port:''SUBJECTSTRING''
'&(executable=/bin/date)'
```

Falls der Befehl nicht mit der Ausgabe des Datums endet, so hilft die Datei *globus-gatekeeper.log* dabei, mögliche Fehlerquellen zu identifizieren. Bei erfolgreicher Ausführung kann man den `gatekeeper` über

```
globus-personal-gatekeeper -killall
```

wieder beenden und den proxy mittels

```
grid-proxy-destroy
```

4.3.10 Beantragen eines Host-Zertifikates

Die nun folgenden Anweisungen sind nur auf einem Server erforderlich.
Als erstes müssen wir einen echten `gatekeeper` aufsetzen, der dann auch einen ”remote login” ermöglicht.
Dazu benötigen wir ein Zertifikat, das es uns erlaubt, Ressourcen unseres lokalen Rechners dem Grid zur Verfügung zu stellen. Das benötigte Zertifikat nennt sich *host certificate* und muss als ’root’ beantragt werden:

[19] Anführungszeichen müssen gesetzt werden, sofern Leerzeichen vorkommen.

```
grid-cert-request -service host -host FQDN
```
[20].
Über unsere SimpleCA können wir das Zertifikat wieder unterzeichnen:
```
grid-ca-sign -in /etc/grid-security/hostcert_request.pem
-out /etc/grid-security/hostcert.pem
```
Das signierte Zertifikat muss in */etc/grid-security* abgelegt werden.

4.3.11 Starten der Dämonen

Um den `gatekeeper` bei jedem Bootvorgang zu starten, müssen folgende Systemdateien als 'root' modifiziert werden. In */etc/services* muss ein neuer Dienst registriert werden:
```
gsigatekeeper      2119/tcp        # Globus Gatekeeper
```
Dieser Eintrag dient dem "Internet Superserver" (`inetd`)[21] der Zuordnung des Ports 2119 zu dem dafür zuständigen Programm, nämlich dem `gsigatekeeper`. Da die Portnummer des `gatekeeper` größer als 1024 ist, kann der `gatekeeper` prinzipiell auch als Programm mit einfachen Benutzerrechten ausgeführt werden.
Da wir den `inetd` als Akteur für die Vermittlung eines Requests aus dem Internet auf der Portnummer 2119 und dem zuständigen `gsigatekeeper` wählten, müssen wir den `inetd` neu konfigurieren. Dazu tragen wir in */etc/inetd.conf* folgende Werte ein[22]:
```
gsigatekeeper stream tcp nowait root
/usr/bin/env env LD_LIBRARY_PATH = GLOBUS_LOCATION/lib
GLOBUS_LOCATION/sbin/globus-gatekeeper
-conf GLOBUS_LOCATION/etc/globus-gatekeeper.conf
```

Um diese Veränderungen wirksam werden zu lassen, ist ein Neustart des `inetd`-Dienstes erforderlich:
```
killall -HUP inetd
```

Nun muss noch festgelegt werden, welche Rechte ein entfernter Grid-Benutzer auf unserer lokalen Maschine hat.
Diese Aufgabe wird mittels des */etc/grid-security/grid-mapfile* vorgenommen, welches einem authentifizierten und durch seinen "subject name" identifizierten, entfernten Grid-Benutzer ein lokal bekanntes Benutzerkonto zuordnet. Die Rechte des entfernten Grid-Benutzers ergeben sich damit aus den Rechten des lokalen Benutzerkontos.
Die Einträge im */etc/grid-security/grid-mapfile* haben fogendes Format:
```
''/O=HPI/OU=Globus/CN=Muster Mustermann''   <Benutzerkonto>
```
Zur sicheren Ressourcenverwaltung wird ein in seinen Rechten stark beschränktes Gastkonto, wie zum Beispiel der Benutzer "nobody" empfohlen.
Nun kann man direkt mit dem Test der Dämonen fortfahren oder aber GridFTP und MDS konfigurieren.
Um GridFTP nutzen zu können wird ein Zertifikat für die Nutzung des "Leightweight

[20] FQDN ist der Fully Qualified Domain Name, also zum Beispiel hpi.uni-potsdam.de
[21] `Xinetd` ist prinzipiell auch möglich. Für mehr Informationen bezüglich `xinetd` sei an dieser Stelle auf www.globus.org verwiesen.
[22] GLOBUS_LOCATION muss wieder ersetzt werden.

Data Access Protocols" (LDAP) benötigt. Analog zum *host certificate* müssen wir also ein LDAP-Zertifikat beantragen:

```
grid-cert-request -service ldap -host FQDN
```

Dieses Zertifikat muss dann signiert werden:

```
grid-ca-sign -in
/etc/grid-security/ldap/ldapcert_request.pem
-out /etc/grid-security/ldap/ldapcert.pem
```

Um den zugehörigen Dämonen zu konfigurieren, tragen wir in */etc/services* als Benutzer 'root' folgenden Dienst ein:

```
gsiftp  2811/tcp
```

Ferner benötigen wir in `/etc/inetd.conf` noch folgenden Eintrag[23]:

```
gsiftp  stream  tcp     nowait  root
/usr/bin/env env LD_LIBRARY_PATH = GLOBUS_LOCATION/lib
GLOBUS_LOCATION/sbin/in.ftpd -l -a -G GLOBUS_LOCATION
```

Abermals ist ein Neustart des `inetd`-Dienstes erforderlich:

```
killall -HUP inetd
```

Schließlich müssen wir uns noch dem MDS-Dienst widmen. Um diesen zu starten, genügt der folgende Befehl.

```
$GLOBUS_LOCATION/sbin/globus-mds start
```

Für einen automatischen Start des MDS-Dienstes muss dieser Aufruf in ein entsprechendes *Startup-Skript* eingefügt werden.
Nun muss noch überprüft werden, ob ein "remote host" überhaupt Zugang zum Rechner hat. Um dies zu gewährleisten, muss `/etc/hosts.allow` angepasst werden. Die generelle Syntax dieser Datei lautet:

```
SERVICE: HOST [,HOST]
```

Ein beispielhafter Eintrag wäre der Folgende:

`gsigatekeeper: 127.0.0.1, *.mydomain.com`
`gsigatekeeper: ALL` (nur in lokalen Netzen zu empfehlen)

4.3.12 Test der Dämonen

Bevor wir eine der Komponenten des Globus-Toolkits testen können, müssen wir wieder einen proxy starten:

```
grid-proxy-init
```

Nun schicken wir dem gatekeeper einen ersten "request" über den Globus Befehl `globus-job-run`[24]:

[23] GLOBUS_LOCATION muss wieder ersetzt werden.
[24] domain_name muss entsprechend angepaßt werden.

```
globus-job-run domain_name:2119 /bin/date
```

Wird das Datum ausgegeben, so haben wir einen funktionierenden GRAM-Dienst in unserem Grid.

4.4 Installation eines weiteren Clients

Die Installation einens weiteren Clients auf einer anderen Linux Maschine, in unserem Beispiel Maschine B, erfolgt bis zum Installationsschritt 6 analog zur Serverinstallation. Schritt 7 wird natürlich ausgelassen, da in unserem Grid nur eine ”SimpleCA” existieren soll.
Das erste Zertifikat eines Benutzers auf Maschine B wird von der ”SimpleCA” des Servers (Maschine A) signiert. Dieses signierte Zertifikat wird auf Maschine B in das Verzeichnis `/home/`*`username`*`/.globus` abgelegt. Da Maschine B unsere ”SimpleCA” noch nicht kennt, wird diese der CA auf Maschine A natürlich auch nicht vertrauen. Darum muss man als 'root' die folgenden Dateien aus */etc/grid-security/certificates* von Maschine A nach `/etc/grid-security/certificates` auf Maschine B kopieren:

```
3d67cfb4.0
3d67cfb4.signing_policy
```

Diese beiden Dateien identifizieren eine spezifische CA eindeutig.
Nun ist man in der Lage einen proxy über `grid-proxy-init` auf B zu starten.
Über `globus-job-run domainname:2119 /bin/date` können wir dann einen ersten ”job” auf A starten, sofern `domainname` der Domänenname von A ist, also zum Beispiel `hpi.uni-potsdam.de`.

4.4.1 Installation eines zweiten Servers

Um auch Maschine B als Server verwenden zu können, braucht man wiederum ein `host-` und ein `ldap-certificate`. Diese werden dann von der ”SimpleCA” auf A signiert.
Nun erfolgt noch eine zu A analoge Konfiguration der Startdateien und Dämonen. Nicht vergessen sollte man das Erstellen eines neuen `grid-mapfiles` für B, das auf die dort vorhandenen Benutzerkonten angepasst wird.

Literatur

[1] William E. Allcock. GridFTP Update January 2002. Technical report, http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf, 2002.

[2] Uwe Harms. The Grid - verteiltes Rechnen im Internet. *iX*, (5):162, 2001.

[3] Leon Kuntz. MDS 2.2: Creating a Hierarchical GIIS. Technical report, http://www.globus.org/mds/hierarchical_GIIS.pdf, 2002.

[4] Leon Kuntz. MDS 2.2 Users Guide. Technical report, http://www.globus.org/mds/mdsusersguide.pdf, 2003.

[5] Lawrence Lessig. *Code und andere Gesetze des Cyberspace*. Berlin Verlag, 2001. Eine erste grundlegende Antwort eines Verfassungsrechtlers auf die Frage, ob das Internet regierbar ist?

[6] The Globus Project. Globus Toolkit 1.1.3 System Administration Guide. Technical report, http://www.globus.org/toolkit/documentation, 2000.

[7] Tony Gargya und Boas Betzler. Unter Strom - Grid Computing im Enterprise-Einsatz. *Linux Enterprise*, (3):67–71, 2003.

[8] Ian Foster und Carl Kesselman. The Globus Project: A Status Report. Technical report, http://www.globus.org, 1998.

[9] Ian Foster und Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., 1999.

[10] Ian Foster und Carl Kesselman und Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Technical report, http://www.globus.org/research/papers/anatomy.pdf, 2001.

[11] Ian Foster und Carl Kesselman und Steven Tuecke. Data [m]anagement and Transfer in High-Performance Computational Grid Environments. Technical report, http://www.globus.org/gt2/datagrid.html, 2002.

[12] Ian Foster und Carl Kesselman und Steven Tuecke und Jeffrey M. Nick. The Physiology of the Grid - An Open Grid Services Architecture for Distributed Systems Integration. *http://www.globus.org/research/papers/ogsa.pdf*, pages 8–9, 23–25, 2002.

[13] Martin Müller und Linus Torvalds und andere. *Open Source kurz und gut*. O'Reilly-Verlag, 1999.

Open Grid Services Architecture

Die kommende Generation des *Grid Computing*

Lars Trieloff
lars@trieloff.net

Die Open Grid Services Architecture ist ein Versuch, einen kanonischen Ansatz zum Grid Computing zu entwickeln. In diesem Artikel werde ich auf die wichtigsten Bestandteile der Open Grid Services Architecture eingehen, die daraus abgeleitete Spezifikation der Open Grid Services Infrastructure vorstellen und Programmbeispiele zweier Implementierungen dieser geben.

Inhaltsverzeichnis

1. Einführung 3
1.1. Begriffsübersicht 3
1.2. Globus Toolkit 3 4
1.3. Wozu braucht man OGSA? 4

2. Basistechnologien 5
2.1. Globus Toolkit 5
2.2. Web Services 7
2.2.1. Schnittstellenbeschreibung 8
2.2.2. XML Protokolle 11
2.2.3. Auffindung von Diensten 12
2.2.4. Sicherheit von Web Services 13

3. Use-Cases für OGSA 13
3.1. Kommerzielles Rechenzentrum 13
3.2. Sturmvorhersage 17
3.2.1. Genutzte Ressourcen 18
3.2.2. Weitere Betrachtungen 19

4. Aufbau nach Spezifikation 19
4.1. Anforderungen an eine Grid-Umgebung 19
4.2. Schnittstellen 22
4.2.1. Grid Service 22
4.2.2. Factory 23
4.2.3. Registry 23
4.2.4. HandleMap 23
4.2.5. NotificationSource 23
4.2.6. NotificationSink 24
4.3. Sicherheitsaspekte 24
4.4. Hosting Environments 24
4.5. Ausführungsbeispiel 25

5. Open Grid Services Infrastructure 26
5.1. WSDL Erweiterungen 26
5.2. Service Data 27
5.2.1. Die WSDL-Erweiterung für Service Data 27

6. Implementationen 27
6.1. Globus Toolkit 3 28
6.1.1. Installation 28
6.1.2. Nachinstallation 29
6.1.3. Beispielprogramm 30
6.1.4. Weitere Komponenten 33
6.2. OGSI.NET 34

6.2.1. Installation 34
6.2.2. Kompilieren der Dienste 36
6.2.3. Beispielprogramm 36

7. Ausblick und Schlussbetrachtung 38
Glossar 39
Bibliographie 40

1 Einführung

Im Juni 2002 stellten Ian Foster, Carl Kesselman, Jeffrey Nick und Steve Tuecke ihr Whitepaper *The Physiology of the Grid*[physiology2002], in dem Sie Ihre Vision des Grid-Computing vorstellen. Etwa ein Jahr später liegt mit dem Globus Toolkit Version 3.0 eine Implementation dieser Vorschläge vor und OGSA, die Abkürzung für *Open Grid Services Architecture* ist für alle, die sich mit Grid-Computing beschäftigen zum Teil des Standardvokabulars geworden.

In dieser Ausarbeitung zum gleichlautenden Vortrag, den ich am 26.06.2003 im Hasso-Plattner-Institut für Softwaresystemtechnik, im Rahmen des Seminars Grid-Computing, unter der Leitung von Prof. Dr. habil. Andreas Polze und Prof. Dr. Bettina Schnor, gegeben habe, werde ich auf die technischen Grundlagen und den Aufbau der *Open Grid Services Architecture* eingehen und zwei Implementationen vergleichen.

1.1 Begriffsübersicht

In der Diskussion um aktuelle und kommende Standards im Grid-Computing werden die Begriffe OGSA, OGSI und GT3 immer wieder auftauchen. In diesem Abschnitt soll es um eine Klärung dieser Begrifflichkeit gehen, da ich im weiteren Verlauf wieder darauf zurückkommen werde.

OGSA — *Open Grid Services Architecture*. Diese Architektur wie sie in [physiology2002] beschrieben wird, definiert, was *Grid Services* sind und beabsichtigt, als gemeinsame Standardarchitektur für Grid-Applikationen zu dienen. OGSA stellt keine genaue Beschreibung der technischen Erfordernisse oder Implementierungsansätze dar, sondern nur eine vergleichsweise offene Architekturbeschreibung.

OGSI — Die *Open Grid Services Infrastructure* dagegen ist eine detaillierte und formale technische Spezifikation der in der OGSA beschriebenen Konzepte, insbesondere von *Grid Services*.

GT3 — Das Globus Toolkit 3 ist eine Implementation der *Open Grid Services Infrastructure*. Die Komponenten des Globus Toolkit 3 sind lauffähige Implementierungen von allem, was in der OGSI spezifiziert wurde und damit auch von allem was in OGSA beschrieben wurde.

Wichtig zu bemerken ist, dass das Globus Toolkit zwar die einzige im Moment vollständige Implementierung ist, aber nicht die einzig denkbare. Es ist also möglich, anhand der OGSI-Spezifikation eine al-

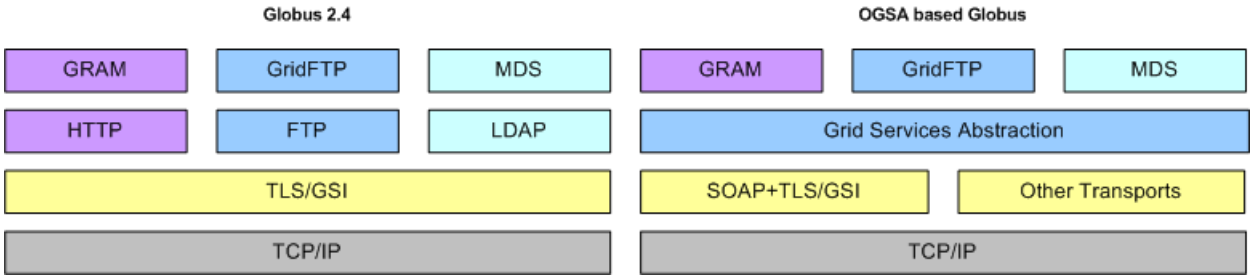
ternative Implementierung zu entwickeln, die dann zum Globus Toolkit kompatibel ist.

Grid Services Die *Grid Services*, von denen hier die Rede ist, sind eine Erweiterung des Konzeptes der *Web Services*, auf welches ich im Abschnitt 2, „Basistechnologien“ eingehen werde.

1.2 Globus Toolkit 3

Das besondere des Globus Toolkit 3 ist es, dass es die aus dem Globus Toolkit 2 bekannten Komponenten und Dienste wie GRAM, GridFTP und MDS[1]beinhaltet , es unterstützt mehrere mögliche Laufzeitumgebungen und stellt über die *Globus Security Infrastructure* Datensicherheit bei der Übertragung von SOAP-Nachrichten bereit.

Abbildung 1. Globus Komponenten und OGSA



Wie aus der Abbildung Abbildung 1, „Globus Komponenten und OGSA“ deutlich wird, nutzen die aus dem Globus Toolkit 2 bekannten Komponenten im Globus Toolkit 3 *Grid Services*, um von den jeweiligen Transportprotokollen zu abstrahieren.

1.3 Wozu braucht man OGSA?

Die Anwendungsumgebungen unterliegen einer starken Veränderung von homogenen, verlässlichen, sicheren und zentral verwalteten Umgebungen hin zu zunehmender Zusammenarbeit, mehr gemeinsame Nutzung von Daten und Interaktion. Dies alles geschieht über mehrere Netzwerke und administrative Domains hinweg.

Die derzeitigen Applikationen sind auf eine bestimmte Ausführungsumgebung abgestimmt, jedoch die fortschreitende Dezentralisierung und Verteilung von Ressourcen zwingt uns, eine hohe *Quality of Service* auf hochdynamisch zusammengesetzten Systemen zu gewährleisten.

Diese Probleme wurden insbesondere für Entwickler von stark rechenintensiven wissenschaftlichen Forschungsprojekten zum zentralen Fokus ihrer Arbeit. Durch gemeinsame Arbeit wurden Grid Technologien entwickelt, die dieses Problem lösen sollten, indem sie wie das *World Wide Web* standort- und plattformunabhängig Informationen

[1]Zur detaillierteren Besprechung dieser Themen empfehle ich das Lesen der Ausarbeitungen von Benjamin Kuppe, Alexis Krepp und Tobias Rausch sowie Dietmar Bremser und Udo Werner, die im Rahmen des gleichen Seminars entstanden sind.

bereit stellt, standort- und anwendungsumgebungsunabhängig Rechenkapazität zur Verfügung stellt.

Virtuelle Organisationen. OGSA strebt nicht nur an, die heutigen Probleme von Wissenschaftlern zu lösen, sondern die Bildung von Virtuellen Organisationen, den sogenannten *Virtual Organizations* zu erlauben. Diese hochdynamischen und verteilten Organisationen sind im Wesentlichen ein Zusammenschluss von Benutzern und Organisationen, die zu unterschiedlichen Netzwerken gehören und trotzdem Ressourcen in Form von Daten, Programmen oder Rechenkapazität gemeinsam nutzen möchten.

Dabei unterscheiden sich diese Organisationen durch eine große Bandbreite in Hinsicht auf die Größe der Organisation, die von zwei bis zu tausenden Mitgliedern reichen kann, der Lebenszeit, die Minuten bis Jahre betragen kann und dem Verhalten, welchen im Wesentlichen statisch, aber auch sehr dynamisch sein kann. Weiterhin gelten für die Mitglieder der Organisation, da sie zu unterschiedlichen administrativen Wirkungsbereichen gehören, auch unterschiedliche adminstrative Richtlinien.

Grid Technologien wurden zuerst eingesetzt, um in wissenschaftliche Projekten Ressourcen zu teilen, z.B. In dem man Zugang zu ungenutzen Rechenressourcen anbietet um so das verteilte Berechnen von ressourcenintensiven Applikationen zu ermöglichen.

Später kam eine Nutzung im *Business-to-Business*-Bereich hinzu, jedoch nicht um Rechenressourcen gemeinsam zu nutzen, sondern als eine Möglichkeit, neue verlässliche, skalierbare und sichere verteilte Systeme aufzubauen. Dies erlaubt die Dekomposition von hochintegrierter IT-Infrastruktur und das Reintegrieren von verteilten Dienstleistungen.

Eine Folge war die Entstehung verschiedener *Service Provider* wie *Web-Hosting Service Provider*, *Content-Distribution Service Provider*, *Application Service Provider* und *Storage Service Providern*.

2 Basistechnologien

Die Stärke der *Open Grid Services Architecture* liegt darin, dass bereits auf viele bestehende und etablierte Technologien zurückgegriffen wird, diese jedoch in einen neuen Kontext gesetzt und zu einem anderen Gesamtsystem zusammengesetzt werden.

Das hat den Vorteil, dass man Fehler die andere schon gelernt haben zu umgehen nicht in das neue System einfließen und man fremde Softwarekomponenten nutzen kann um den Aufwand zu Erstellung des eigenen Systems zu verringern. In diesem Abschnitt werde ich in die wichtigsten dieser Technologien einen kurzen Einblick geben und in der Bibliographie geeignete Ressourcen zur weiteren Vertiefung angeben.

2.1 Globus Toolkit

Die Kenntnis des Globus Toolkit ist eine nicht-normative Voraussetzung, da zwar das Globus Toolkit 3 darauf aufbaut, dieses jedoch nur eine von vielen Möglichkeiten darstellt, die OGSI-Vorgaben zu implementieren. Eine Wissen um die Grundbestandteile ist dennoch vorteilhaft, da die Entwickler der OGSA-Definition sich in weiten Teilen von den Erfahrungen haben leiten lassen, die sie bei der Entwicklung des Globus Toolkits gemacht haben.

Das Globus Toolkit ist eine Open-Source-Implementierung einer *Grid Service* Architektur, die im Wesentlichen einen Grundstock an Dienstleistungen anbietet (*bag of services*), eine Reihe von Software Bibliotheken, mit denen die Möglichkeiten dieses Frameworks genutzt werden können und verschiedene Grid-Applikationen, auf denen aufbauend man eigene Applikationen erstellen kann.

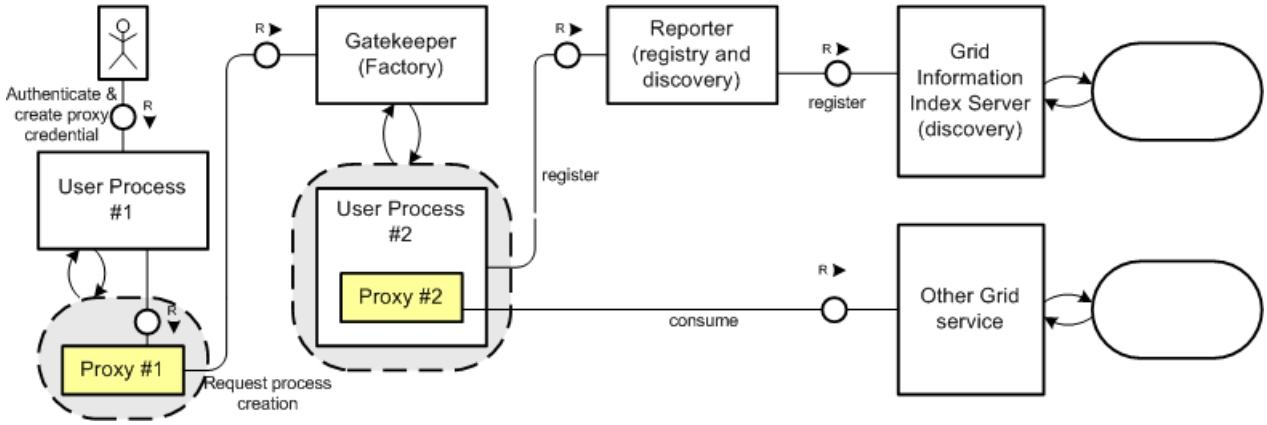
Das Globus Toolkit behandelt so wichtige Punkte wie: Sicherheit, Informationsfindung, Ressourcenverwaltung, Datenverwaltung, Kommunikation, Fehlererkennung und Portabilität. Im Bezug auf die Betrachtung von OGSA sind folgende Komponenten am wichtigsten:

GRAM	Der *Grid Resource Allocation Manager* übernimmt Ressourcen-Allokation und Prozessverwaltung.
Gatekeeper Service	dient als Haupteinstiegspunkt für auszuführende Berechnungsaufgaben
MDS-2	Mithilfe des *Metacomputing Directory Services* kann Globus verteilten Zugang zu Strukturen und Statusinformationen gewährleisten.
GRAM *reporter*	sammelt Informationen über die lokal laufenden Jobs und überträgt diese Informationen an den MDS
GSI	Die *Grid Security Infrastructure* stellt Einrichtungen zur Bewahrung der Datensicherheit bereit.

Beispiel 1. Globus Toolkit Aufbau

Die folgende Abbildung zeigt einen beispielhaften Aufbau für eine auf Globus basierende Grid-Anwendung, bei der ein Benutzerprozess einen lokalen Proxy erzeugt, um mit dem Grid zu kommunizieren. Dieser Proxy nimmt Kontakt zum *Gatekeeper* auf, der wiederum den Benutzerprozess 2 erzeugt, welcher selbst einen Proxy für die Kommunikation mit dem restlichen Grid bereitstellt.

Abbildung 2. Globus Toolkit Aufbau



Dieser zweite Benutzerprozess kann jetzt jeden beliebigen Grid-Service konsumieren und hat die Möglichkeit, sich beim MDS zu registrieren, damit selbst angebotene Dienste konsumiert werden können.

Im Rahmen des Seminars Grid-Computing, welches im Sommersemester 2003 von Prof. Dr. habil. Andreas Polze, Hasso-Plattner-Institut und Prof. Dr. Bettina Schnor, Institut für Informatik der Universität Potsdam angeboten wurde, wurde das Globus Toolkit in zwei Vorträgen detailiert behandelt, deshalb empfehle ich dem Leser, sich den diesbezüglichen Ausarbeitungen zuzuwenden, um weitere Details zu erfahren.

2.2 Web Services

Der zweite wichtige Baustein für die OGS-Archtitektur sind *Web Services*, da die im OGSA definierten *Grid Service* eine Erweiterung der *Web Services* darstellen.

> The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web Services.
>
> —World Wide Web Consortium

Web Services basieren auf anderen Web Standards wie HTTP, MIME-Typen oder XML, um das Ziel, einheitliche Kommunikationsschnittstellen für die Kommunikation zwischen Programmen bereitzustellen, zu gewährleisten.

Die Kernbestandteile von *Web Services*, auf die ich im Folgenden eingehen werde sind: Schnittstellenbeschreibung, XML Protokolle, Auffindung und Meta-Information über *Web Services*.

2.2.1 Schnittstellenbeschreibung

Damit Andere in der Lage sind, die bereitgestellten *Web Services* zu nutzen, ist es sinnvoll, eine standardisierte Schnittstellenbeschreibung bereitzustellen. Die *Web Services Description Language* (WSDL) ist der Standard, um *Web Services* zu beschreiben. Es handelt sich dabei um ein XML-Format, welches Informationen über

- Datentypen
- Nachrichten - welche Art von Daten enthält eine Nachricht
- portTypes - welche Operationen können wie ausgeführt werden
- Protokoll Anbindung - über welche Protokolle können die verschiedenen portTypes angesprochen werden und unter welcher Adresse können Sie angesprochen werden?

Beispiel 2. ein WSDL Dokument

```
<?xml version="1.0"?>

<!-- root element wsdl:definitions defines set of related services -->
<definitions name="StockQuote"
             targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

    <!-- wsdl:types encapsulates schema definitions of communication types;
         here using xsd -->
    <wsdl:types>

       <!-- all type declarations are in a chunk of xsd -->
       <xsd:schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">


           <!-- xsd definition: TradePriceRequest [... tickerSymbol string ...] -->
           <xsd:element name="TradePriceRequest">
              <xsd:complexType>
                  <xsd:all>
                      <xsd:element name="tickerSymbol" type="string"/>
                  </xsd:all>
              </xsd:complexType>
           </xsd:element>


           <!-- xsd definition: TradePrice [... price float ...] -->
           <xsd:element name="TradePrice">
              xsd:<complexType>
                  <xsd:all>
                      <xsd:element name="price" type="float"/>
                  </xsd:all>
              </xsd:complexType>
           </xsd:element>

       </xsd:schema>
    </wsdl:types>

    <!-- request GetLastTradePriceInput is of type TradePriceRequest -->
    <wsdl:message name="GetLastTradePriceInput">
        <wsdl:part name="body" element="xsd1:TradePriceRequest"/>
    </wsdl:message>

    <!-- request GetLastTradePriceOutput is of type TradePrice -->
    <wsdl:message name="GetLastTradePriceOutput">
        <wsdl:part name="body" element="xsd1:TradePrice"/>
    </wsdl:message>

    <!-- wsdl:portType describes messages in an operation -->
    <wsdl:portType name="StockQuotePortType">
```

```
    <!-- the value of wsdl:operation eludes me -->
        <wsdl:operation name="GetLastTradePrice">
           <wsdl:input message="tns:GetLastTradePriceInput"/>
           <wsdl:output message="tns:GetLastTradePriceOutput"/>
        </wsdl:operation>
    </wsdl:portType>

    <!-- wsdl:binding states a serialization protocol for this service -->
    <wsdl:binding name="StockQuoteSoapBinding"
                  type="tns:StockQuotePortType">

        <!-- leverage off soap:binding document style
             @@@(no wsdl:foo pointing at the soap binding) -->
        <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>

        <!-- semi-opaque container of network transport details
             classed by soap:binding above @@@ -->
        <wsdl:operation name="GetLastTradePrice">

           <!-- again bind to SOAP? @@@ -->
           <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
           <!-- furthur specify that the messages in the
                wsdl:operation "" use SOAP? @@@ -->
           <wsdl:input>
               <soap:body use="literal"/>
           </wsdl:input>
           <wsdl:output>
               <soap:body use="literal"/>
           </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <!-- wsdl:service names a new service "StockQuoteService" -->
    <wsdl:service name="StockQuoteService">
        <wsdl:documentation>My first service</documentation>

        <!-- connect it to the binding "StockQuoteBinding" above -->
        <wsdl:port name="StockQuotePort"
                   binding="tns:StockQuoteBinding">

           <!-- give the binding an network address -->
           <soap:address location="http://example.com/stockquote"/>
        </wsdl:port>
    </wsdl:service>

</wsdl:definitions>
```

Dieses WSDL-Dokument stellt einen Dienst bereit, der unter der URL (dem sogenannten *Endpoint*) http://example.com/stockquote angesprochen werden kann und die Operationen `getLastTradePrice` unterstützt, die per SOAP angesprochen werden kann.

2.2.2 XML Protokolle

Der zweite wichtige Bestandteil von *Web Services*, der für OGSA relevant ist, sind XML Protokolle mit denen in XML codierte Nachrichten zwischen Instanzen von *Web Service* Applikationen verschickt werden können.

Dabei ist das *Simple Object Access Protocol* von besonderer Bedeutung, da es der aktuelle Standard für Nachrichtenübertragung von *Web Services* ist. Wie sein Vorgänger XML-RPC besteht das Prinzip von SOAP darin, Nachrichten, bestehend aus einem Umschlag (SOAP *Envelope*), einem optionalen Kopf (SOAP *Header*) und einem Rumpf (SOAP *Body*) der die Daten, also Namen der aufzurufenden Methode sowie XML-codierte Übergabeparameter über HTTP oder andere Protokolle tieferer Ebene zu übertragen.

Beispiel 3. SOAP Request und Response

Ein *Client*-Programm könnte entsprechend der Schnittstellenspezifikation aus Beispiel 2, „ein WSDL Dokument“folgende Anfrage an den *Endpoint* senden:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Eine mögliche Antwort für diese Anfrage könnte folgender Codeausschnitt sein:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Durch Applikationsframeworks wie .NET von Microsoft oder Axis der Apache Software Foundation, die von einigen OGSI-Implementationen benutzt werden ist es möglich, für den Programmierer transparent SOAP Nachrichten aus Objekten zusammenzusetzen und empfangene Nachrichten wieder in Objekte umzusetzen. Weiterhin unterstützen diese Frameworks die Möglichkeit, aus einer gegebenen Schnittstellenbeschreibung Proxy-Klassen zu erzeugen, die einen noch einfacheren Umgang mit SOAP Nachrichten ermöglichen.

Durch den herstellerunabhängigen offenen Standard gibt es sehr viele verschiedene Hersteller von SOAP-Frameworks und Bibliotheken die untereinander weitestgehend kompatibel sind.

2.2.3 Auffindung von Diensten

Ein weiteres Erfordernis für *Web Services* im Allgemeinen und *Grid Services* im Besonderen ist die Auffindbarkeit von Dienstleistern. Dazu existieren zwei Spezifikationen.

2.2.3.1 Einfache Auffindung

In der Nutzung für *Web Services* weniger verbreitet, jedoch von großer Bedeutung für die OGSI-Spezifikation ist die *Web Service Inspection Language* (WSIL), die von Microsoft und IBM entwickelt und befördert wird. Dokumente in der WSIL werden an leicht zugänglichen URLs bereitgestellt und unter Umständen von Web Seiten verlinkt und sie enthalten Informationen darüber, wo Schnittstellenbeschreibungen (WSDL), *Web Service* Verzeichnisdienste (UDDI) oder andere WSIL-Dateien gefunden werden können.

Eine zu dem bekannten *Stock-Quote*-Beispiel passende WSIL-Datei könnte folgendermaßen aussehen:

Beispiel 4. WSIL-Datei

```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
            xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/">
  <service>
    <abstract>A stock quote service with two descriptions</abstract>
    <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                 location="http://example.com/stockquote.wsdl"/>
    <description referencedNamespace="urn:uddi-org:api">
       <wsiluddi:serviceDescription location="http://www.example.com/uddi/inquiryapi">
         <wsiluddi:serviceKey>
           4FA28580-5C39-11D5-9FCF-BB3200333F79
          </wsiluddi:serviceKey>
       </wsiluddi:serviceDescription>
    </description>
  </service>
  <service>
    <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
                 location="ftp://anotherexample.com/tools/calculator.wsdl"/>
  </service>
  <link referencedNamespace="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
        location="http://example.com/moreservices.wsil"/>
</inspection>
```

2.2.3.2 Verzeichnisdienste

Eine andere Methode zur Auffindung von *Web Service* Dienstleistern liegt in der Nutzung von global bekannten *Web Service* Verzeichnisdiensten, die den UDDI-Protokollsatz benutzen.

> The focus of Universal Description Discovery & Integration (UDDI) is the definition of a set of services supporting the description and discovery of businesses, organizations, and other Web services providers, the Web services they make available, and the technical interfaces which may be used to access those services.
>
> —UDDI V3 Specification

Universal Description Discovery and Integration stellt einen Standard, aufbauend auf HTTP, XML, XML Schema und SOAP, bereit, über den man *Web Services* registrieren und registrierte Dienste auffinden kann.

2.2.4 Sicherheit von Web Services

Das Versenden von Nachrichten in Form von XML-Dokumenten, die leicht zu interpretieren sind fordert besondere Sicherheitsvorkehrungen.

Sichere Transportwege. Die Nutzung sicher Transportwege wie *Transport Layer Security* (HTTPS) oder das Tunneln der Verbindung über SSH ist die einfachste Maßnahme zur Gewährleistung von Sicherheit bei *Web Services*.

WS-Security. IBM, Verisign und Microsoft fördern die Entwicklung und Umsetzung einer Spezifikation für Sicherheit in *Web Services* (*Web Services-Security*), die einen allgemeinen Mechanismus bereit stellen soll, „*Security Tokens*“ in SOAP Nachrichten mitzusenden, so dass es möglich wird, Nachrichten digital zu signieren oder Teile von Nachrichten mit einem Private/Public-Key-Verfahren zu kodieren. Die Spezifikation soll erweiterbar sein und beschreibt, wie X.509 Zertifikte oder Kerberos Zertifikate kodiert werden sollen. Zur Zeit liegen jedoch nur Beispiel-Implementierungen in Alpha-Qualität vor.

3 Use-Cases für OGSA

Ein bedeutender Punkt des Aufgabenbereichs der OGSA-Arbeitsgruppe ist es, Use-Cases für die *Open Grid Services Architecture* vorzustellen und damit den Standardisierungsprozess voranzutreiben, Anhaltspunkte für die Gewichtung der Priorität einzelner Komponenten zu liefern und eine Möglichkeit zu geben, Standardisierungsentscheidungen zu begründen.

Im Folgenden möchte ich auf zwei dieser Use-Cases in aller Kürze eingehen, um damit das Zusammenspiel der einzelnen Komponenten verdeutlichen zu können und dem Leser Anhaltspunkte für die Möglichkeit des praxisrelevanten Einsatzes der OGSA-Technologien zu geben.

3.1 Kommerzielles Rechenzentrum

Um die Total Cost of Ownership ihrer IT-Infrastruktur zu senken, fassen viele Unternehmen eine große Zahl ihrer Server in Rechenzentren (engl. Data Center) zusammen. Außerdem lagern einige Firmen Teile ihrer IT-Infrastruktur aus, um im Kerngeschäft bessere Leistungen zu erzielen.

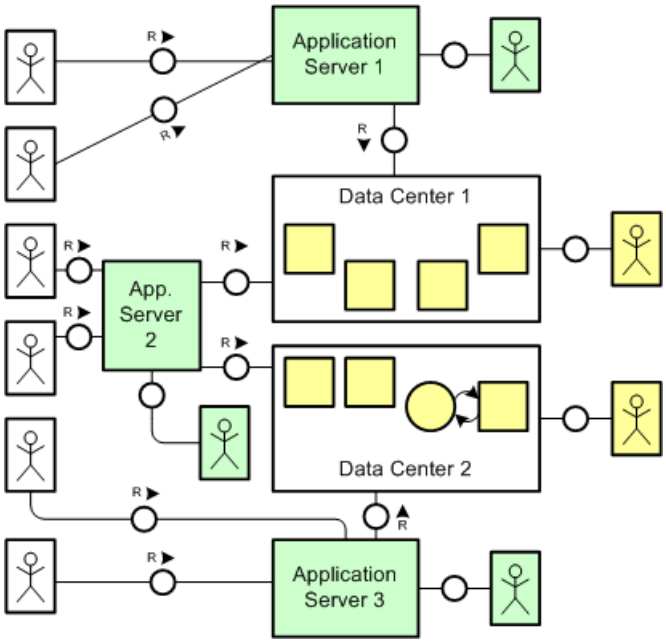
Der *Kunde* des Rechenzentrums ist der „*IT business activity manager*“, der zum Beispiel einen Konzertkartendienst betreibt und Konzertkarten an *Endbenutzer* verkauft. Diese Endbenutzer sind Akteure des kommerziellen Grid, aber keine Kunden.

Der IT business activity manager kann jetzt gewisse Aufgaben an das Grid übermitteln, um sie dort ausführen zu lassen. Diese Aufgabentypen können sein:

- **Java Programme.** In vielen modernen Applikationen liegt die Geschäftslogik in Form von Java Programmen, z.B. als Enterprise Java Bean oder als Servlet vor. Diese Programme erwarten ein bestimmtes Hosting Environment, also einen EJB-Container oder eine Servlet Engine, um ausgeführt zu werden.
- **Stapelverarbeitungsprogramme.** Ein weiterer Anwendungsbereich sind Geschäftsprozesse, auch wiederkehrende, die keine Benutzerinteraktion erfordern und somit von Stapelverarbeitungssystemen abgearbeitet werden können. Ein Beispiel wäre eine monatliche Erzeugung von Berichten über Lohn- und Gehaltszahlungen an Arbeitnehmer. Unter bestimmten Umständen kann auch ein Workflow-Management erforderlich sein.
- **Zusammengesetzte Systeme.** Es soll auch möglich sein, eine komplette Geschäftsanwendung, z.B. in Form zweier Web-Server, drei Applikationsservern und einer Datenbank als einen Arbeitsauftrag an das Grid zu übermittteln. Solche Art von Anfragen erfordert die Zusammensetzung eines Systems, welches in der Lage ist, derartige Applikationen laufen zu lassen und das Deployment dieser Applikationen zu übernehmen.

Zwei weitere zu betrachtende Rollen in diesem Use-Case haben der „IT System Itegrator“ und der „IDC GRID Administrator“ inne. Ersterer ist ein indirekter Kunde des Grid, da er die Aufgabe hat, heterogene Systeme zu konstruieren, und gleichzeitig Garantien über Zuverlässigkeit und Verfügbarkeit abzugeben. Zweiterer stellt die eigentlichen Dienste bereit, und hat Interesse daran, die Auslastung seiner Systeme zu erhöhen und vom Grid ein Form einer leichteren Wartbarkeit seines Grid-Knotens zu profitieren. Das Rechenzentrum, welches der IDC GRID Administrator verwaltet, kann Dienste in Form von Speicherplatz, Rechenzeit und Anwendungen bereitstellen.

Abbildung 3. Commercial Data Center



In der obigen Abbildung kann man die einzelnen Rollen sehr gut erkennen. Die Endnutzer (weiß) auf der rechten Seite nehmen Dienste verschiedener Application Server (grün) in Anspruch. Diese Application Server werden von dem *IT System Integrator* verwaltet, welche die Verbindung zu den einzelnen Data Center herstellen. Diese Data Center werden vom IDC GRID Administrator (gelb) verwaltet und stellen Datenhaltungsdienste und Datenverarbeitungsdienste bereit.

Die Aktivitäten, die im Grid zur Ausführung eines einfachen Java-Programms oder eines Stapelverarbeitungssystems oder eines zusammengesetzten Systems durchgeführt werden müssen sind sich sehr ähnlich. Ich werde im Folgenden auf die einzelnen Aktivitäten eingehen und bei Unterschieden diese genauer betrachten.

Aktivitäten zur Nutzung kommerzieller Rechnenzentren

1. **Auffindung.** Um einen Dienst nutzen zu können, muss der Kunde zuerst eine Referenz auf das GRID erhalten, welches er nutzen möchte, dazu kann er eine der zuvor beschriebenenen Auffindungsmöglichkeiten für *Web Services* nutzen.
2. **Authentifizierung, Authorisierung und Abrechnung.** Sobald der Kunde eine Anfrage abgeschickt hat, authentifiziert das Grid den Nutzer und authorisiert die übertragene Anfrage.
3. **Planung des zeitlichen Ablaufs der Aufgaben des Dienstes.** Ausgehend von den Planungsstrategien des Kunden kann das Grid entscheiden, wann die Abarbeitung der Anfrage beginnen kann. Das Grid interpretiert dabei eine in einer Aufgabenbeschreibungssprache verfasste Beschreibung der Anfrage und überprüft, ob der Kunde die Berechtigungen hat, diese Anfrage auszuführen.
4. **Workflow Management.** Für den Stapelverarbeitungsauftrag muss sich das *Workflow Management* um den Arbeitsablauf der Anfrage kümmern.

Workflow Management. Im Falle eines zusammengesetzen Systems ist das Workflow Management eine besonders anspruchsvolle und komplizierte Aufgabe, für die intern der *Grid Service Orcestration Service* genutzt wird.

5. **Vermittlung der Ausführungsknoten.** Das Grid vermittelt ausgehend von den Anforderungen des auszuführenden Programms und den Vermittlungsstrategien des Kunden die bestgeeigneten Ressourcen für den jeweiligen Ausführungszeitraum. Die Ressourcen werden reserviert und eine Referenz auf sie wird an den Kunden übermittelt.
6. **Data Sharing.** In den meisten Fällen werden die benötigten Daten, seien es Datenbanken oder Dateien, in der Anfrage spezifiziert. Die Zugänglichkeit der benötigten Daten muss von dem vorhergehenden Vermittlungsdienst übernommen werden. Weiterhin kann ein *Datenverwaltungsdienst* sich um die notwendige Datenreplikation und Migration kümmern.
7. **Planung des zeitlichen Ablaufs der Aufgaben des Dienstes und Auffindung.** Kurz vor der reservierten Ausführungszeit beginnt das GRID damit, den Dienst einzusetzen. Das Grid findet das gewünschte Programm (z.B. in Form einer JAR-Datei) und startet das *Deployment* auf den reservierten Ressourcen. Für Java-Programme ist das *Deployment* bereits sehr gut definiert und wird von den meisten Ausführungsumgebungen umgesetzt.

   Management des Altsystems. Für den Stapelverarbeitungsdienst ist es außerdem nötig, Altsysteme (*Legacy Application*) einzusetzen. (Nur Stapelverarbeitungssysteme)
8. **Datentransport.** Bestimmte Teile des Arbeitsablaufs des Stapelverarbeitungsdienstes können bestimmte Daten erfordern, die zwischen Servern ausgetauscht werden. Der Datendienst (FTP) kopiert die Dateien auf die gewünschten Server.
9. **Fehlermanagement.** Für den Fall, dass der Kunde im Fall eines Fehlers in der Ausführung seiner Applikation benachrichtigt werden möchte, so kann der *ressource service* und der *instrumentation and monitor service* Hardware- und Softwarefehlerereignisse veröffentlichen. Der Kunde muss sich, um benachrichtigt zu werden, auf diese Veröffentlichungen subskribieren.

   Neustart des Stapelverarbeitungsdienstes. Für den Fall, dass die Abarbeitung auf einem Knoten fehlschlägt, kann das Grid einen Neustart der Stapelverarbeitung auf einem anderen Server veranlassen.
10. **Planung des zeitlichen Ablaufs der Aufgaben des Dienstes.** Mit Beginn der reservierten Zeit setzt das GRID den Dienst in Gang. Während der Ausführungszeit misst der Abrechnungsdienst die Nutzung der Ressourcen, um sie dem Kunden in Rechnung stellen zu können. Wenn ein Fehler auftritt, wird er durch das Fehlermanagement behandelt. Wenn die Aufgabe erfolgreich abgearbeitet wurde, beauftragt der Planungsdiesnt die Ressourcenverwaltungsdienst damit, die Ressourcen freizugeben.
11. **Authentifizierung, Authorisierung und Abrechnung.** Die Nutzung der Ressourcen wird zusammengerechnet und dem Kunden berechnet.

Sicherheitsbetrachtungen. Jedes einzelne kommerzielle IT-System (welches einer Virtuellen Organisation entspricht) sollte sicher isoliert von allen anderen Systemen laufen, da konkurrierende Unternehmen Dienste des gleichen Rechenzentrums (Reale

Organisation) in Anspruch nehmen können. Deshalb sollte man, bevor man ein kommerzielles System startet, die Virtuellen Organisationen durch die Nutzung von VPN-Technologie voneinander trennen. WS-Security ist ein Ausgangspunkt für weitergehende Datensicherheit im kommerziellen Grid, auch wenn Erweiterungen notwendig sein könnten.

Performance-Betrachtungen. Anders als bei wissenschaftlichen Grid-Anwendungen genießt der Durchsatz nicht die höchste Priorität, sondern *Quality of Service*. Aus diesem Grund wird auf die Angabe von Deadlines für Aufgaben zurückgegriffen, die mithilfe erweiterter Ressourcen-Planungs-Algorithmen zu einer möglichst effektiven Nutzung der Ressourcen führen können. Weiterhin können Systemadministratoren *Service Level Agreements* (SLA) herausgeben, aufgrund derer jeder Job unter hoher Last zusätzliche Ressourcen beantragen kann, oder Grid-Knoten, im Falle das die Ressourcen nicht ausreichen, niedrigpriore Jobs ablehnen können.

Lebenszeit. Die Lebensspanne von kommerziellen Grid-Systemen kann wie bei den meisten *Batch*-Prozessen recht kurz sein, oder aber für zusammengesetzte Systeme, die z.B. einen *Web Service* anbieten auch sehr lang sein.

Momentan gibt es einige Technologien und Produkte wie das Océano Project von IBM [oceano2003] oder Jareva [jareva2003], die bereits versuchen Teilaspekte dieses Szenarios abzudecken, aber einen proprietären Ansatz begrenzter Reichweite verfolgen. Die *Open Grid Services Architecture* hingegen ist eine offene, erweiterbare und verständliche Architektur, die genutzt werden kann, um dieses Probleme zu lösen.

Zur Zeit wird an der OGSA-konformen Umsetzung dieses Use-Cases noch geforscht, aber ein OGSA-basierter Prototyp eines kommerziellen Grid ist geplant.

3.2 Sturmvorhersage

Kunden. Eine Gruppe von Meteorologen und Klimaforschern versucht ein Grid aufzubauen, um den genauen Ort von starken Stürmen wie Tornados ausgehend von einer Kombination aus großflächiger Echtzeitwetterbeobachtung und Wettersimulationen im großen Umfang, gekoppelt mit Datenmodellierungstechniken, vorherzusagen. Die Virtuelle Organisation ist weit verteilt und viele Teilnehmer sind mobil.

Die Szenario ist in etwa Folgendes: Die Messergebnisse vieler, im ganzen Land verteilter Instrumente werden kontinuierlich verglichen und nach bekannten Mustern durchsucht. Tritt ein Muster auf, welches auf die Entstehung eines ernsten Sturms hinweist, werden die Mitglieder der Virtuellen Organisation benachrichtigt und eine große Zahl von Simulationen wird gestartet.

Data Mining Tools werden genutzt, um die Ausgabe der Simulationen zu durchsuchen und die Resultate gegen die weiteren Messergebnisse zu prüfen, außerdem wird in Datenarchiven nach ähnlichen Mustern gesucht. Einige der Instrumente werden automatisch neu konfiguriert, um genauere Messungen vornehmen zu können.

Wenn der Sturm sich ausweitet, werden weitere Simulationen gestartet, um die Genauigkeit der Vorhersagen zu erhöhen. Die Ausgabe der Simulationen kann in Form von Animationen visualisiert werden, um während mobile Messstationen weitere Daten eingeben, die Medien und die Behörden über die Vorhersagen zu informieren.

Machbarkeit. Dieses Szenario lässt sich zur Zeit noch nicht umsetzen, weil die nötige Infrastruktur noch nicht vorhanden ist. Zur Zeit existieren die Komponenten bereits,

aber sie sind noch nicht entsprechend integriert. Die derzeitige Aktivität der diesbezüglichen Arbeitsgruppe besteht darin, die Simulationen und das *Data-Mining* zu testen und die Simulationen mit den bestehenden Datenströmen zu integrieren.

3.2.1 Genutzte Ressourcen

Die wichtigsten genutzen Ressourcen sind:

1. Das Instrumentennetzwerk, welches verschiedenen Betreibern gehört
2. Das Datenarchiv der Messungen vorhergegangener Stürme
3. Die Rechenressourcen, die auch das Terragrid einschließen sollen

Folgende Dienstleistungen sollen erbracht werden:

- Ein integriertes Grid erlaubt es den Mitgliedern der Virtuellen Organisation, die Simulationswerkzeuge und *Data-Mining-Tools* zu nutzen und Datenarchive zu lesen sowie Messergebnisse des Sensorennetzwerkes in Erfahrung zu bringen.
- Unter Umständen ein automatisierter, autonomer *Grid Service*, der das oben beschriebene Szenario ausführen kann

Erforderliche Dienste

1. Verwaltung der Virtuellen Organisation. Es ist sehr wichtig, festlegen zu können, wer Zugang zu welchen Teilen der Messgeräte, Daten oder Rechenressourcen hat.
2. Sicherheit (Authentifikation und Authorisierung) der Mitglieder der Virtuellen Organisation, auch für mobilen Zugang.
3. Datagrid Services: Kataloge von Metadaten, Verzeichnis- und Indizierungsdienste, Grid-weiter Zugang zu Datenarchiven, virtuelle Datenverwaltung.
4. Grid-weites Überwachen und Benachrichtigen sowie Event- und Logging-Services.
5. Die meisten Instrumente benötigen *Web Service* Schnittstellen.
6. *Workflow engines*, die die verknüpften Simulations- und Datamining- und Planungsaufgaben aufeinander abstimmen
7. Vermittlungsdienste für Rechen- und Datenressourcen. Außerdem werden Scheduling-Dienste benötigt.

3.2.2 Weitere Betrachtungen

Sicherheitsbetrachtungen

- Der Zugang zu weiten Teilen des Sensorennetzwerkes ist öffentlich, jedoch die Möglichkeit es umzukonfigurieren ist auf sehr wenige Teilnehmer beschränkt.
- Simulation und *Data-Mining* laufen auf Supercomputern, die spezielle Authorisierung erfordern.
- Die Virtuelle Organisation muss in der Lage sein, Ressourcen so weit wie möglich gemeinsam zu nutzen.
- Es ist von essentieller Wichtigkeit, Mitglieder der Virtuellen Organisation, die von mobilen Geräten aus am Grid teilnehmen zu authentifizieren.

Performance-Betrachtungen. Das Ziel ist es, Vorhersagen zu machen, die das Besser-als-Echtzeit-Kriterium erfüllen. Daher ist ein hoher Durchsatz und die dynamische Neuverteilung von Ressourcen von höchster Bedeutung.

Lebenszeit. Dieses Grid muss jederzeit verfügbar sein.

4 Aufbau nach Spezifikation

Wie bereits erwähnt, liefert die *Open Grid Services Architecture* nur eine, ausgehend von den Anforderungen an eine Grid-Umgebung, Definition dessen, was *Grid Services* sind, macht jedoch keine Aussage über die Implementierung dieser.

In diesem Abschnitt werde ich zuerst auf die Anforderungen eingehen, dann zeigen, welche Ableitungen daraus für die Definition der OGSA gezogen wurden und schließlich die genaue Spezifikation in Form der *Open Grid Services Infrastructure* betrachten.

Für das Verständnis der folgenden Ausführungen ist es wichtig, die Nutzung der Begriffe „*Service*“ und „*Virtual Organization*“ zu verinnerlichen.

Service. Im Kontext des OGSA-basierten Grid-Computing ist ein Service (Dienst) ein netzwerk-fähiges Gebilde, die zu irgend etwas befähigt ist. Man könnte auch den Begriff „Objekt“ nutzen, doch wird dieser aufgrund überladenener Bedeutung vermieden.

Virtual Organisation. Nach [anatomy2002] stellt eine virtuelle Organisation das flexible, sichere und koordinierte Bereitstellen und Nutzen von Ressourcen, an dem sich ändernde Gruppen von Individuen und Organisationen beteiligt sind, und bei dem sich die zu nutzenden Ressourcen ebenfalls dynamisch ändern.

4.1 Anforderungen an eine Grid-Umgebung

Foster, Kesselman, Nick und Tuecke sind in [pysiology2002] von den Erfordernissen des Grid-Computing im Wissenschaftsbetrieb und im e-Business ausgegangen und haben folgende wichtige Anforderungen identifiziert.

A. **Dynamische Erzeugung von Diensten.** Teilnehmer Virtueller Organisationen wollen mehr als persistente und statische Dienste, bei denen jegliche Interaktion

komplett von jeder anderen getrennt werden kann. Stattdessen wollen Sie in die Lage versetzt werden, Instanzen von Diensten zu erzeugen, um die Verwaltung und der Interaktion zu übernehmen, die mit dem *Zustand* bestimmter Aktivitäten verknüpft ist. Um die Erzeugung dynamischer Dienste zu erlauben, nutzt OGSA das Konzept der *Service Factory*, also einem Dienst den man nutzen kann, um Exemplare anderer Dienste zu erzeugen. Das der erzeugte Dienst immer eine begrenzte Lebenszeit hat ist dabei kein Problem, denn die diensterzeugende Fabrik ist ein persistenter Dienst, so dass man jederzeit ein neues Exemplar des gewünschten Dienstes erzeugen kann.

B. **Dynamische Verwaltung von Diensten.** Weiterhin benötigt man einen Mechanismus, um einen Dienst den man erzeugt hat, zu identifizieren, z.B. um seinen Zustand zu untersuchen oder um aus der Zusammenfassung der Resultate meherer asynchron ausgeführten Diensten neue Ergebnisse zu gewinnen. OGSA nutzt dazu den `HandleResolver` portType, die *Grid Service References* (GSR) und die *Grid Service Handles* (GSH).

   Nachdem ein Dienst erzeugt wurde, stellt GSH eine Methode dar, alle OGSA Dienste zu benennen, aber es gibt einem keine Möglichkeit, den Dienst zu nutzen, da man dafür die *Grid Service Reference* braucht. Um von einem gegebenen *Grid Service Handle* auf die *Grid Service Reference* zu kommen stellt OGSA das Konzept des `HandleResolvers` bereit, der ein GSH annimmt und eine GSR zurückgibt.

   Damit man den richtigen HandleResolver nutzt, um eine gegebene GSH umzuwandeln, schreibt OGSA vor, dass jeder *Grid Service Handle* einen *home handle resolver* für diesen Dienst enthält. Da eine beliebiger `HandleResolver` um eine beliebige Dienstinstanz wissen *kann*, aber der *home HandleResolver* immer um den zugehörigen Dienst weiß, ist sichergestellt, dass man jederzeit aus einer GSH einen GSR bestimmen kann.

   Dies bedeutet, dass die *Factory*, welche den Dienst erzeugt, auch eine GSH und ein GSR erzeugen muss. Die Fabrik kann dann den *home HandleResolver* auswählen und überprüfen, dass der *home HandleResolver* den GSH in die entsprechende GSR auflösen kann.

C. **Upgrade und Kompatibilitätsaspekte.** Benötigt wird ein Mechanismus, um Dienste mit der Zeit auf neuere Versionen upgraden zu können und es muss klar werden, ob der neue Dienst immer noch kompatibel zur alten Version ist. Es ist nicht möglich zu garantieren, dass ein Dienst zu einer bestimmten Zeit aktualisiert wird. Ein Kunde, der mit einem Dienst interagiert, der gerade aktualisiert wird, sollte im besten Falle nicht merken, dass eine Aktualisierung stattgefunden hat. Wenn das Upgrade jedoch zur Folge hatte, dass neue Schnittstellen, Protokolle und neue Funktionen unterstützt werden, so kann der Kunde jetzt auch diese nutzen.

   Aus diesem Grund muss eine Infrastruktur bereitgestellt werden, um dem Kunden zu erlauben, dieser Änderungen gewahr zu werden. In OGSA wird dies durch die Bereitstellung des *Grid Service* portType. GSR können dieses Problem, da sie bei einem Upgrade effektiv ungültig werden, da der Dienst im Grunde ein anderer ist,

nicht lösen. Es ist nicht bekannt, welche Dienste alles die GSR halten, so dass die Halter nicht informiert werden können.

Das *Grid Service Handle* hingegen ist die *Grid Service Reference* ohne spezifische Informationen über Protokoll und die jeweilige Instanz des Services, das GSH ist einmalig für jedes Exemplar des Dienstes und wird sich auch über einen Upgrade nicht ändern, deshalb kann man das GSH nutzen, um alle Nutzer des Dienstes über den Upgrade zu informieren und das GSH bleibt nach dem Upgrade immernoch gültig.

Außerdem bekommt jede GSR eine Lebenszeit zugeordnet. Während dieser Zeit kann das GSR wiederverwendet werden und es ist nicht nötig, bei jeder Anfrage ein GSH in ein GSR zu wandeln. Natürlich kann es vorkommen, dass ein GSR ungültig wird, bevor die angegebene Lebenszeit abgelaufen ist, aber dann kann der Umwandlungsprozess erneut gestartet werden, ohne die Funktionalität zu beeinträchtigen. Der Effekt ist, dass die zusätzlichen Kosten der Umwandlung von GSH in GSR in vielen Fällen entfallen, bevor man einen Dienst konsumieren möchte.

D. **Lifetime Management.** Da jede Dienstinstanz vergänglich und dynamischer Natur ist, muss es einen Weg geben, sie zu löschen, wenn Sie nicht länger benötigt wird. In einer Grid-Umgebung kann es aber vorkommen, dass die Aufforderung zur Zerstörung aufgrund eines unverlässlichen Transports in einer verteilten Umgebung das Ziel nie erreicht. Aus diesem Grund löst OGSA dieses Problem durch Bereitstellung einer „*soft state registration*“ im GridService portType. Das bedeutet, dass mit jedem erzeugten Dienst eine maximale Lebenspanne verknüpft wird. Wird die Lebensspanne überschritten, wird der Dienst beendet. Es kann jedoch jeder, entsprechende Berechtigungen vorausgesetzt, eine Verlängerung der Lebenszeit beantragen. Wird diese Verlängerung abgelehnt, so kann man den Dienst auf einem anderen Knoten replizieren.

E. **Registrierung und Auffindung von Diensten.** Eine weitere Notwendigkeit ist die Standardisierung der Informationen über einen Dienst und ein Mechanismus, mit dem man gleiche oder ähnliche Dienste auffinden kann. Das ermöglicht es, zwischen Instanzen von Diensten zu wählen und anhand von Merkmalen wie *Quality of Service*, angebotenen Funktionen oder dem jeweiligen Zustand des Dienstes zu entscheiden.

Dieses Problem wird von OGSA gelöst, indem Elemente mit Dienstinformationen im GridService portTpye und im Registration portType bereitgestellt werden. Es deutet an, dass eine Registerdatenbank bereitstehen soll, die Dienstbeschreibungen enthält und die Auswahl eines Dienstes ermöglicht. Diese Datenbank gibt eine GSH zurück, die genutzt wird, um eine GSR zu erhalten, mit der die Instanz des Dienstes aufgerufen werden kann.

F. **Benachrichtigung.** Eine Gruppe verteilter Dienste muss in der Lage sein, sich gegenseitig asyncron über Zustandsänderungen zu informieren. Es sollte einen grundlegenden Standardmechanismus geben, es er erlaubt, Benachrichtigungen eines Dienstes zu abonnieren und diese Nachrichten zu übertragen.

Die Lösung für OGSA wird durch die Elemente des Notfication-Source, Notification-Sink und Notifikation-Subscription portType bestimmt.

Weitere Anforderungen wie Authorisierung und Nebenläufigkeit sollen später von OGSA adressiert werden und Authentifizierung und die Verlässlichkeit des Aufrufs von Diensten sind Protokoll-abhängig.

4.2 Schnittstellen

OGSA definiert eine Reihe von Schnittstellen, die eine OGSA-konforme Grid-Applikation ausmachen. Davon ist nur die Schnittstelle GridService obligatorisch.

Von besonderer Bedeutung zum Verständnis der folgenden Schnittstellen ist die Unterscheidung von *Grid Service Handle* und *Grid Service Reference*, die im vorhergehenden Abschnitt schon kurz erläutert wurde.

Grid Service Handle. Da ein *Grid Service* ein *Web Service* ist, und *Web Services* immer durch eine URI angesprochen werden können, hat auch jeder *Grid Service* eine URI. Diese für jede Instanz eines *Grid Services* einzigartige Identifikation nennt man *Grid Service URI* oder, gebräuchlicher *Grid Service Handle*.

Grid Service Reference. Ein *Grid Service Handle* enthält zwar Informationen darüber, wo der Service aufzurufen ist, aber keine Informationen wie, also welche Protokolle, Methoden und Datentypen ausgetauscht werden. Diese Informationen enthält die *Grid Service Reference*, welche in den meisten Fällen ein WSDL-Dokument ist, welches man durch Anhängen von `.gsr` oder `?WSDL` an die *Grid Service URI* erhält.

4.2.1 Grid Service

Ein Grid-Service ist ein Web-Service, der einen Satz wohldefinierter Schnittstellen bereit stellt und bestimmten Konventionen folgt. Diese Schnittstellen umfassen Punkte wie Auffindung, dynamische Erzeugung von Diensten, Lebenzeitverwaltung, Benachrichtigung und Upgradebarkeit.

Jeder Grid Service hat nur eine begrenzte Lebenszeit und einen Zustand. Um diesen Fakt darzustellen hat der grundlegende Grid Service den portType `GridService` und unterstützt folgende Operationen

`FindServiceData`	Liefert eine Reihe von Informationen über den Dienst zurück. Diese Informationen sind unter anderem: Handle, Reference, Primärschlüssel, home handleMap. Weitergehende Informationen über jede einzelne Schnittstelle und Dienst-spezifische Informationen wie eine Liste der Exemplare des Dienstes, die seiner Registry bekannt sind. Diese Schnittstelle soll für verschiedene Sprachen erweiterbar sein.
`SetTerminationTime`	Über diese Schnittstelle kann man die Zeit bestimmen, zu der ein Service zerstört wird und eine Verlängerung der Lebenszeit beantragen.

`Destroy` Zerstört den Grid Service, bevor die maximale Lebenszeit abgelaufen ist. Die maximale Lebenszeit ist ein Rückfallmechanismus für den Fall, dass diese Nachricht nicht ankommt.

4.2.2 Factory

Der portType Factory muss wie alle anderen die grundlegenden Schnittstellen des portType GridService implementieren. Dieser Grid Service behandelt die Anforderung, dynamisch Dienste erzeugen zu können, damit unterschiedliche Konsumenten identischer Dienste den Status des Dienstes getrennt halten können.

In der Praxis werden Clients immer zuerst eine Factory beauftragen, ein neues Exemplar des gewünschten Dienstes zu erzeugen, dann mit diesem Dienst zu arbeiten und ihn nach getaner Arbeit mit `Destroy` zu zerstören.

`CreateService` Erstellt ein neues Exemplar eines *Grid Services*. Gibt Grid Service Handle und *Grid Service Reference* zurück. Aus der Grid Service Reference kann die home HandleMap bestimmt werden.

4.2.3 Registry

Der Registry portType ist das Standardinterface, um Informationen über Instanzen von *Grid Services* zu registrieren und abzufragen.

`RegisterService` Registriert eine *Grid Service Handle*

`UnRegisterService` Löscht die Registrierung des *Grid Service Handle*s aus der Registrierdatenbank

4.2.4 HandleMap

Mit der HandleMap kann aus einer gegebenen *Grid Service Handle* eine *Grid Service Reference* ermittelt werden. Voraussetzung ist, dass der jeweilige Grid Service der HandleMap bekannt ist. Deshalb hat jeder Grid Service eine eigene HandleMap, die die jeweilige GSH und GSR kennt. Dies nennt man die home HandleMap.

`FindByHandle` Gibt die *Grid Service Reference* zurück, die gerade dem übergebenen *Grid Service Handle* zugeordnet ist

4.2.5 NotificationSource

Die Quelle von Benachrichtigungen. *Grid Services* können Nachrichten abonnieren und werden dann bei entsprechenden Ereignissen benachrichtigt.

`SubscribeToNotificationTopic` Subskribiert einen Dienst (dies kann auch ein dritter sein), ausgehend vom Typ der Nachricht und einem „*Interest Statement*“.

4.2.6 NotificationSink

Grid Services vom portType NotificationSink sind in der Lage, Nachrichten zu übertragen. Sie müssen nicht zwangsläufig der Empfänger der Nachricht sein.

`DeliverNotification`	Führt eine asynchrone Übertragung der Nachrichten durch.

4.3 Sicherheitsaspekte

Die Sicherheitsaspekte der *Open Grid Services Architecture* sind ein derart umfangreiches Thema, dass sich eine eigene Spezifikation damit befasst und weitere noch ausgearbeitet werden. Die Sicherheitsanforderungen in einer Grid-Umgebung lassen sich grob in drei Gruppen einteilen:

a. Integrationslösungen, in denen bereits existierende Lösungen genutzt werden müssen und Schnittstellen erweitert werden müssen, um eine erweiterbare Architektur zu schaffen.

b. Interoperabilitätslösungen, die in verschiedenen Virtuellen Organisationen eingesetzt werden, die unterschiedliche Sicherheitsmechanismen und Sicherheitsstrategien haben und trotzdem in der Lage sein sollen, zu interagieren.

c. Lösungen, um Vertrauensregeln in einer dynamischen Grid-Umgebung zu definieren, zu verwalten und durchzusetzen.

Das OGSA Sicherheitsmodell wird die folgenden Punkte, die ich hier nur nenne, adressieren: Authentifikation, Delegation, *Single Logon*, Lebenszeit und Erneuerung von Berechtigungsnachweisen, Authorisierung, Datenschutz, Vertraulichkeit, Integrität der Nachrichten, Austausch von Sicherheitspolicies, Sicheres Logging, Zusicherung eines Sicherheitslevels, Verwaltbarkeit, Überwindbarkeit von Firewalls, Absicherung der Infrastruktur des Grid.

4.4 Hosting Environments

OGSA beschreibt die Semantik von *Grid Services*, wie Sie erstellt und benannt werden, wie Ihre Lebenszeit bestimmt wird und so weiter, nicht jedoch ihre Implementierung.

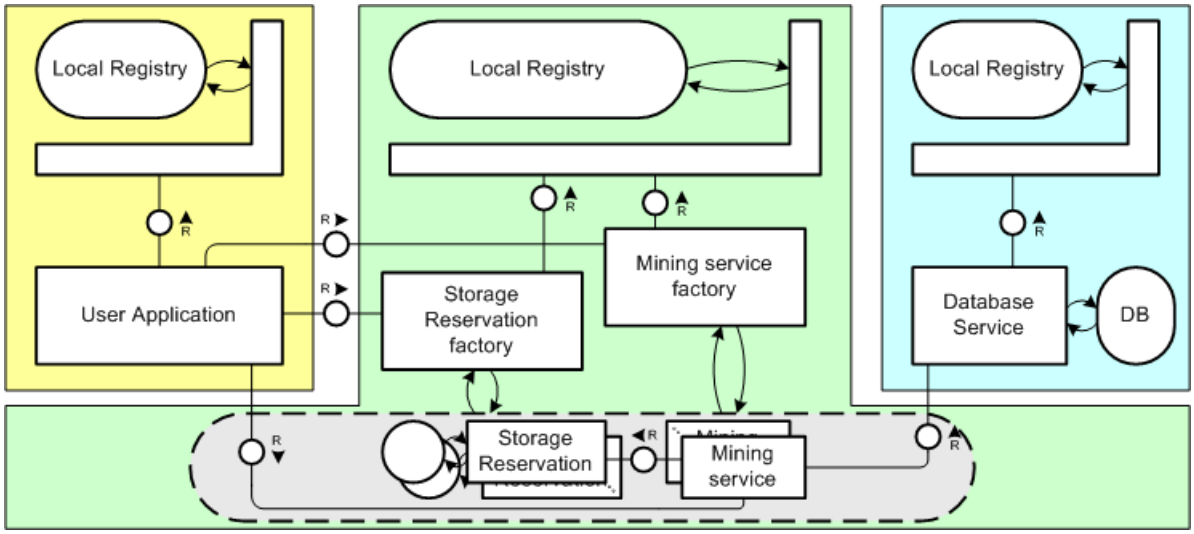
In der Praxis werden *Grid Services* in einem bestimmten *Hosting Environment* ausgeführt. Dieses *Hosting Environment* umfasst die Programmiersprache, das Programmiermodell, Entwicklungstools, Debugging Tools, und nicht zuletzt die Art und Weise, *wie* der Grid Service die Anforderung der Open Grid Services Architecture umsetzt.

Heutige wissenschaftliche Grid-Anwendungen arbeiten typischerweise auf Basis eines Unix-ähnlichen Betriebssystems, welches das *Hosting Environement* für Anwendungen darstellt, die in Sprachen wie C, C++, Fortran, oder Java geschrieben sind. Für OGSA-basierte Anwendungen werden *Hosting Environments* wie die Microsoft .NET-Umgebung für Sprachen wie C# und andere und Servlet- und EJB-Container für Java eine größere Rolle spielen.

4.5 Ausführungsbeispiel

Die folgende Abbildung soll die im vorhergegangenen Abschnitt beschriebenen Konzepte verdeutlichen. Es handelt sich dabei um ein FMC-Aufbaubild eines kleinen OGSA-basierten *Data-Mining-Service*.

Abbildung 4. Data Mining Service



Dabei beauftragt die Nutzerapplikation die Factory, einen *Mining Service* zu erzeugen und andere Factory einen *Storage Reservation Service* zum temporären Speichern der Daten zu erstellen. Diese Dienste müssen als vergängliche Dienste erstellt werden, da sie nutzerspezifische Daten enthalten.

Danach fragt die Instanz des *Mining Service* beim Datenbankdienst an, um Daten aus der Datenbank zu erhalten. Da dieser Service keine nutzerspezifischen Daten speichert, braucht keine Factory beauftragt zu werden. Der Mining Service speichert seine temporären Daten, indem er den Storage Reservation Service beauftragt.

Nehmen wir an, der *Mining Service* hat das *Datamining* nicht beendet, aber die Lebenszeit der erzeugten Instanz des Storage neigt sich dem Ende zu und der hostende Grid-Knoten lehnt eine Verlängerung des Services ab, weil er momentan ausgenutzt ist. In diesem Fall wird die User-Application benachrichtigt, da sie sich für diese Art von Nachrichten auf die Abonnentenliste dieses Dienstes hat setzen lassen.

Die Nutzerapplikation kann dann auf einem anderen Knoten eine zweite Instanz des *Storage Reservation* Dienstes erzeugen, die Daten des ersten replizieren und dem *Mining Service* mitteilen, fortan die neue Instanz als temporären Speicher zu nutzen. Nachdem der *Mining Service* seine Arbeit beendet hat, wird er von der User-Application zerstört.

Jeder Dienst verfügt über eine Lokale Registrierdatenbank, in der die GSH und GSR verwaltet werden.

5 Open Grid Services Infrastructure

Die *Open Grid Services Architecture* wie sie in *Pysiology of the Grid* beschrieben wird ist nur ein relativ unspezifischer Verbund von Anforderungen, Konzepten und Ideen, wie man ein offenes und erweiterbares Grid schaffen könnte.

OGSI, die *Open Grid Services Infrastructure* ist eine klare Spezifikation dessen, wie eine Implementierung der *Open Grid Services Architecture* gestaltet sein könnte. Diese Spezifikation legt fest, wie Exemplare von *Grid Services* benannt werden und wie man auf sie verweist, die Schnittstellen und zugehörigen Verhaltensweisen, die einen Grid Service ausmachen sowie die zusätzlichen und optionalen Schnittstellen, die für Factories und Gruppen von *Grid Services* gelten.

Die *Open Grid Services Infrastructure* behandelt nicht, wie Instanzen von Diensten erzeugt, zerstört und verwaltet werden, da das Abhängig von dem jeweiligen *Hosting Envrironment* ist.

Nach OGSI ist ein Exemplar eines *Grid Services*, welches dieser Spezifikation folgt, eine adressierbare und möglicherweise statusbehaftete Programminstanz, die ein oder mehrere Schnittstellen implementiert, die durch portTypes eines WSDL-Dokuments beschrieben werden.

Grid Service Factories können genutzt werden, um Exemplare von Grid Services zu erzeugen, die einen gegebenen Satz von portTypes implementieren und es existiert eine Möglichkeit, identifiziert zu werden und von den anderen Exemplaren im System zu unterschieden werden. Man kann jedes Exemplar als einen bestimmten Status, der ein typenspezifisches Verhalten zeigt, kennzeichnen. Die OGSA unterstützt die Möglichkeit, dass *Grid Services* sich selbst beschreiben, insbesondere die Schnittstellen die der jeweilige Dienst implementiert.

Damit Client-Applikationen *Grid Services* nutzen können, existieren *Grid Service Handle*s, die als permanente Zeiger auf eine Grid Service Instanz gesehen werden können, die jedoch zuvor in eine *Grid Service Reference* aufgelöst werden müssen. Eine *Grid Service Reference* enthält alle nötigen Informationen, um einen Dienst zu nutzen.

Die Open Grid Service Architecture bietet mit dem HandleResolver eine Möglichkeit, eine *Grid Service Handle* in eine *Grid Service Reference* aufzulösen.

In den folgenden Unterabschnitten werde ich nicht noch einmal gesondert auf die einzelnen Schnittstellen eingehen, da dies den Umfang dieser Ausarbeitung sprengen würde und stattdessen zwei Besonderheiten der OGSI-Spezifikation betrachten: Die WSDL-Erweiterungen und das Konzept der *Service Data*.

5.1 WSDL Erweiterungen

Die WSDL-Version 1.1, auf der OGSI aufbaut ist in Hinsicht auf die Erstellung von *Grid Services* beschränkt. Da OGSI jedoch auf *Web Services* aufbaut und WSDL zur Schnittstellenbeschreibung nutzt, fallen diese zwei Punkte besonders ins Gewicht.

1. Es gibt keine Vererbung zwischen portTypes (Schnittstellen)

2. und es fehlt eine Möglichkeit, zusätzliche Informationen in Form anderer Elemente in portTypes unterzubringen

Da WSDL 1.2 noch in Arbeit ist, wird OGSI auf einer Erweiterung von WSDL 1.1 aufsetzen. Diese Erweiterung betrifft nur das `wsdl:portType` Element, dem mit den Namespace-Prefix `gwsdl` und der Namespace-URI http://www.ggf.org/namespaces/2003/03/gridWSDLExtensions eine Erweiterung zugeordnet wird. Das `wsdl`-Element im Namespace mit dem Prefix `gwsdl` wird das gleiche Verhalten zeigen, wie es der Entwurf für WSDL 1.2 beschreibt.

Es kann also das optionale Attribut `extends` enthalten sowie eine beliebige Zahl von Elementen und Attributen anderer Namespaces.

5.2 Service Data

Ein bedeutendes Konzept für OGSI sind statusbehaftete Web Services, die jedoch das Problem mit sich bringen, dass man einen standardisierten Mechanismus braucht, um die Zustandsdaten des Dienstes abzufragen. Für diese Zustandsdaten wird der Begriff „Service Data“ genutzt.

Um eine vollständige Schnittstellenbeschreibung des *Web Services* zu liefern, ist es also auch notwendig, diese Elemente des Dienstes zu beschreiben, die von außerhalb zu beobachten sind. Diese Vorstellung kann man mit der Idee vergleichen, Schnittstellen in der Objektorientierung Attribute hinzuzufügen. Service Data kann also öffentlich zugänglich gemacht werden um, lesenden, schreibenden und modifizierenden Zugriff zu erlauben, man kann aber auch darauf verzichten und die Service Data kapseln und nur über die definierten Operationen der Schnittstelle Zugriff erlauben.

5.2.1 Die WSDL-Erweiterung für Service Data

Um Service Data in dem WSDL-Dokument als Schnittstellenbeschreibung unterzubringen, wird unterhalb des Elements portType (mit Namespace-Prefix gwsdl) das Element serviceData (mit Namespace-Prefix sd und der Namespace-URI http://www.ggf.org/namespaces/2003/02/serviceData) eingeführt.

Beispiel 5. portType mit serviceData

```
<gwsdl:portType name="portName">
  <wsdl:documentation />
  <wsdl:operation name="operationName" />
  <sd:serviceData name="sd1"
    type="xsd:String">
    <sd:staticServiceDataValues>
      <tns:sd1>initValue</tns:sd1>
    </sd:staticServiceDataValues>
</gwsdl:portType>
```

6 Implementationen

Zur Zeit gibt es zwei wichtige Implementationen, ich hier vorstellen werde. Zum ersten das Globus Toolkit Version 3, welches eine vollständige Umsetzung der OGSI-

Spezifikation darstellt und OGSI.NET, ein Projekt, welches eine Microsoft .NET-basierte OGSI-Implementation erschaffen will.

6.1 Globus Toolkit 3

Das Globus Toolkit 3 stellt eine vollständige Umsetzung der OGSI-Spezifikation dar und basiert zu großen Teilen auf dem Globus Toolkit 2.4. Zur Implementation der OGSI-Komponenten wurde das SOAP-Framework Axis der Apache Software Foundation genutzt.

In diesem Abschnitt möchte ich die Grundlagen der Installation und Konfiguration beschreiben, sowie eine kleine Beispielapplikation entwickeln. Diese Beispielapplikation wird einen Grid-Service anbieten, der grundlegende Trigonometrische Berechnungen erlaubt.

6.1.1 Installation

Die Installation des Globus Toolkit 3, oder GT3 erfordert ein Linux oder Unix mit den Standard-Entwicklungstools wie perl, make und gcc sowie das Java-Build-Tool ant [http://ant.apache.org/], welches in Version 1.5 oder höher installiert sein muss, so dass die Umgebungsvariable `ANT_HOME` gesetzt ist und die ausführbare Datei **ant** sich im `PATH` befindet.

Ant und GT3 erfordern Java 1.3 oder höher, doch da die Nutzung des Java SDK 1.3 die Installation weiterer Bibliotheken erfordert, empfiehlt es sich, auf Java 1.4 oder höher zurückzugreifen. Das Java SDK muss installiert sein, so dass die Umgebungsvariable `JAVA_HOME` gesetzt ist und die ausführbaren Dateien **java** und **javac** sich im `PATH` befinden.

Die Installation des relationen Datenbankmanagementsystems postgresql [http://www.postgresql.org] ist optional und wird hier, der Einfachheit halber übergangen.

Die Java-Bibliotheken Oro [http://jakarta.apache.org/oro/] und JUnit [http://www.junit.org] sollten auf dem System installiert sein und eine Kopie von `oro.jar` und `junit.jar` sollten im Verzeichnis `ANT_HOME/lib` abgelegt werden.

Die Installation

Die gesamte Installation sollte nicht als root durchgeführt werden. Ich habe dazu einen eigenen User globus angelegt.

1. Herunterladen der Source-Distribution von der Globus Toolkit Download Website [http://www-unix.globus.org/toolkit/download.html]. Man sollte auf die Source-Distribution zurückgreifen, da es bei der vorkompilierten Version bei einigen Linux-Distributionen zu Inkompatibilitäten mit der glibc kommen kann.

   ```
   globus@notebook $ wget http://www-unix.globus.org/\
     ftppub/gt3/3.0/3.0.1/gt3.0.1-source-installer.tar.gz
   ```

2. Entpacken der Distribution und wechseln in das Verzeichnis der entpackten Quellen

```
globus@notebook $ tar -zxvf gt3.0.1-source-installer.tar.gz
globus@notebook $ cd gt3.0.1-source-installer
```

3. Starten der Installation durch Aufrufen des Installationskripts. Der erste Parameter ist das Ziel der Installation.

   Bei Fehlern sollte man die Datei `install.log` durchsuchen. In der Regel zieht der erste Fehler keinen direkten Abbruch nach sich, sondern führt erst bei Folgefehlern zum Abbruch. Diesen ersten Fehler muss man identifizieren und beheben.

   ```
   globus@notebook $ ./install-gt3 /home/globus/gt3 | tee install.log
   ```

Damit ist die Installation abgeschlossen.

6.1.2 Nachinstallation

Zur Nachinstallation wechselt man in das Installationsverzeichnis und startet ant, um das initiale Setup des Programms vorzunehmen.

```
globus@notebook $ cd /home/globus/gt3
globus@notebook $ ant setup
```

Danach erzeugt man das Verzeichnis `JAVA_HOME/endorsed` und kopiert die Datei `endorsed/xerces.jar` dorthin, um den Standard-XML-Parser von Java 1.4 durch den mächtigeren Xerces-Parser zu ersetzen.

```
globus@notebook $ sudo mkdir /opt/sun-sdk-1.4.1.03/endorsed
globus@notebook $ sudo cp endorsed/xerces.jar /opt/sun-jdk-1.4.1.03/endorsed
```

Bevor weitere Schritte ausgeführt werden, ist sicher zu stellen, dass alle Umgebungsvariablen korrekt gesetzt sind. Insbesondere die Variable `GLOBUS_LOCATION` muss auf das Installationsverzeichnis weisen. Um die Variablen nicht bei jedem login neu zu setzen, empfiehlt es sich, sie in die `.bashrc` aufzunehmen.

```
export ANT_HOME=/usr/share/ant
export JAVA_HOME=/opt/sun-sdk-1.4.1.03
export PATH=$JAVA_HOME/bin:$PATH
export GLOBUS_LOCATION=/home/globus/gt3
```

Dann kann aus dem Installationsverzeichnis aus die *Globus Security Infrastructure* aufgesetzt werden, die über einen Konsolen-Dialog gesteuert wird.

```
globus@notebook $ ./setup/globus/setup-gsi
```

Schlussendlich kann der OGSI-Container gestartet werden. Durch den letzten Parameter kann der Port bestimmt werden, auf dem der Container laufen wird.

```
globus@notebook $ source etc/globus-user-env.sh
globus@notebook $ bin/globus-start-container -p 8080
```

6.1.3 Beispielprogramm

Das folgende Beispielprogramm soll es erlauben, Winkelberechnungen wie Sinus, Consinus und Tangens auszuführen. Dazu muss zuerst die Schnittstelle des Dienstes beschrieben werden, wobei man zum einen direkt eine WSDL-Datei als Schnittstellenbeschreibung erstellen kann oder ein Interface in Java schreibt und sich daraus eine WSDL-Datei erzeugen lässt.

Ich werde aus Gründen der Einfachheit den zweiten Weg wählen, auch wenn es bei komplexen Datentypen zu Mapping-Schwierigkeiten zwischen WSDL und Java kommen kann, die in dieser Betrachtung aber keine Rolle spielen.

Beispiel 6. `Trigonometry.java`

Dieses Java-Interface und alle weiteren Dateien auf denen gearbeitet wird liegt in einem eigenen Verzeichnis `example`.

```
package net.trieloff.ogsa;

public interface Trigonometry {
  public float sinus(float a);
  public float cosinus(float a);
  public float tangens(float a);
}
```

Aus dieser Schnittstellenbeschreibung in Java wird eine Schnittstellenbeschreibung in WSDL erzeugt. Dazu wird die Source-Datei erst kompiliert und dann mit einem Java-Programm in WSDL umgewandelt.

```
globus@notebook $ javac net/trieloff/ogsa/Trigonometry.java
globus@notebook $ java org.apache.axis.wsdl.Java2WSDL \
  -P TrigonometryPortType \
  -S TrigonometryService \
  -l http://localhost:8080/ogsa/services/core/first/TrigonometryService \
  -n http://ogsa.trieloff.net/TrigonometryService net.trieloff.ogsa.Trigonometry
```

Die resultierende Datei `TrigonometryService.wsdl` wird dann mit einem weiteren Java-Programm `dekoriert`, bzw. um OGSI-Erweiterungen ergänzt.

```
globus@notebook $ java org.globus.ogsa.tools.wsdl.DecorateWSDL \
  ~/gt3/schema/ogsi/ogsi_bindings.wsdl \
  TrigonometryService.wsdl
```

Bevor das beschriebene Interface implementiert werden kann, müssen zuerst die Server- und Client-seitigen Stubs erzeugt werden, gegen die programmiert wird. Dabei ist ein weiteres Java-Programm behilflich.

```
globus@notebook $ java org.globus.ogsa.tools.wsdl.GSDL2Java \
  TrigonometryService.wsdl
```

Dieser Befehl erzeugt ein neues Verzeichnis `org/gridforum/ogsi` und `net/trieloff/ogsa/TrigonometryService`, in denen die erzeugten Klassen

liegen. Damit sind alle Anforderungen erfüllt, um die Implementation des Services zu schreiben.

Beispiel 7. `TrigonometryImpl.java`

```
package net.trieloff.ogsa;

import org.globus.ogsa.impl.ogsi.GridServiceImpl;
import net.trieloff.ogsa.TrigonometryService.TrigonometryPortType;
import java.rmi.RemoteException;

public class TrigonometryImpl
    extends GridServiceImpl
    implements TrigonometryPortType {

  public TrigonometryImpl() {
    super("Simple Trigonometric Calulation Service");
  }

  public float sinus(float a) throws RemoteException {
    return (float) Math.sin(a);
  }

  public float cosinus(float a) throws RemoteException {
    return (float) Math.cos(a);
  }

  public float tangens(float a) throws RemoteException {
    return (float) Math.tan(a);
  }
}
```

Diese Datei importiert `GridServiceImpl` aus der OGSI-Kern-API, von dem `TrigonometryImpl` erbt und das Interface `TrigonometryPortType`, welches aus der GSWDL-Datei generiert wurde.

Um den Dienst der eigentlichen Benutzung zuzuführen, muss zuerst noch ein Deployment-Deskriptor erstellt werden, der dem Container mitteilt, wie der *Grid Service* einzusetzen ist. Die zugehörige Datei muss im gleichen Verzeichnis liegen wie die Java-Sourcen die zuvor erstellt wurden.

Beispiel 8. `Trigonometry.wsdd`

```
<?xml version="1.0"?>
<deployment name="defaultServerConfig"
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service
      name="example/TrigonometryService"
      provider="java:RPC">
    <parameter name="allowedMethods" value="*"/>
    <parameter
        name="className"
        value="net.trieloff.ogsa.TrigonometryImpl"/>
  </service>
</deployment>
```

Aus dem Wert des Parameters `name` des Elements `service` und der Basis-URL des Container ergibt sich die URI des Dienstes. Für den GT3 Container ist die resultierende URI http://localhost:8080/ogsa/services/example/Trigonometry.

Im nächsten Schritt werden schrittweise zuerst die Stubs und dann die Implementation kompiliert und in eine JAR-Datei komprimiert.

```
globus@notebook $ javac -sourcepath ./ net/trieloff/ogsa/TrigonometryService/*.java
globus@notebook $ javac -sourcepath ./ net/trieloff/ogsa/*.java
globus@notebook $ jar cvf Trigonometry-stub.jar \
  net/trieloff/ogsa/TrigonometryService/*.class
globus@notebook $ jar cvf Trigonometry.jar net/trieloff/ogsa/*.class
```

Aus den Java-Archiven, dem Deployment-Deskriptor und der WSDL-Datei wird jetzt ein GAR- oder Grid Archive erzeugt. Dieses kann dann im Container *deployt* werden. Dazu wird ein Verzeichnis für die zu packenden Dateien angelegt, die Dateien werden in dieses Verzeichnis kopiert und zum Schluss gepackt.

```
globus@notebook $ mkdir gar
globus@notebook $ cp *.jar gar/
globus@notebook $ cp net/trieloff/ogsa/Trigonometry.wsdd gar/server-deploy.wsdd
globus@notebook $ mkdir gar/schema
globus@notebook $ mkdir gar/schema/example
globus@notebook $ cp TrigonometryService.wsdl gar/schema/example/
globus@notebook $ jar cvf Trigonometry.gar -C gar/ ./
```

Von dem Installationsverzeichnis des GT3 aus kann das GAR-Paket *deployt* werden. Dazu existiert ein spezieller ant-Task, dessen letzter Parameter den Pfad zum GAR-Archiv weist.

```
globus@notebook $ ant deploy -Dgar.name=/home/globus/example/Trigonometry.gar
```

Wenn man jetzt noch sicherstellt, dass der Container läuft, so kann im nächsten Schritt ein einfacher Kommandozeilen-Client für diesen *Grid Service* geschrieben werden.

Beispiel 9. `TrigonometryClient.java`

```
package net.trieloff.ogsa;

import net.trieloff.ogsa.TrigonometryService.TrigonometryServiceLocator;
import net.trieloff.ogsa.TrigonometryService.TrigonometryPortType;
import java.net.URL;

public class TrigonometryClient {
  public static void main(String[] args) {
    try {
      float a = Float.parseFloat(args[1]);
      URL GSH = new URL(args[0]);
      TrigonometryServiceLocator trigonometryService =
          new TrigonometryServiceLocator();
      TrigonometryPortType trigonometry =
          trigonometryService.getTrigonometryService(GSH);

      float sin = trigonometry.sinus(a);
      float cos = trigonometry.cosinus(a);
      float tan = trigonometry.tangens(a);
      System.out.println("sin("+a+") = "+ sin);
      System.out.println("cos("+a+") = "+ sin);
      System.out.println("tan("+a+") = "+ sin);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Dieses Programm liest zwei Kommandozeilenparameter ein. Der erste wird als URI des Services oder *Grid Service Handle* interpretiert, und genutzt, um eine Referenz auf den Dienst an sich zu erlangen. Der zweite Parameter wird diesem Dienst als Parameter für die Operationen `sinus`, `cosinus` und `tangens` übergeben.

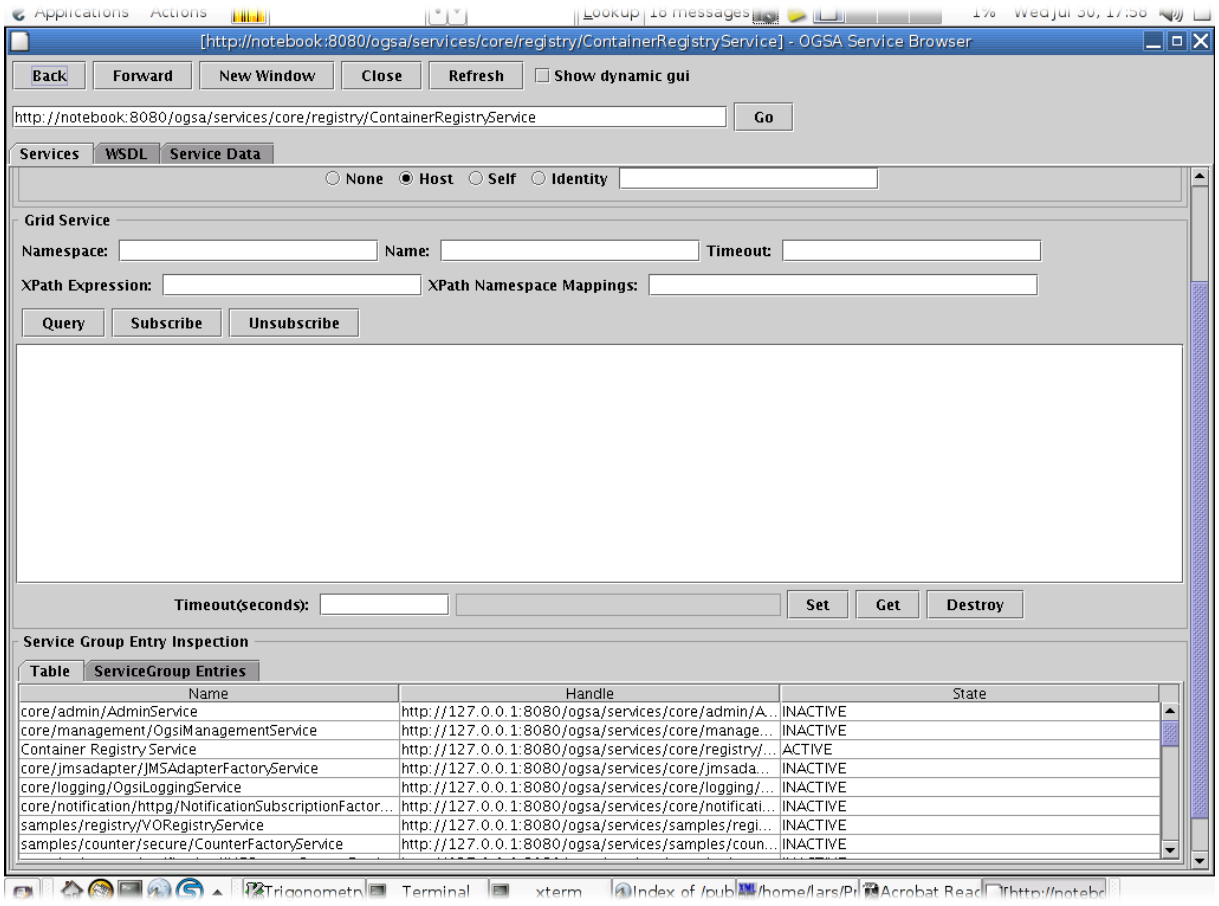
Dieser Client kann anschließend kompiliert und aufgerufen werden, mit der vorhin bestimmten Service URL.

```
globus@notebook $ javac -sourcepath ./ net/trieloff/ogsa/*.java
globus@notebook $ java net.trieloff.ogsa.TrigonometryClient \
  http://localhost:8080/ogsa/services/example/TrigonometryService 282
sin(282.0) = -0.67674977
cos(282.0) = 0.73621315
tan(282.0) = -0.91923076
```

6.1.4 Weitere Komponenten

Das Globus Toolkit 3 enthält eine Reihe weiterer Komponenten, wobei der *OGSI Service Browser*, eine Java-Swing-Applikation eine der offensichtlich naheliegendesten ist. Der *OGSI Service Browser* kann genutzt werden, um sich die Liste der registrierten Dienste eine *Registry* anzeigen zu lassen.

Abbildung 5. Screenshot des Java OGSI Service Browser



6.2 OGSI.NET

Die zweite wichtige Implementation der *Open Grid Services Architecture* nennt sich OGSI.NET und ist ein Projekt der Grid Computing Group der University of Virginia, unterstützt durch Microsoft Research, National Partnership for Advanced Computer Infrastructure und der National Science Foundation. Das Ziel des Projektes ist es, ein auf dem Microsoft .NET-Framework basierendes Pendant zur Java-basierten OGSI-Implementation des Globus-Tookits zu schaffen.

Zum Zeitpunkt dieses Schreibens liegt ein *Tech Preview* 1.1 vor, welches sowohl Server-Komponenten enthält als auch einen graphischen Service Browser wie das Globus Toolkit 3.

6.2.1 Installation

Die Installationspakete für den Server und den Client findet man unter http://www.cs.virginia.edu/~gsw2c/ogsi.net.html.

Installationsvoraussetzungen sind das .NET-Framework 1.1 und die Internet Information Services. Wenn die Internet Information Services nach dem .NET-Framework installiert worden sind, so muss das .NET-Framework komplett deinstalliert und erneut

installiert werden, um eine korrekte Integration mit den Internet Information Services zu gewährleisten.

Zweite Voraussetzung ist das Web Services Enhancements Service Pack 1.0 [http://msdn.microsoft.com/webservices/building/wse/default.aspx]. Eine optionale Voraussetzung ist das Visual Studio 2003, welches man benötigt, um eigene Dienste zu entwickeln.

Die Installation

1. Herunterladen und Entpacken der Server- und Client-Pakete. Beide Pakete haben eine `setup.exe`, die graphisch durch die Installation führt.

2. OGSI.NET erfordert die Installation eines ISAPI Filters, die über die Systemsteuerung durchgeführt werden muss.

   2.a. Man wählt Start->Systemsteuerung->Verwaltung->Internetinformationsdienste-Manager

   2.b. Auswählen des Symbols *$RECHNERNAME* (lokaler Computer), wobei `$RECHNERNAME der Name des aktuellen Rechners ist.`

   2.c. Auf der rechten Seite rechtsklickt man auf das Piktogramm Websites und wählt aus dem Kontext-Menü Eigenschaften in dem sich öffenden Dialog wählt man den Reiter ISAPI-Filter und klickt dort auf Hinzufügen

   2.d. In dem Dialogfeld trägt man im Feld Filtername den Wert „OGSIdotNetIsapiUrlFilter“ ein, die entsprechende Ausführbare Datei findet man unterhalb des Installationsverzeichnisses der Server-Komponente als `OGSIdotNet\bin\OGSIdotNetIsapiUrlFilter.dll`.

   2.e. Durch Klicken von OK im Dialogfeld und OK im Eigenschaftsdialog bestätigt man das Hinzufügen.

3. Im nächsten Schritt werden spezielle HttpHandler für .wsdl- und .xsd-Dateien installiert. Dazu doppelklickt man auf das Symbol Standardwebsite dort klickt man mit der rechten Maustaste auf das virtuelle Verzeichnis „schema“ und wählt abermals Eigenschaften.

   Durch Drücken des Druckknopfes Konfiguration... öffnet sich der Dialog Anwendungskonfiguration. Der erste geöffnet Reiter ist Zuordnungen. Dort klickt man auf Hinzufügen.

   3.a. Die Ausführbare Datei ist `c:\WINDOWS\Microsoft.Net\Framework\v1.1.4322\aspnet_isapi.dll`

   3.b. Die Erweiterung ist „.wsdl“

3.c. Im Abschnitt Verben wählt man den Radiobutton Begrenzen auf und trägt im zugehörigen Textfeld ein „GET, POST, HEAD, DEBUG“

3.d. Das Auswahlkästchen Verifizieren, dass Datei existiert darf nicht ausgewählt sein.

3.e. Durch Klicken von OK werden die Änderungen übernommen

Diese Vorgehensweise wird mit den gleichen Werten wiederholt mit dem Unterschied, dass als Erweiterung „.xsd“ gewählt wird.

4. Der OGSI.NET *Hosting Container* kann neu gestartet werden, indem man den Rechner neu startet oder mit der rechten Maustaste auf das Symbol *$RECHNERNAME* (lokaler Computer) klickt und Alle Tasks->IIS neu starten wählt.

Aus dem Startmenü kann der OGSI Browser gestartet werden und die lokale Installation getestet werden.

6.2.2 Kompilieren der Dienste

Unterhalb des Installationsverzeichnisses gibt es das Verzeichnis `Src`, in dem die Quellen des Containers liegen und Projektdateien für Microsoft Visual Studio zu finden sind. Diese Projektdateien sind nicht vollständig, es ist nicht möglich, das Projekt zu kompilieren.

Um dies zu bewerkstelligen, müssen die Teilprojekte einzeln erstellt werden und die dabei erzeugen DLL-Dateien der globalen Projektkonfiguration hinzugefügt werden.

Erst wenn gewährleistet ist, dass der Container kompiliert werden kann, ist es auch möglich, eigene Dienste zu entwickeln.

6.2.3 Beispielprogramm

Die Erzeugung eines Beispielservices ist sehr viel einfacher als beim Globus Toolkit 3.0 aufgrund der Aspektorientierten Features der Programmiersprache C#, die es möglich macht, Schnittstellenbeschreibung und Implementation in einer Datei zu vereinen.

Um den Trigonometrie-Dienst, den ich im vorhergegangenen Abschnitt eingeführt habe unter .NET implementieren zu können habe ich unterhalb des Src-Verzeichnisses das Verzeichnis `Services\Samples\Trigonometry` angelegt und dort die Datei `TrigonometryService.cs` bearbeitet.

Beispiel 10. `TrigonometryService.cs`

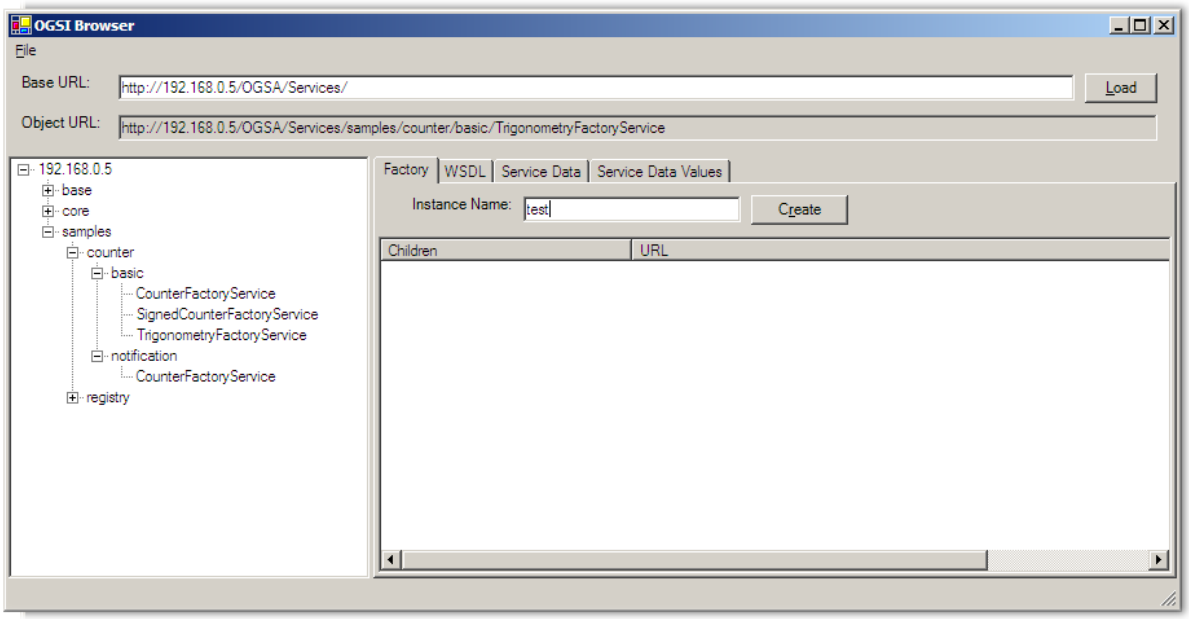
```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Serialization;
using UVa.Grid.OGSIGridServiceLib;
using UVa.Grid.WsdlDocumentLib;

namespace net.trieloff.ogsa
{

  [WebService(Name="CounterService",
    Namespace="http://ogsa.trieloff.net/trigonometryService")]
  [WebServiceBinding(Name="TrigonometrySOAPBinding",
    Namespace="http://ogsa.trieloff.net/trigonometryService/bindings")]
  [OGSIPortType(typeof(GridServicePortType))]
  public class TrigonometryService : GridServiceSkeleton
  {

    public TrigonometryService()
    {
    }

    [WebMethod()]
    [SoapDocumentMethod("http://ogsa.trieloff.net/trigonometryService#sinus",
      Binding="TrigonometrySOAPBinding")]
    [return: XmlElement("returnValue")]
    public float sinus(float a)
    {
      return (float) Math.Sin(a);
    }

    [WebMethod()]
    [SoapDocumentMethod("http://ogsa.trieloff.net/trigonometryService#cosinus",
      Binding="TrigonometrySOAPBinding")]
    [return: XmlElement("returnValue")]
    public float cosinus(float a)
    {
      return (float) Math.Cos(a);
    }

    [WebMethod()]
    [SoapDocumentMethod("http://ogsa.trieloff.net/trigonometryService#tangens",
      Binding="TrigonometrySOAPBinding")]
        [return: XmlElement("returnValue")]
    public float tangens(float a)
    {
      return (float) Math.Tan(a);
    }
  }
}
```

Durch die Aspektorientierten Konzepte in C# ist es möglich, die gesamte Klasse zum *Grid Service* zu erklären (mit entsprechendem Namespace) und jede einzelne Methode für das Grid zugänglich zu machen.

Diese Klasse wird dem OGSI.NET-Container-Projekt hinzugefügt und das Projekt wird neu erstellt. Die neu erstellte Datei `Services\Samples\bin\Debug\OGSISampleServices.dll` wird in das Verzeichnis `bin` unterhalb des Installationsverzeichnisses kopiert.

Nach einem Neustart der Internet Informationsdienste kann der Dienst mit dem Service Browser inspiziert werden.

Abbildung 6. TrigonometryService im .NET-OGSI Service Browser



7 Ausblick und Schlussbetrachtung

Die *Open Grid Services Architecture* ist ein sehr vielversprechendes Konzept, welches in Zukunft sicher weitere Verbreitung finden wird. Besonders hervorzuheben ist der Ansatz, einen offenen und herstellerunabhängigen Standard für das Grid-Computing zu schaffen und mit den in der *Open Grid Services Infrastructure* definierten Schnittstellen und Semantiken ein einheitliches Austauschformat zu nutzen.

Die aktuellen Implementierungen sind nach meinen Erfahrungen noch nicht bereit für den produktiven Einsatz, was bei OGSI.NET insbesondere an mangelnder Dokumentation und einem fehlenden Deployment-Mechanismus liegt und beim Globus Toolkit 3 an der fehlenden Integration der verschiedenen Tools.

Jedoch werden diese Probleme mit der fortschreitenden Entwicklung der Implementationen in absehbarer Zeit beseitigt werden und damit den Voraussetzungen für eine weitere Verbreitung und Intensivierung der Nutzung des Grid-Computing Genüge tun.

Glossar

Glossar

Grid Computing	Als Grid bezeichnet man ein massiv verteiltes System, welches das Anbieten, Auffinden und die Nutzung von Ressourcen über verschiedene administrative Domänen hinweg ermöglicht. Dabei werden unter anderem Faktoren wie Verfügbarkeit, Authentifizierung, Kosten-Abrechnung und *Quality-of-Service* als Nutzer-Anforderungen berücksichtigt. Somit können Ressourcen von weitverteilten, völlig verschiedenen Organisationen koordiniert genutzt werden. Als Analogie werden hier die Eigenschaften des Stromnetzes (*Power Grid*) gesehen - die jeweilige Ressource ist im Idealfall unabhängig von der Quelle gleichbleibend verfügbar.
Virtuelle Organisation	Nach [anatomy2002] stellt eine virtuelle Organisation das flexible, sichere und koordinierte Bereitstellen und Nutzen von Ressourcen, an dem sich ändernde Gruppen von Individuen und Organisationen beteiligt sind, und bei dem sich die zu nutzenden Ressourcen ebenfalls dynamisch ändern.
Web Services	Web Services ist der Sammelbegriff für die Schnittstellen, die es ermöglichen verteilte Applikation auf Basis von Web Standards wie XML und HTTP zu entwickeln.

Abkürzungsverzeichnis

EJB	*Enterprise Java Beans*
FMC	*Fundamental Modeling Concepts*
GRAM	*Grid Ressource Allocation Manager*
GridFTP	*Grid File Transfer Protocol*
GSH	*Grid Service Handle*
GSI	*Grid Security Infrastructure*
GSR	*Grid Service Reference*
GT3	*Globus Toolkit Version 3*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Potocol* via *Transport Security Layer*
ISAPI	*Internet Server Application Program Interface* ist eine Schnittstelle der Internetinformationsdienste, die es erlaubt, dynamische Web-Applikationen zu entwickeln
MDS	*Metacomputing Directory Services*
MIME	*Multipurpose Internet Mail Extensions*

OGSA	*Open Grid Services Architecture*
OGSI	*Open Grid Services Infrastructure*
SOAP	*Simple Object Access Protocol*
SSH	*Secure Shell*
UDDI	*Universal Description Discovery and Integration*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
VPN	*Virtual Private Network*
WSDL	*Web Service Description Language*
WSIL	*Web Service Inspection Language*
XML	*Extensible Markup Language*
XML-RPC	*XML-based Remote Procedure Call*

Bibliographie

Artikel

[gwdc2003] Ian Foster, D. Gannon und H. Kishimoto. *Open Grid Services Architecture Use Cases*. 5. Juni 2003.

[gwdr2003a] Ian Foster und D. Gannon. *The Open Grid Services Architecture Platform*. 16. Feburar 2003.

[sandholm2002] Thomas Sandholm, Steve Tuecke, Jarek Gawor und Rob Seed. *Java OGSI Hosting Environment Design*. A Portable Grid Service Container Framework.

[gwdr2003] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling und P. Vanderbilt. *Open Grid Services Infrastructure (OGSI)*. 5. April 2003.

[gwdi2003] Frank Siebenlist, Von Welch, Steven Tuecke, Ian Foster, Nataraj Nagaratnam, Philippe Janson, John Dayka und Anthony Nadalin. *OGSA Security Roadmap*. Global Grid Forum Specification Roadmap towards a Secure OGSA. July 2003.

[gwdi2002] Nataraj Nagaratnam, Philippe Janson, John Dayka, Anthony Nadalin, Frank Siebenlist, Von Welch, Steven Tuecke und Ian Foster. *Security Architecture for Open Grid Services*. July 2002.

[seed2003] Rob Seed und Thomas Sandholm. *A Note on Globus Toolkit 3 and J2EE*. 1. August 2003.

[anatomy2002] Ian Foster, Carl Kesselman und Steve Tuecke. *The Anatomy of the Grid*. Enabling Scalable Virtual Organizations.

[physiology2002] Ian Foster, Carl Kesselman, Jeffrey M. Nick und Steve Tuecke. *The Physiology of the Grid*. An Open Grid Services Architecture for Distributed Systems Integration.

[soap2000] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelson, Henrik Frystyk Nielsen, Satish Thatte und Dave Winer. *Simple Object Access Protocol (SOAP) 1.1 [http://www.w3.org/TR/SOAP/]*. 8. Mai 2000.

[wsil2001] Keith Ballinger, Peter Brittenham, Ashok Malhotra, William A. Nagy und Stefan Pharies. *Web Services Inspection Language (WS-Inspection) 1.0 [http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html]*. November 2001.

[uddi2002] *UDDI Version 3.0 [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm]*. UDDI Spec Technical Committee Specification. 19. Julu 2002.

[wsdl2001] Erik Christensen, Francisco Curbera, Greg Meredith und Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1 [http://www.w3.org/TR/SOAP/]*. 15. März 2001.

[winer1999] Dave Winer. *XML-RPC Specification [http://www.xmlrpc.com/spec]*. 15. Juli 1999.

Bücher

[ibm2002] Luis Ferreira, Viktors Berstis, Jonathan Armstrong, Mike Kendzerski, Andreas Neukötter, Masanobu Takagi, Richard Bing-Wo, Adeeb Amir, Ryo Murakawa, Olegario Hernandez, James Magowan und Norbert Bieberstein. *Introduction to Grid Computing with Globus*. IBM Redbook.

Webseiten

[jareva2003] *Jareva [http://www.jareva.com/index.jhtml]*.

[oceano2003] *Oceano Project [http://www.research.ibm.com/oceanoproject]*.

[ibmdev2003] *IBM Developer Works [http://www-106.ibm.com/developerworks/grid/]*. Grid Computing.

Software

[vs2003] *Microsoft Visual Studio 2003 [http://msdn.microsoft.com/vstudio/]*.

[nop]*Microsoft Windows 2003 Server [http://www.microsoft.com/windowsserver2003/default.mspx]*.

[wse2003] *Microsoft Web Services Enhancement Service Pack 1.0 [http://msdn.microsoft.com/webservices/building/wse/default.aspx]*.

[java2003] *Java SDK 1.4.1.03 [http://java.sun.com/]*.

[globus2003] *Globus Toolkit 3.0.1 [http://www.globus.org/toolkit/]*.

[ogsinet2003] *OGSI.NET Tech Preview [http://www.cs.virginia.edu/~gsw2c/ogsi.net.html]*.

[gentoo2003] *Gentoo/Linux [http://www.gentoo.org/]*.

[ant2003] *Apache Ant 1.5.3 [http://ant.apache.org/]*.

[oro2003] *Jakarta Oro 2.0.7 [http://jakarta.apache.org/oro/]*.

Condor, Condor-G, Classad

Stefan Henze, Kai Köhne
stefan.henze@hpi.uni-potsdam.de, kai.koehne@hpi.uni-potsdam.de

Condor ist ein leistungsfähiges Workload-Management-System für rechenintensive Jobs, welches seit vielen Jahren erfolgreich benutzt wird. Einsetzbar für verschiedenste Aufgaben und Netzwerkgrößen, zeichnet es sich vor allem durch sein klares Design, Unterstützung für heterogene Umgebungen und leichte Anpassbarkeit an wechselnde Anforderungen aus. Desweiteren lässt es sich auf unterschiedliche Arten in eine Grid-Infrastruktur einbinden.

In dieser Seminararbeit soll nach einer kurzen Einführung in das Problemfeld des High Throughput Computing das Condor-System vorgestellt werden. Insbesondere wird dabei auf die Umsetzung der drei Features *Machtmaking*, *Checkpointing* und *Remote System Calls* bei Condor eingegangen. Im Anschluss daran werden die möglichen Spielarten, in denen Condor in die Grid Infrastruktur eingebunden werden kann, vorgestellt. Am Schluß folgen dann einige Beispiele für den Einsatz von Condor in Projekten verschiedener Größe.

Inhaltsverzeichnis

1 Einleitung **4-2**

2 Das Condor-System **4-3**
2.1 Geschichte 4-3
2.2 Rollen 4-4
2.3 Systemaufbau 4-6
2.3.1 Dämonen 4-7
2.3.2 Universen 4-8
2.4 Konzepte 4-10
2.4.1 Matchmaking 4-10
2.4.2 Checkpoiting und Migration 4-13
2.4.3 Remote System Calls 4-15

3 Condor und Grid Computing **4-16**
3.1 Flocking 4-17
3.1.1 Gateway Flocking 4-17
3.1.2 Direct Flocking 4-18
3.2 Condor-G 4-18
3.2.1 GlideIn 4-20
3.3 Globus Site Job Scheduler 4-21

4 Einsatz in der Praxis **4-21**
4.1 Einsatzfelder 4-22
4.2 Anwendungsbeispiele 4-22
4.2.1 Einsatz an der UW-Madison 4-23
4.2.2 Einsatz bei Oracle 4-23

5 Zusammenfassung **4-23**

1 Einleitung

Vor dem Siegeszug des Personal Computers waren Mainframe-Computer, die mehrere Benutzer über Terminals bedienen konnten, der Stand der Technik. Die Benutzer mussten sich die vorhandenen Ressourcen mit Anderen teilen, da die Hardware teuer war und möglichst gut ausgenutzt werden sollte. Der Schwerpunkt lag also auf der *effizienten Ausnutzung* der verfügbaren Rechenkapazitäten.

Mit der Zeit wurde die Hardware allerdings immer billiger, so dass sich die Idee des Personal Computers nach und nach durchsetzen konnte: Die Rechenkapazität wurde dezentralisiert und jedem einzelnen Nutzer exklusiv zur Verfügung gestellt. Möglich wurde dies durch die dramatische Abnahme der Kosten von Ressourcen in Verhältnis zu ihrer Leistung, so dass die *Antwortzeit* für den einzelnen Benutzer in den Vordergrund rückte. Allerdings bezahlte man dafür den Preis, dass die Ausnutzung der Ressourcen immer schlechter wurde.

Es gab und gibt allerdings eine wachsende Zahl von Benutzern, die weniger die Antwortzeit, sondern mehr der Durchsatz ihrer Programme interessiert. Die Frage hierbei ist nicht „Wie schnell kann ich Simulation X auf dieser Maschine ausführen?“ sondern „Wie oft kann ich die Simulation X im Laufe des nächsten Tages, Monats ... durchführen?“. Der Maßstab ist dabei nicht mehr die Anzahl von CPU Zyklen pro Sekunde, die ein Computer unter idealen Bedingungen erreicht, sondern die Anzahl der CPU Zyklen pro Tag/Monat/Jahr, die ein oder mehrere Computer unter alltäglichen Bedingungen liefern können. Eine weitere Besonderheit dieser Benutzergruppe ist, dass ihr Bedarf nach Ressourcen oft keine Obergrenze kennt: So werden etwa die Ergebnisse vieler Simulationen mit zunehmender Laufzeit immer besser, ohne dass ein Maximum für die gewünschte CPU-Zeit existiert.

Die auf diese Zielsetzung eingehende Disziplin wird – im Gegensatz zum *High Performance Computing (HPC) – High Throughput Computing (HTC)* genannt.[1] Der Schlüssel für die Erstellung leistungsfähiger HTC Systeme ist dabei das effiziente Verwalten und Ausnutzen aller verfügbaren Computerressourcen, die heutzutage vor allem aus einer großen Anzahl von einzelnen PC's oder Workstations besteht, deren theoretische Leistungsfähigkeit unter normalen Umständen kaum von ihren jeweiligen Benutzern ausgeschöpft wird.

Die Idee bei der Entwicklung des Condor-Systems war es, diese Lücke zu schließen: Als Scheduling-System für High-Throughput Computing ist Condor in der Lage,

[1] Die Unterscheidung zwischen HPC und HTC wurde laut [1] erstmals in einem Seminar des Goddard Flight Center der NASA im Juli 1996 und einen Monat später am CERN eingeführt.

die Rechenkapazität einzelner, lose gekoppelter Maschinen zu bündeln und über ein komfortabel zu bedienendes Interface vielen Benutzern zur Verfügung zu stellen.

Im Folgenden wird zunächst auf die Eigenschaften des Condor-Systems, seine Geschichte und sein Benutzermodell eingegangen, um dann den grundsätzlichen Systemaufbau zu beschreiben. Im Anschluss folgt eine genauere Untersuchung der unserer Meinung nach interessantesten Features von Condor: Das Matchmaking, Checkpointing und die sogenannten Remote System Calls.
Im Kapitel *Condor und Grid Computing* werden die verschiedenen Wege beschrieben, mit der sich Condor in ein größeres Grid integrieren lässt.
Darauf folgen einige Beispiele für den Einsatz von Condor in der Praxis. Abschließend werden dann die Ergebnisse dieser Seminararbeit noch einmal zusammengefasst.

2 Das Condor-System

Condor ist ein an der Universität von Wisconsin-Madison seit 1988 entwickeltes System zum Management von Jobs und Ressourcen in heterogenen Netzwerken. Erklärtes Ziel des gleichnamigen Projektes ist es, möglichst alle Rechenkapazitäten einer Organisation einfach kombinieren und nutzbar machen zu können. Die besonderen Stärken des Systems liegen daher in der Unterstützung für heterogene verteilte Ressourcen und für Anwender mit einem großen Bedarf an Rechenkapazität.

Condor bietet dabei alle Features eines Batch-Systems, wie etwa einen Job Queuing Mechanismus, eine flexible Unterstützung für verschiedene Scheduling-Politiken, Prioritäten und Möglichkeiten zum Beobachten und Managen von Ressourcen. Durch eine Architektur, die relativ wenig zentralisierte Infrastruktur voraussetzt, ist Condor ausreichend flexibel um Netzwerke verschiedenster Größenordnungen gut zu unterstützen, und gleichzeitig auch verhältnismäßig robust gegenüber Ausfällen.

Condor unterstützt eine große Anzahl von unterschiedlichen Plattformen[2], und bietet jeweils spezielle Umgebungen (so genannte Universen) für C, Java und andere Programme sowie für Software, die MPI oder PVM zur Kommunikation einsetzen. Das System stellt relativ geringe Anforderungen an die vorhandene Netzwerkinfrastruktur, lässt sich aber so konfigurieren, dass etwa ein verteiltes Dateisystem effizient genutzt wird.

Aufgaben, die gut mittels eines Condor-Systems erledigt werden können, sind insbesondere relativ lang laufende Programme, die keine User-Interaktion benötigen. Beispiele hierfür sind das Rendern von 3D-Szenen, automatische Regressionstests in der Softwareentwicklung oder auch Schaltkreissimulationen für neue elektronische Geräte.

2.1 Geschichte

Am Anfang des Condor-Projektes 1988 an der Universität von Wisconsin-Madison stand das Ziel, ein HTC-System zu entwickeln, welches seine Rechenzeit ausschließlich aus den ungenutzten CPU-Zyklen von Workstations (siehe [5]) beziehen sollte.

[2]Dazu zählen verschiedene UNIX-Derivate (HPUX, Solaris, IRIX, OSF/1, Linux.) und auch Windows NT/2000/XP

Die Grundlage dieses Projektes bildeten dabei die Arbeiten des Remote-Unix Projektes von Prof. D. Dewitt, R. Finkel und M. Solomon und der Gruppe um Prof. M. Livny, die sich mit Distributed Ressource Management beschäftigt hatten: Beide Gruppen implementierten schon 1986 gemeinsam eine erste Version des Condor Resource Management Systems (zu dieser Zeit noch Remote Unix (RU) System genannt) unter der Leitung von M. Litzkow.

Das Condor-System wurde im Rahmen des Projektes stetig weiterentwickelt und überarbeitet. So kamen etwa ein wesentliches Feature wie der Matchmaking-Algorithmus erst 1998 mit Version 6.0 dazu, nachdem vorhergehende Lösungen sich immer wieder als zu unflexibel erwiesen hatten. Zugleich wurde das Einsatzgebiet von Condor stetig erweitert. Anfänglich darauf ausgerichtet, via „opportunistischem Scheduling“[3] die freien CPU-Zyklen von gerade nicht benutzten Workstations zu stehlen (siehe [5]), rückte im Lauf der Jahre auch der Einsatz von Condor als Job Scheduling System für Cluster mehr und mehr in den Mittelpunkt. Außerdem wurde Condor immer wieder um Unterstützungen für neue Programmumgebungen (wie Java, PVM oder MPI) erweitert, teils wurden auch Protokolle und Standards mit Hilfe von Condor evaluiert und erprobt, was u.a. zu einer sehr guten Unterstützung des Globus-Frameworks führte. Allerdings ist bemerkenswert, dass sich die grundsätzliche Arbeitsweise des Condor-Kerns, wie sie in Abschnitt 2.3 auf Seite 4-6 aufgezeigt wird, seit 1988 trotz all dieser Erweiterungen nicht verändert hat. [4]

Am Condor Research Projekt sind heute an der Universität von Wisconsin-Madison mehr als 30 Fakultäten beteiligt, das engere Entwicklungsteam besteht aus 21 Mitarbeitern und zehn Studenten [1], die sich – neben der Weiterentwicklung von Condor an sich – mit einer ganzen Reihe von Forschungsprojekten beschäftigen, die im Umfeld von Condor entstanden sind: Dazu gehören die dynamische Modifikation vom Programmen, die Entwicklung von neuen I/O-Protokollen und einer fehlertoleranten Sprache für Systemintegration, die Entwicklung eines GLOBUS ASCII Helper Protocolls (GAHP) oder einem Scheduler für die Bewegung von größeren Datenmengen im Grid. Finanziell unterstützt wird das Projekt dabei unter anderem von der US-Regierung (DoD, DoE, NASA, NSF), AT&T, IBM, Intel und Microsoft.

2.2 Rollen

In Abbildung 1 auf Seite 4-5 sind die typischerweise an einem Condor-System beteiligten Benutzergruppen dargestellt. Die Gruppe der Systemadministratoren setzt das Condor-System auf und konfiguriert es, die Applikationsentwickler schreiben Programme, die mit Hilfe von Condor ausgeführt werden sollen, die Benutzer lassen diese Programme via Condor ausführen, und die Ressourcenbesitzer stellen schließlich die für die Ausführung benötigten Maschinen bereit. All diese Gruppen haben spezifische Anforderungen an das System, und Condor versucht, jeden dieser Ansprüche mehr oder weniger gerecht zu werden. Die Applikationsschreiber wollen beispielsweise möglichst wenig Einschränkungen, die beim Entwickeln von Programmen beachtet werden müssen, die Systemadministratoren wollen ein leicht zu konfigurierendes

[3] „opportunistic scheduling“ wird in [4] als die Fähigkeit eines Ressource Management Systems bezeichnet, Ressourcen effizient zu nutzen, ohne 100% Verfügbarkeit dieser Ressourcen vorauszusetzen

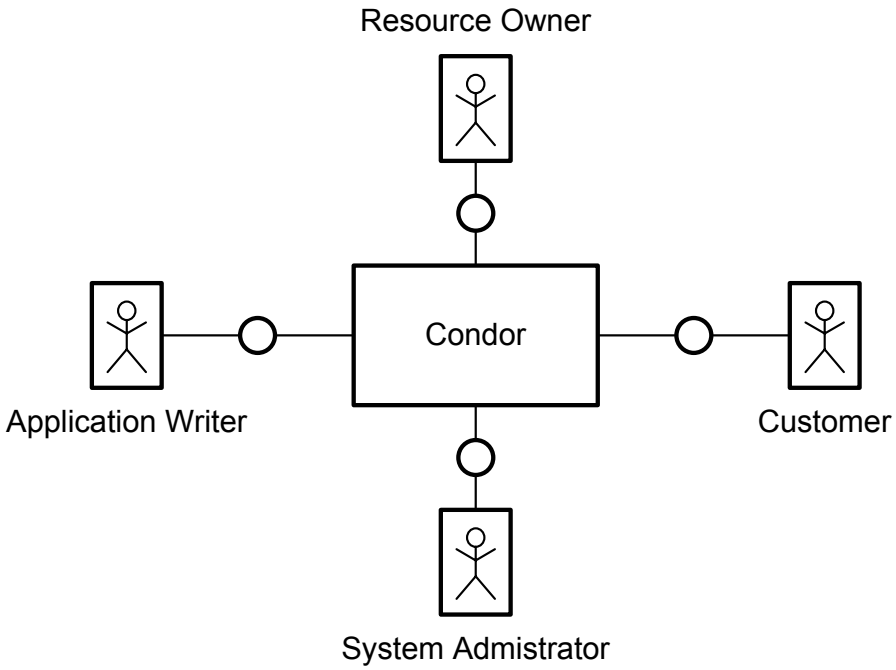


Abbildung 1: An einem Condor-System beteiligte Gruppen

System, welches sich flexibel an die spezifischen Voraussetzungen der Umgebung anpassen lässt, der Benutzer will komfortabel Condor Jobs starten, überwachen und beenden können, und die Ressourcenbesitzer wollen schließlich möglichst wenig Einschränkungen bei der Benutzung seines eigenen Rechners erfahren und individuell steuern können, wann welche Jobs auf seinem Rechner ausgeführt werden. Es ist allerdings auch offensichtlich, dass nicht alle Wünsche gleichermaßen beachtet werden können; sinnvollerweise müssen daher Schwerpunkte gesetzt und eine Art Hierarchie der Benutzerrollen aufgestellt werden.

Diese Hierarchie ergibt sich bei Condor aus dem ursprünglichen Einsatzziel des „Stehlens“ von CPU Zyklen ungenutzter Workstations: Ein solches System ist um so erfolgreicher, je mehr Workstations von ihren jeweiligen Besitzern (oder lokalen Administratoren) dem Condor-System zur Verfügung gestellt werden. Umgekehrt hat der Ressourcenbesitzer – falls er nicht zugleich auch Nutzer des Condor-Systems ist – erst einmal keinen Vorteil von seiner „freiwilligen“ Abgabe von CPU-Zeiten; vermutlich wird er seine Workstation sehr schnell wieder aus Condor-System entfernen, falls er durch Condor oder die durch Condor auf seiner Ressource ausgeführten Jobs in seiner Arbeit behindert wird. Deswegen war es bei der Entwicklung von Condor oberste Priorität, dem Ressourcenbesitzer wo immer möglich entgegenzukommen. So kann er etwa ausgefeilte Politiken aufstellen, wann sein System in welchem Maß überhaupt für Condor zur Verfügung steht oder welche Art von Jobs von welchen Benutzern auf seinem System ausgeführt werden können. Zur Entstehungszeit Condors (also in den späten Achzigern) war dieser Ansatz einzigartig, denn das damals dominante Modell einer zentralen Kontrolle für Batch-Systeme sah meist keine Möglichkeit für Ressourcen vor,

weitergehende Einschränkungen bezüglich ihrer Verfügbarkeit vorzunehmen. [4]

Am anderen Ende der Hierarchie steht der Benutzer: Er ist der eigentliche Nutznießer des Condor-Systems, und dementsprechend muß er bei einem Interessenskonflikt mit dem Ressourcenbesitzer immer zurückstehen. So wird z.B. der Job sofort unterbrochen, falls die Politik des Besitzers der benutzten Ressource dies vorsieht, selbst wenn dies bedeutet, dass alle bisherigen Ergebnisse des Jobs verworfen werden und der Job im schlimmsten Fall auf einem anderen Rechner von vorne beginnen muss. Allerdings besitzt Condor sehr ausgefeilte Mechanismen, um diesen Fall möglichst zu vermeiden, wie etwa das *Checkpointing*, welches im Abschnitt 2.4.2 auf Seite 4-13 vorgestellt wird.

2.3 Systemaufbau

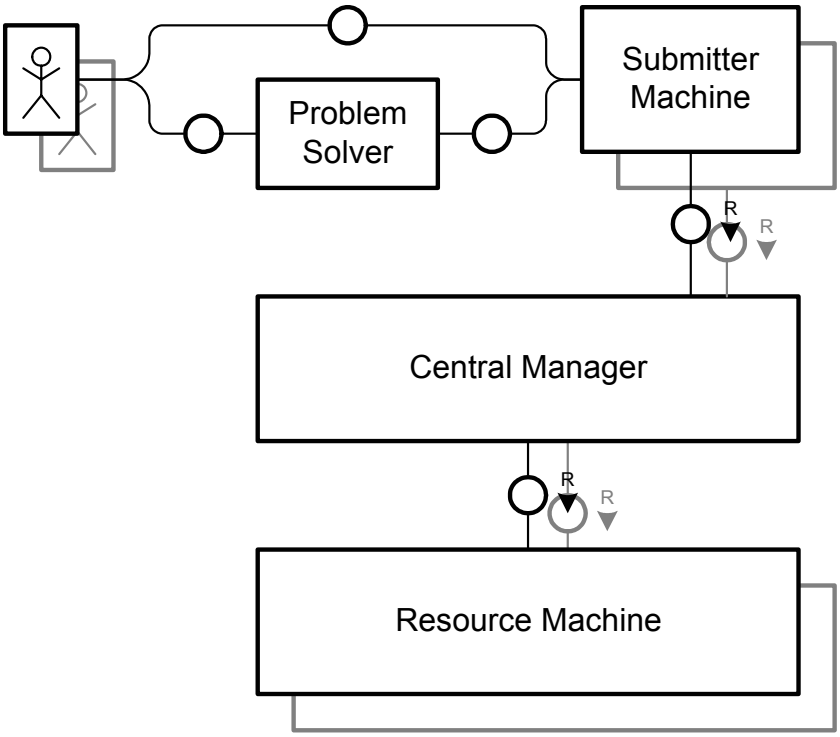


Abbildung 2: Grundsätzlicher Aufbau eines Condor-Systems

Der grundsätzliche Aufbau eines Condor-Systems in FMC-Notation (siehe [17]) ist in Abbildung 2 dargestellt: Entsprechend der im Abschnitt 2.2 auf Seite 4-4 eingeführten Rollen gibt es Rechner, die ihre nicht genutzte Rechenzeit dem Condor-System zur Verfügung stellen (*Resource Machine*), und andere, deren Benutzer sich diese zu nutzen machen wollen (*Submitter Machine*).[4] Ein Benutzer einer Submitter Machine kann nun Rechenzeit auf einer Resource Machine benutzen, indem er einen Job (das heißt ein ausführbares Programm, z.B. ein Shellscript, eine Kommandozeilenapplikation) in

[4]Ein Rechner kann natürlich auch beides in einem sein, Resource und Submitter Machine.

das Condor System einbringt. Für die Annahme dieser Jobs wird eine zentrale Instanz benötigt, der *Central Manager*: Er nimmt Jobs entgegen, und ordnet sie – je nach Priorität, benötigter Systemarchitektur, Größe – einer Resource Machine zu, die dann den Auftrag erhält, den Job des Benutzers auszuführen. Die Ergebnisse des Jobs werden nach Ende des Jobs wieder an die Submitter Machine zurück übertragen.

Will der Benutzer mehrere Jobs einbringen, und sind diese abhängig voneinander, kann er einen sogenannten *Problem Solver* benutzen. Dieser stellt auf der Submitter-Machine zusätzliche Komfortfunktionen bereit (wie z.B. das Einbringen von Job B in das System, sobald Job A fertig ist) und ist für den Rest des Condor-Systems völlig transparent, da er letztlich auch nur die verfügbaren Grundfunktionalitäten zum Einbringen und der Kontrolle von Jobs auf der Submitter Machine benutzt.

2.3.1 Dämonen

Ein Ziel bei der Entwicklung von Condor war ein möglichst modularer und skalierbarer Aufbau, weswegen verschiedene sich semantisch unterscheidende Aufgaben des Condor-Systems auch in eigenen Programmen realisiert wurden. Da Condor in der Unix-Welt zu Hause ist(es gibt mittlerweile auch eine Portierung für die Windows NT-Familie), bezeichnet man diese Programme, die auf einem Condor-Rechner immer im Hintergrund mitlaufen, als Dämonen.

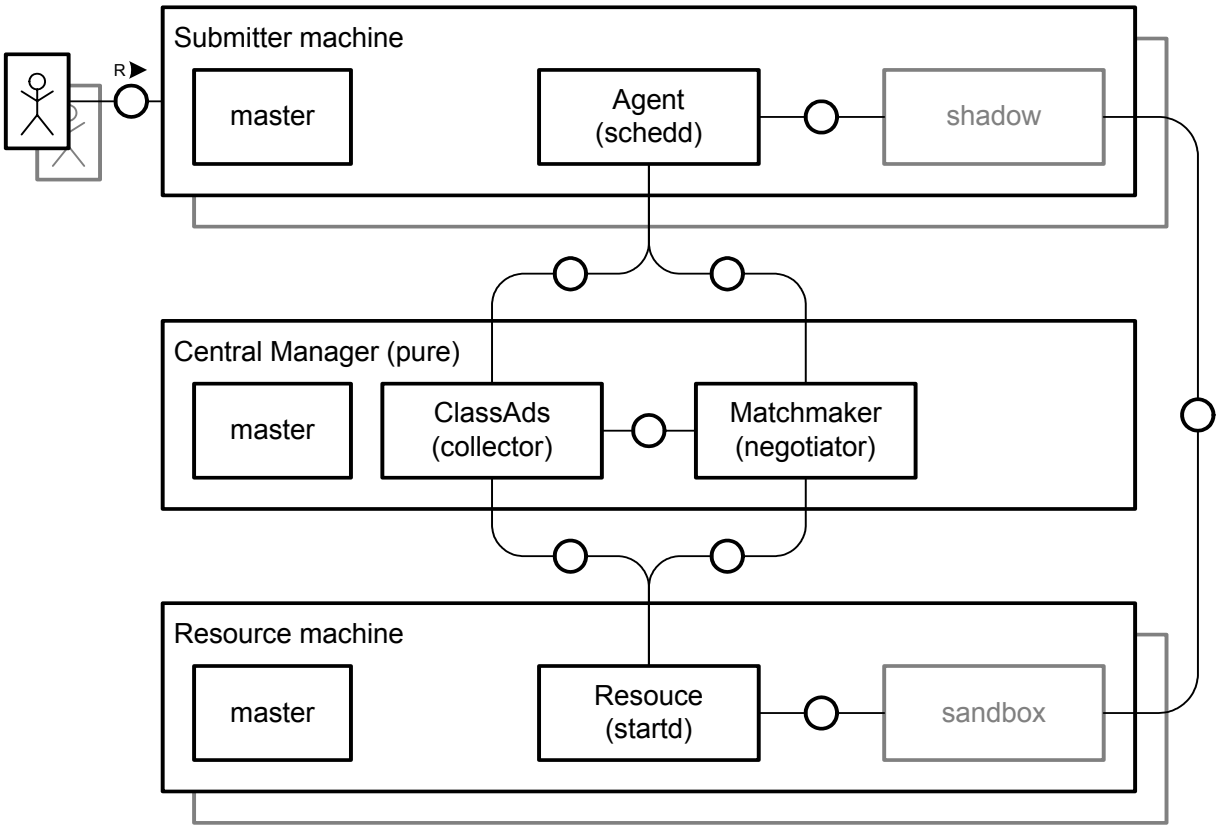


Abbildung 3: Die Dämonen eines Condor-Systems

In Abbildung 3 auf Seite 4-7 ist das Aufbaubild 2 auf Seite 4-6 verfeinert worden, um die verschiedenen auf den einzelnen Maschinen laufenden Dämonen darzustellen. In Klammern stehen die Namen, wie sie in der Liste der Prozesse auf der jeweiligen Maschine auftauchen. Da diese aber zum Teil etwas irreführend gewählt sind, verwenden wir im Folgenden die ohne Klammern angegebenen Bezeichnungen.

Alle Rechner im Condor System müssen einen Master-Prozess besitzen, dessen Aufgabe es ist, die anderen Dämonen zu starten und anhand von Konfigurationsdateien[5] zu initialisieren. Sollte ein Dämon einmal abstürzen, ist der der Master-Prozess auch dafür zuständig, diesen wieder neu zu starten. Die Benutzerkommandos, die sich mit der Administration des Condor-Systems beschäftigen (wie `condor_reconfig`, `condor_restart`, `condor_off`, `condor_on`, `condor_config_val ...`), kommunizieren immer mit diesem Dämonen.

Die Funktionalität des Central Manager wird durch zwei getrennte Dämonen realisiert: Der ClassAd-Server nimmt sogenannte ClassAds[6] von den Submitter- und Resource-Machines entgegen. Sowohl Resource- als auch Submitter-Machines sind dazu verpflichtet, regelmäßig Updates bezüglich der ClassAds an den Dämonen zu senden. Der Matchmaker ist nun dafür zuständig, die Liste aller Jobs mit der Liste aller verfügbaren Resourcen-Machines zu vergleichen und Paare zu finden, die zueinander passen. Ist ein solches gefunden, wird dies der entsprechenden Resource- und Submitter-Machine mitgeteilt.

Auf einer Submitter Machine startet der Master-Prozess zunächst nur einen Agenten, der dem System gegenüber als Vertreter der lokalen Benutzer auftritt. An ihn werden auszuführende Jobs der Benutzer geschickt, er erstellt für jeden Job eine gültige Beschreibung, übergibt diese den ClassAd-Server auf dem Central Manager und überwacht den Status des Jobs. Wurde eine Ressource für die entfernte Ausführung des Jobs gefunden, startet der Agent einen sogenannten Shadow-Prozess, der dafür zuständig ist, die Job-Datei und evtl. benötigte Ein- und Ausgabedateien der Resource Machine zugänglich zu machen und auch sogenannte Remote System Calls (siehe Abschnitt 2.4.3 auf Seite 4-15) serverseitig abzuwickeln.

Die Entsprechung zum Agent-Dämonen auf Seiten der Resource-Machine ist der Ressourcen-Prozess. Er vertritt eine Resource-Machine gegenüber dem Condor-System, erstellt aufgrund einer Konfigurationsdatei und Umweltdaten eine Ressourcen-Beschreibung, schickt diese regelmäßig an den ClassAd-Dämon auf Seiten des Central Managers und startet einen Sandbox-Prozess im Fall eines auszuführenden Jobs. Dieser kommuniziert dann direkt mit dem Shadow-Prozess auf der Seite der entsprechenden Submitter-Machine und stellt dem Job die benötigte Umgebung zur Verfügung.

2.3.2 Universen

Ein zentrales Ziel bei der Entwicklung von Condor war es, eine möglichst universelle Lösung für verschiedenste Arten von Anwendungen und Umgebungen zu entwickeln.

[5] Diese Konfigurationsdateien bestimmt u.a. auch, was für eine Rolle die Maschine im Condor-Netzwerk spielen soll.

[6] ClassAds werden im Abschnitt 2.4.1 auf Seite 4-10 beschrieben.

Dem Benutzer (bzw. dem Programmierer) soll hier ein Höchstmaß an Freiheit gegeben werden, wie seine Programme, die er auf Condor ausführen will, auszusehen haben; gleichzeitig gibt es viele sinnvolle Features (wie etwa das Checkpointing, welches in Abbschnitt 2.4.2 auf Seite 4-13 beschrieben wird), die auch Unterstützung von Seiten der Programme voraussetzen. Condor versucht mit der Verwendung von unterschiedlichen Universen für Programme allen Anforderungen gerecht zu werden: Universen sind dabei Laufzeitumgebungen, die beim Erstellen eines Jobs für ein bestimmtes Programm angegeben werden können.

Vanilla Das Vanilla-Universum von Condor ist die einfachste Laufzeitumgebung von Condor. Sie ist vor allem für Programme gedacht, die sich nicht für die Benutzung in Condor neu linken lassen (dazu zählen z.B. auch Shellscripte). Aufgrund dieser Einschränkung kann Condor für Vanilla-Jobs weder Checkpointing noch Remote System Calls bereitstellen (siehe Abschnitt 2.4.2 auf Seite 4-13 bzw. 2.4.3 auf Seite 4-15). Damit ein Vanilla-Job trotzdem Dateien lesen oder schreiben kann, muss entweder ein verteiltes Dateisystem vorhanden sein oder Condor angewiesen werden, bestimmte Dateien vor und nach Ausführung des Jobs zwischen Submitter- und Resource-Machine zu übertragen.[7]

Standard Das Standard-Universum ist das älteste von Condor unterstützte. Es bietet sowohl Remote System Calls als auch Checkpointing, erfordert aber ein Linken des Programmes mit der Condor Bibliothek via `condor_compile`. Deswegen ist ein Zugang zu den Objektdateien Voraussetzung.

Mit Remote System Calls und Checkpointing werden Condor Jobs zuverlässiger ausgeführt und können ohne Aufwand auf benötigte Ressourcen zugreifen; allerdings gibt es auch einige Einschränkungen, die bei der Programmierung von Programmen im Standard-Universe beachtet werden müssen. So sind weder Interprozesskommunikation noch mehrere Prozesse oder Threads erlaubt. Es dürfen auch keine Alarme, Timer, Memory Mapped Files oder Sleep-Anweisungen verwendet werden. Bei der Arbeit mit Dateien muss zudem berücksichtigt werden, dass Dateien nur entweder für Lesen oder für Schreiben geöffnet werden und Datei-Locks zwar erlaubt sind, aber nach einem (transparent ablaufenden) Checkpointing nicht mehr wiederhergestellt werden.

MPI Programme, die das MPICH (Message Passing Interface) benutzen, können im MPI-Universum mit Condor ausgeführt werden. Zur Zeit (in Version 6.4) unterstützt Condor dabei MPICH version ch_p4, einer kostenlosen MPI-Implementation der Argonne National Labs.

Ressourcen, die MPI-Jobs ausführen sollen, müssen dafür konfiguriert werden: Sie stehen dann für die Ausführung von Jobs in anderen Universen nicht mehr zur Verfügung. Ebenso müssen der Scheduler und die Submitter-Machine dedizierte Rechner sein, da MPI-Jobs prinzipiell nicht unterbrochen oder angehalten werden und damit nicht vom normalen Scheduler behandelt werden können.

[7] Unter UNIX ist ersteres der Standardfall, unter Windows letzteres. Über die genaue Konfiguration des sogenannten *Condor File Transfer Mechanism* gibt [2] Auskunft.

PVM Zur Ausführung von Programmen, die das PVM (Parallel Virtual Machine) Interface nutzen, gibt es das sogenannte PVM-Universum in Condor. Condor übernimmt dabei die Rolle eines Resource-Managers für den PVM-Dämonen, so dass PVM-Prozesse keine dedizierten Rechner benötigen und damit auch im normalen Condor-System ausgeführt werden können. Da das Condor-PVM binärkompatibel zum „normalen“ PVM (Version 3.4.0 - 3.4.2) ist, müssen die Programme nicht neu gelinkt werden, allerdings sollten sie (aufgrund der opportunistischen Natur des Condor-Systems) mit dem Ausfall von Knoten umgehen können, was etwa durch den Einsatz des Master-Worker Paradigmas geschehen kann. [2]

Globus Das Globus-Universum in Condor ist für Benutzer gedacht, die Globus-Jobs mit Hilfe des Condor-Systems zur Ausführung bringen wollen. Jeder Job wird dabei in ein Globus RSL übertragen und an das Globus-System übergeben. Eine genauere Beschreibung der Verwendung dieses Universums mit Condor-G findet sich unter Abschnitt 3.2 auf Seite 4-18.

Java Java Programme sind für den Einsatz als Condor-Jobs ideal. Sie sind nicht auf einen Maschinentyp beschränkt und stellen im Allgemeinen keine speziellen Anforderungen an die Umgebung (außer der, dass eine Java Virtual Machine vorhanden sein muss). Läuft ein Programm im Java-Universum, kümmert sich Condor darum, das JVM-Binary im Dateisystem zu finden und den Classpath zu setzen. Es erstellt auch einen Benchmark für jede Resource-Machine, die eine Java VM zur Verfügung stellen, so dass die Geschwindigkeit des Rechners beim Scheduling berücksichtigt werden kann.

Scheduler Das Scheduler-Universum nimmt insofern eine Sonderrolle ein, als dass Jobs in diesem Universum nicht via Condor auf andere Rechner verteilt, sondern sofort lokal und ohne Unterbrechung auf der Submitter-Machine ausgeführt werden.

Interessant ist dieses Universum eigentlich nur für die weiter oben schon beschriebenen „Problem Solver“, die – wie etwa DAGMan – Condor-Jobs des Benutzers auf der lokalen Rechner schedulen, bevor sie dem Central Manager übergeben werden.

2.4 Konzepte

2.4.1 Matchmaking

Die zentrale Aufgabe des Condor Systems ist die Zuordnung von Jobs zu Ressourcen, also das sogenannte *Scheduling* von Jobs. Es werden hierbei einige grundsätzlich andere Anforderungen an den Scheduling-Algorithmus gestellt als etwa bei der Prozessverwaltung eines Betriebssystems. Der Scheduler muss aufgrund der opportunistischen Natur eines Condor-Systems mit sehr viel mehr Unwägbarkeiten umgehen können als ein zentrales System, außerdem ist eine gute Skalierbarkeit und effiziente Ausnutzung der – verhältnismäßig langsamen – Netzwerkkommunikation erforderlich.

Des weiteren sollte der Scheduling-Algorithmus auch sehr viel flexibler sein, um die speziellen Anforderungen der Benutzer erfüllen zu können.

Bei einem typischen Condor-System hat man es mit einem relativ unzuverlässigen Netzwerk von Rechnern zu tun. Netzwerkverbindungen und Rechner können ausfallen, Benutzer von Rechnern können diesen plötzlich für sich beanspruchen und Jobs können andere Anforderungen an Ressourcen haben, als eigentlich angegeben. Anders als etwa bei einem lokalen Rechner hat es daher für den Schedulingalgorithmus wenig Sinn, sehr weit in die Zukunft zu planen. Erschwerend kommt hinzu, dass aufgrund der langsamen Netzwerkverbindungen der Scheduler auf potentiell veraltete Umgebungsdaten zugreift: So sind Ressourcen, die dem Condor System als frei bekannt sind, evtl. schon wieder in Benutzung, oder Benutzer haben Jobs schon abgebrochen, die im Scheduler noch bereitstehen. Um diese Probleme zu umgehen, unterteilt Condor das traditionelle Scheduling in *Planning* und *Claiming*: Das Planning übernimmt im Condor eine zentrale Komponente, während das Claiming, also die nochmalige Überprüfung der Voraussetzungen und die Übertragung des auszuführenden Programmes auf die Ressource, zwischen der Submitter- und der Resource-Machine ausgehandelt wird.

Zunächst muss natürlich dabei sichergestellt werden, dass der Job überhaupt zu dem vorhandenen Rechner passt (etwa in Bezug auf die Systemarchitektur); es macht keinen Sinn, Jobs Rechnern zuzuordnen, auf denen diese letztlich gar nicht laufen können. Daneben gibt es aber auch Wunschfaktoren, wie zum Beispiel dass ein bestimmtes Programm idealerweise auf einem Rechner mit mathematischem Co-Prozessor ausgeführt werden soll, aber nicht unbedingt muss. Dann gibt es wiederum Schedulingpolitiken, die sich auf den einzelnen Benutzer beziehen. So sollen beispielsweise Jobs eines Gelegenheitsbenutzers den Jobs eines Vielnutzers vorgezogen werden, oder die Jobs der Forschungsabteilung werden wichtiger eingestuft als die Jobs der Studenten.

Eine der Stärken von Condor liegt in der Flexibilität, mit all diesen Anforderungen umgehen zu können, ohne das System übermäßig zu belasten oder nicht mehr skalieren zu können. Erreicht wird dies durch den Mechanismus der ClassAds. Hier gibt jede beteiligte Entität[8] in einer sogenannten ClassAd bekannt, welche (Muss- und Kann-)Ansprüche sie an die potentiell ihr zuzuordnenden Entitäten hat, und welche Eigenschaften, die die Gegenseite interessieren könnte, sie selbst aufweist. Diese ClassAds werden dann an einem zentralen Ort im System (dem ClassAd-Dämonen) gesammelt, und regelmäßig untereinander durch den Matchmaker verglichen. Ergeben sich dabei Paare, wird das den beteiligten Entitäten mitgeteilt, so dass diese aushandeln können, ob sie wirklich zueinander passen, um dann – im Fall von Jobs und Ressourcen – den Job auf die Ressource zu übertragen und auszuführen. Der Matchmaker selber versteht dabei nichts von der Semantik der verglichenen Angaben zwischen Entitäten, er vergleicht letztlich nur Attribut-Wert-Paare, aus denen die ClassAds bestehen.

ClassAds Damit der Scheduler fundierte Entscheidungen über die Zuordnungen von Jobs zu Ressourcen treffen kann, müssen ihm die verschiedenen Anforderungen und

[8] Das Modell lässt sich nicht nur für die Zuordnung von Jobs und Ressourcen benutzen.

```
MyType = "Machine"
TargetType = "Job"
Memory = 1005
Arch = "Intel"
OpSys = "LINUX"
Disk = 56220192
FileSystemDomain = "haiti.cs.uni-potsdam.de"
Requirements = KeyboardIdle >= 15*60 && LoadAvg <= 0.3 &&
   Group != "mit"
Rank = (Owner == "henze" * 3) + (Group == "hpi" * 2)
```

Quelltext 1: Beispiel-ClassAd einer Ressource

```
MyType = "Job"
TargetType = "Machine"
Owner = "henze"
Group = "hpi"
DiskUsage = 37045
ImageSize = 28672
Requirements = (((OpSys == "IRIX65" && Arch == "SGI") || (
   OpSys == "LINUX" && Arch == "Intel"))) && (Disk >=
   DiskUsage) && ((Memory * 1024) >= ImageSize) && (
   FileSystemDomain == "haiti.cs.uni-potsdam.de")
Rank = Memory >= 64
```

Quelltext 2: Beispiel-ClassAd eines Jobs

Wünsche der Beteiligten bekannt gemacht werden: Dies geschieht über den flexiblen Mechanismus der ClassAds (Classified Advertisements). Der Begriff der ClassAds soll an Zeitungsanzeigen erinnern, bei der Anbieter und Angebotene ja auch ihre Wünsche und Vorstellungen beschreiben; allerdings müssen bei Condor beide Seite eine ClassAd veröffentlichen, und die Durchführung der Zuordnung liegt bei einer dritten Instanz.

Im Quelltext 1 findet sich einen vereinfachten ClassAd für eine Ressource, während Quelltext 2 eine ClassAd für einen Job enthält.

ClassAds bestehen aus einer Menge von Attributen; einige haben spezielle Bedeutungen, andere haben für das Condor-System keine spezielle Semantik. Jedes Attribut besteht aus einem Bezeichner und einem Ausdruck, der selbst wieder aus Konstanten, Referenzen auf andere Attribute, eingebauten Funktionen und Unterausdrücken mit Operatoren und Klammern bestehen kann. Zwei der wichtigsten für Condor mit Semantik belegten Attribute sind die in 1 und 2 beiden kursiv gesetzten Bezeichner *Requirements* und *Rank*: Bei der Überprüfung, ob ein Resourcen-ClassAd und ein Job-ClassAd zusammenpassen, wertet Condor zunächst einmal das Attribut *Requirements* aus. Ergibt es für beide ClassAds *true*, passen die beiden ClassAds prinzipiell zueinander. Gibt es nun noch eine andere mögliche Zuordnung eine der beiden ClassAds, wird mittels *Rank* ermittelt, welche Zuordnung letztlich getroffen wird. Je höher

der *Rank* für den jeweiligen Job/die jeweilige Resource, desto wahrscheinlicher werden zwei ClassAds einander zugeordnet.

Schedulingpolitiken Einer der Vorteile von Condor sind die vielfachen Möglichkeiten, mit denen alle Beteiligten Einflüsse auf das Scheduling von Jobs nehmen können. So können Benutzer des Systems über einen sogenannten Meta-Scheduler darüber entscheiden, welche Jobs wann dem restlichen System übergeben werden; zusätzlich können sie – wie auch die Ressourcenverwalter – das zentrale Scheduling via lokalen Schedulingpolitiken in Form der weiter oben beschriebenen ClassAds steuern. Neben diesen lokalen Schedulingpolitiken besitzt das Condor-System allerdings auch noch eine *globale Schedulingpolitik*, die den sogenannten Up-Down Algorithmus umsetzt, und immer dann zum Tragen kommt, wenn die ClassAds von sich aus keine eindeutige Schedulingentscheidung ermöglichen.

Der Algorithmus basiert auf der Beobachtung, dass es in einem typischen Condorsystem im Normalfall zwei verschiedene Gruppen von Nutzern gibt ([5]): Die sogenannten *Heavy Users* haben einen unbeschränkten Ressourcenbedarf und nutzen diese dann auch für eine lange Zeit aus; die *Light Users* hingegen benutzen die Condor-Ressourcen nur gelegentlich und dann für eine kürzere Zeit. Um nun ein faires Scheduling der Jobs sicherzustellen, bezieht der Algorithmus die durchschnittliche Nutzung des Condor-Systems durch die Jobbenutzer mit ein. Jeder Benutzer erhält eine Priorität, die sich aus dem Quotient der erhaltenen Rechenzeit und der Wartezeit des Benutzers ergibt; ist dieser Quotient klein (was für die Light Users meistens zutrifft), werden die Jobs dieses Benutzers bevorzugt geschedulet. Ist sie groß, müssen die Jobs gegenüber Jobs mit höherer Priorität zurücktreten.

2.4.2 Checkpoiting und Migration

Wie schon weiter oben ausgeführt, ist ein Condor-System vor allem dafür gedacht, relativ rechenintensive und damit lang laufende Jobs auszuführen; zugleich handelt es sich allerdings um eine opportunistische Umgebung, bei der der Scheduler kaum Annahmen über die zukünftige Verfügbarkeit einer Ressource machen kann. Man kann also nicht davon ausgehen, dass ein Job im Normalfall auf einer einzigen Ressource ohne Unterbrechung ausgeführt werden kann. Vielmehr ist für eine effiziente Ausnützung der Ressourcen ein Mechanismus vonnöten, der es erlaubt, laufende Prozesse zu unterbrechen und diese Prozesse dann später (evtl. sogar auf einem anderen Rechner) weiter ausführen zu lassen. Diese Funktionalität bietet Condor unter anderem im Standarduniversum mit der Checkpointing-Funktionalität.[9]

Ein Checkpoint ist für Condor dabei eine Beschreibung des Programmzustandes eines Jobs, so dass dieser (fast) vollständig zu einem späteren Zeitpunkt und gegebenenfalls auch auf einem anderen System wiederhergestellt werden kann; da ein Job letztlich auf Betriebssystemebene in Form eines Prozesses auftritt, muss also unter

[9] Die Idee des Checkpointing dient traditionell in der Informatik dazu, die Sicherheit einer Umgebung gegen Ausfälle zu erhöhen, das heißt zur Absicherung gegen einen Ausnahmefall. Demgegenüber ist aber im HTC-Bereich das Brechen einer Allokation der Normalfall, und stellt damit auch andere Anforderungen an die Effizienz des Algorithmus.

den verschiedenen unterstützten UNIX-Betriebssystemen der Zustand des Prozesses ausgelesen und serialisiert werden können.[10] Dafür unterbricht Condor den Job und sichert unter Anderem den Stack, das Datensegment, den Zustand der CPU, das Textsegment[11] und die Text- und Datensegmente von verwendeten Bibliotheken. Auch gesichert werden müssen der Status von gerade geöffneten Dateien, von Signalhändlern und noch nicht behandelten Signalen. Im MPI und PVM-Universum wird Checkpointing auch für Programme unterstützt, die gerade Netzwerkverbindungen offen haben. In diesem Fall muss allerdings die Gegenseite auch ein Condor-Job sein, um sicherstellen zu können, dass die Gegenseite mit dem zeitweiligen „Verschwinden“ des Kommunikationspartners zurückkommt, und dass keine Pakete mehr im Netzwerk unterwegs sind, die Inkonsistenzen bei der Wiederherstellung des Jobs bewirken könnten.

Voraussetzung für das Checkpointing und die Migration von Prozessen im Standard-Universum ist, dass diese gegen eine spezielle Bibliothek von Condor (der System Call Library) gelinkt wurden. Diese bietet das gleiche Interface wie die Standard C Bibliothek und fügt unter anderem einen Signalhandler ein, der auf das Checkpoint Signal reagiert, sowie Methoden, um einen serialisierten Prozess vollständig wieder herzustellen. Das Programm selber muss nicht zwangsweise angepasst werden, allerdings gibt es verschiedene Voraussetzungen, die bei der Programmierung zu beachten sind: So ist jede Art von Nebenläufigkeit (sei es in Form von Prozessen oder Threads), die Benutzung der meisten Formen von Interprozesskommunikation oder Alarmen, Timern und Sleep-Funktionen erst einmal nicht erlaubt. Ist dies nicht akzeptabel, kann der Entwickler auch zeitweise die Checkpointing-Funktionalität des Jobs programmatisch deaktivieren und aktivieren.

Problematisch an der Idee des Checkpointing sind vor allem die potentiell hohen Laufzeitkosten: So müssen bei jedem Checkpoint alle Daten- und Textsegmente des Prozesses ausgelesen werden, was das Swappen von virtuellen Speicherseiten von der Festplatte in den Speicher zur Folge haben kann. Soll der Prozess migriert werden, müssen die gesamten Informationen auch über das Netzwerk fließen. Um diese Kosten im Verhältnis zum Nutzen des Checkpointing zu halten, ist Condor bei der Entscheidung, wann genau ein Checkpoint erzeugt wird, sehr gut konfigurierbar.

Standardmäßig erzeugt Condor regelmäßig automatische Checkpoints (die genaue Zeitspanne zwischen diesen lässt sich einstellen); außerdem wird im Falle der Verfügbarkeit eines höherprioren Jobs oder des Abbruchs von Jobs aufgrund der Ressourcenpolitik ein Checkpoint angefertigt. Dauert dieser allerdings im letzteren Fall zu lange, wird er abgebrochen, um gegebenenfalls den Benutzer nicht zu lange auf sein System warten zu lassen. Wie weiter oben schon beschrieben, kann der Programmierer eines Jobs selbst die Checkpointing-Funktionalität aktivieren und deaktivieren und auch selbst via `ckpt()` die Erstellung eines Checkpoints erzwingen. Checkpoints aller betroffenen Jobs werden ebenfalls bei den Condor-Befehlen `condor_checkpoint`,

[10] Von den unterstützten Systemen bietet nur das Irix System von Silicon Graphics von sich aus eine Art Prozess-Checkpointing an.

[11] Das Textsegment eines Prozesses ändert sich im Normalfall (bei sich nicht selbst modifizierenden Programmen) zwar nicht, und könnte deswegen immer wieder auch von der Submitter-Machine geladen werden; allerdings weist [5] darauf hin, dass sich dann die Programmdatei auf der Submitter-Machine während der gesamten Laufzeit des Programmes nicht ändern darf, was bei langläufigen Jobs schwer sicherzustellen ist.

`condor_vacate`, `condor_off` und `condor_restart` erstellt.

2.4.3 Remote System Calls

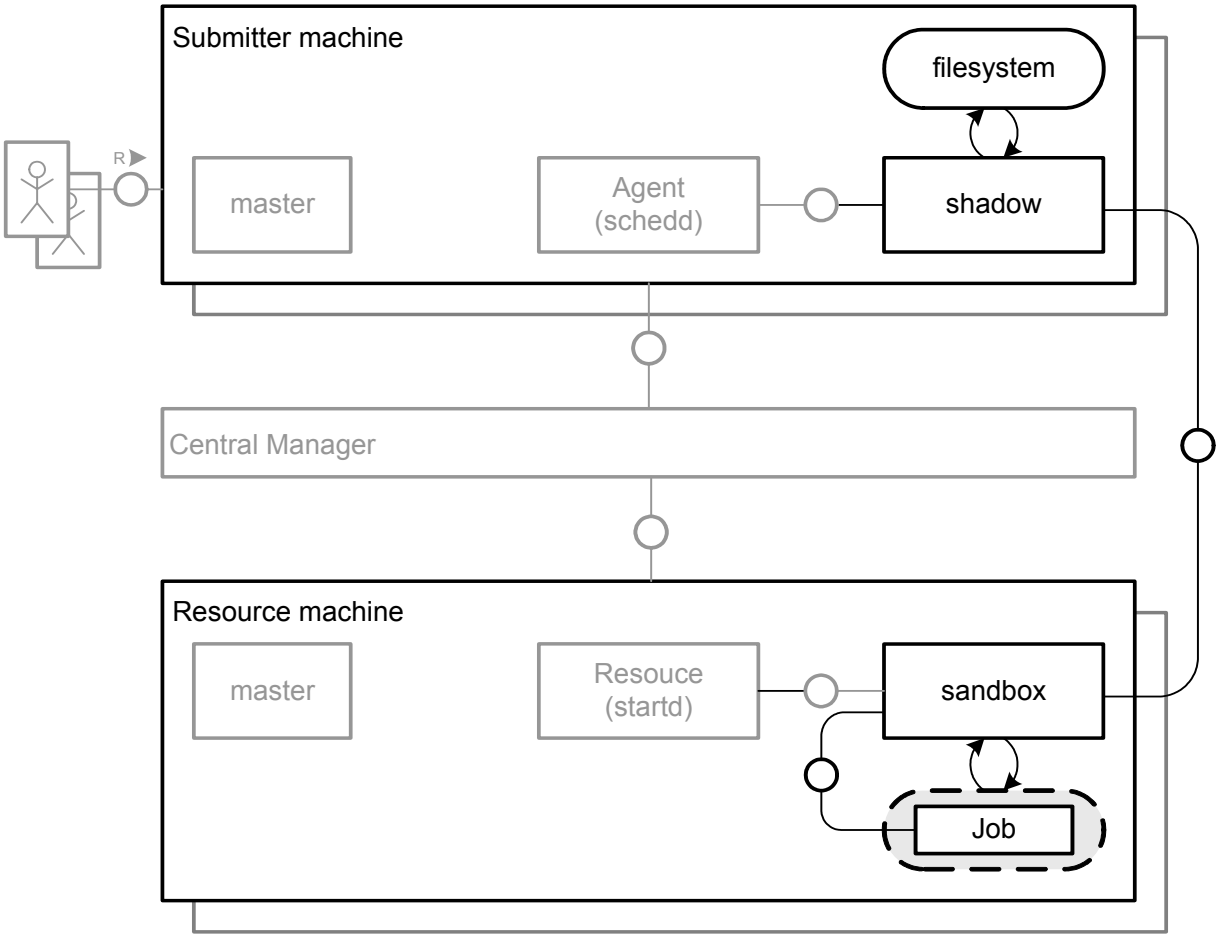


Abbildung 4: Ein Remote System Call im Condor-System

Eine der Herausforderungen für einen verteilten Scheduler wie Condor ist es, den ausgeführten Programmen – die während ihrer Ausführung zum Teil sogar von einem auf einen anderen Rechner migriert werden können, wie im vorausgegangenen Abschnitt beschrieben – zu jeder Zeit eine möglichst gleiche Umgebung zur Verfügung zu stellen. Dies ist insbesondere wichtig für den Zugriff auf Dateien, da normalerweise jeder Condor Job Dateien lesen und/oder schreiben wird.

Condor bietet mehrere Mechanismen, um Jobs den Zugriff auf von ihnen benötigte Dateien zu ermöglichen. So können für Jobs im Vanilla Universum Dateien spezifiziert werden, die dann vor dem Start des Jobs zur Resource-Machine hin und nach Beendigung des Jobs zur Submitter-Machine zurück kopiert werden. Dies kann allerdings für große Dateien sehr ineffizient sein und setzt zudem voraus, dass dem Ersteller der Jobbeschreibung im Voraus die Namen der benötigten Dateien bekannt sind. Auch

funktioniert diese Vorgehensweise nicht bei speziellen Dateien, wie sie unter Unix benutzt werden, um etwa auf Geräte etc. zuzugreifen.

In vielen Umgebungen steht auch ein verteiltes Dateisystem zur Verfügung, auf dem Condor aufbauen kann und auch tatsächlich wenn möglich nutzt. Allerdings soll Condor ja auch gerade in heterogenen Umgebungen und zwischen vergleichsweise lose gekoppelten Arbeitsplatzrechnern funktionieren, bei denen ein gemeinsames performantes Dateisystem oft nicht vorhanden ist: Für diese Umgebungen bietet Condor das System der Remote System Calls.[12]

Voraussetzung für die Benutzung von Remote System Calls ist – wie schon beim Checkpointing, welches im Abschnitt 2.4.2 auf Seite 4-13 beschrieben wurde – dass das Executable mit der Condor System Call Library gelinkt wurde. Diese nimmt die I/O Systemrufe des Programmes entgegen und sendet sie an den Shadow-Prozess auf der Submitter-Machine, der dann den Systemruf lokal ausführt und evtl. Ergebnisse wieder an den Job auf der Resource-Machine zurückschickt. In Abbildung 4 auf Seite 4-15 ist dieses Zusammenspiel noch einmal im Gesamtaufbaubild des Condor-Systems eingeordnet (Die Sandbox ist dabei eine Abstraktion der Rolle der System Call Library).

Da auch die Remote System Calls sich mit dem potentiell langsamen Netzwerk als Mittler äußerst ungünstig auf die Performance ausüben können, finden im Condor-System noch einige Optimierungen statt. So erkennt der Shadow-Prozess etwa, ob die angeforderte Datei der Resource-Machine eventuell über ein verteiltes Dateisystem zur Verfügung steht, und schickt gegebenenfalls die Adresse der Datei samt dem zu verwendeten Protokoll[13] an den Prozess auf der Resource-Machine zurück.

Neben der größeren Flexibilität gegenüber den anderen Lösungen bieten die Remote System Calls auch noch ein paar andere Vorteile: So brauchen die Condor-Jobs auf der Resource-Maschine keinerlei Zugriff auf das lokale Dateisystem und können zum Beispiel unter einem sehr eingeschränkten Benutzeraccount laufen. Des weiteren erlaubt der Dateizugriff über dem Shadow-Prozess diesem, transparent Log-Files über die I/O-Aktivitäten von Condor-Jobs führen zu können. Allerdings setzen Remote System Calls wie weiter oben schon angesprochen das Linken der Programme mit einer speziellen Condor Library voraus, was unter Anderem auch dazu führt, dass diese Funktionalität nur unter UNIX-Derivaten zur Verfügung steht und man Zugriff auf den Quell- oder zumindest Objektcode der Executables benötigt.

3 Condor und Grid Computing

Anfang der 90er fand das Gebiet des verteilten Rechnens großen Zulauf: In der Wissenschaft und auch in der Wirtschaft versuchte man, statt teurer Supercomputer verhältnismäßig billige gekoppelte Standardkomponenten einzusetzen, was unter anderem zu der Entwicklung und dem Einsatz von leistungsfähigen Batch-Systemen wie LoadLeveler (einem Abkömmling von Condor), LSF, Maui, NQE und PBS führte [4].

[12] Remote System Calls stehen normalerweise nur im Standard-Universum zur Verfügung; für Java Programme gibt es allerdings eine ähnliche Funktionalität unter dem Namen *Chirp*, die allerdings die Anpassung des Java-Quellcodes erfordert.

[13] Im Augenblick werden HTTP, GridFTP, NeST und Kangaroo unterstützt.

Gleichzeitig wurde die Vision eines *Computer Grids* populär, welches das gemeinsame Benutzen von Resourcen über administrative und organisationelle Grenzen hinweg ermöglichen sollte. Im Folgenden werden die drei Varianten, in denen Condor eine Rolle im Grid spielt, betrachtet.

3.1 Flocking

Condor-Pools sind im Allgemeinen auf Organisationen beschränkt und umspannen nicht mehrere administrative Domänen. Selbst innerhalb einer Organisation können mehrere Condor-Pools parallel existieren. In bestimmten Situationen kann es aber sinnvoll sein, mehrere Pools zu verknüpfen, um Jobs aus einem Pool auf Rechnern aus dem anderen Pool auszuführen. Zu diesem Zweck gibt es in Condor zwei Flocking-Mechanismen.

3.1.1 Gateway Flocking

Gateway Flocking wurde 1994 eingeführt und ist in älteren Condor-Versionen noch verfügbar, wird aber in aktuellen Versionen aufgrund zu hoher Komplexität und der Notwendigkeit, fortwährend Änderungen an den Gateways vornehmen zu müssen nicht mehr bereit gestellt.

Beim Gateway Flocking werden zwei oder mehr Condor-Pools mit Gateways verbunden. Es findet also eine Zusammenarbeit auf administrativer Ebene statt. Zwischen Gateways werden Vertrauensstellungen erstellt, die den Austausch von Jobs ermöglichen.

Gateway Flocking ist für den Benutzer vollständig transparent. Er schickt einen Job in seinem lokalen Condor-Pool ab. Steht keine Maschine zur Durchführung des Jobs zu Verfügung, kontaktiert der Matchmaker des lokalen Pools den Matchmaker eines über Gateways mit diesem verbunenen Pools. Wenn hier freie Ressourcen zur Verfügung stehen, wird der Job im fremden Pool ausgeführt (siehe Abbildung 5).

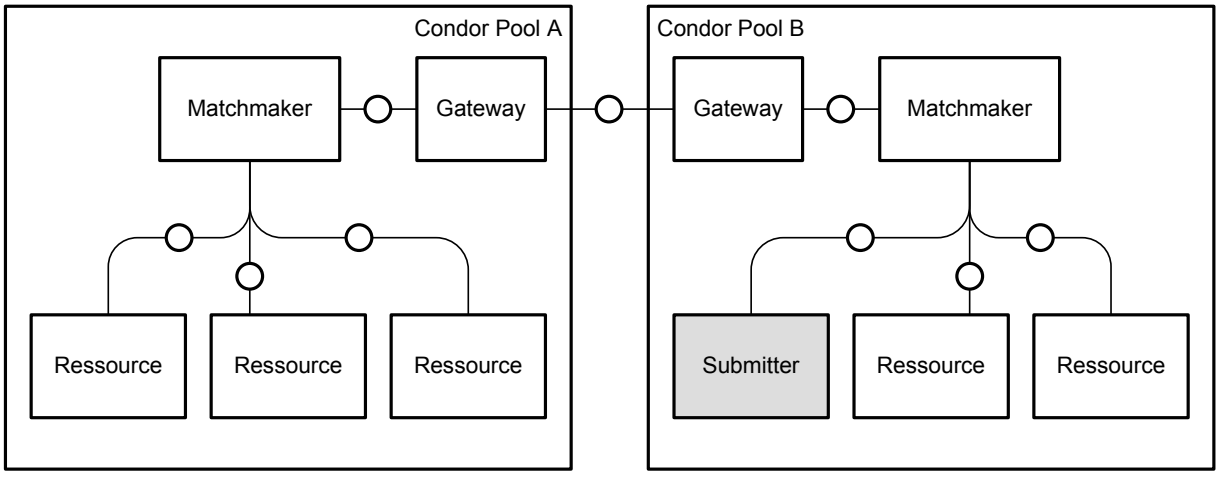


Abbildung 5: Condor Gateway Flocking

Über Politiken lässt sich ein Gateway ähnlich wie ein Job bzw. eine Ressource konfigurieren, da es in einer dieser Rollen auftritt. Die Konfigurationen können unterschiedlich sein, so dass verschiedene Kriterien für das Weiterleiten oder Annehmen von Jobs existieren können.

3.1.2 Direct Flocking

Während das Gateway Flocking transparent mehrere Organisationen verbindet, stellt das Direct Flocking lediglich Verbindungen zwischen einem Benutzer und einem (oder mehreren) fremden Condor-Pool(s) her (Abbildung 6).

Es erwies sich als schwierig, Vertrauensstellungen zwischen Organisationen und deren Condor-Pools zu etablieren. Beim Direct Flocking ergreift lediglich ein einzelner Benutzer die Initiative und beantragt den Zugriff auf einen fremden Condor-Pool. Dieses Verfahren ist wesentlich einfacher, da es weniger administrativen und organisatorischen Aufwand erfordert und dem fremden Pool die vollständige Kontrolle über Zugriffe entfernter Benutzer ermöglicht.

Ein Benutzer, der Direct Flocking benutzt, wählt selbst aus, zu welchem Pool er seinen Job übermitteln möchte. Die Transparenz ist also ein Stück weit eingeschränkt, die Kontrollmöglichkeiten sind aber größer.

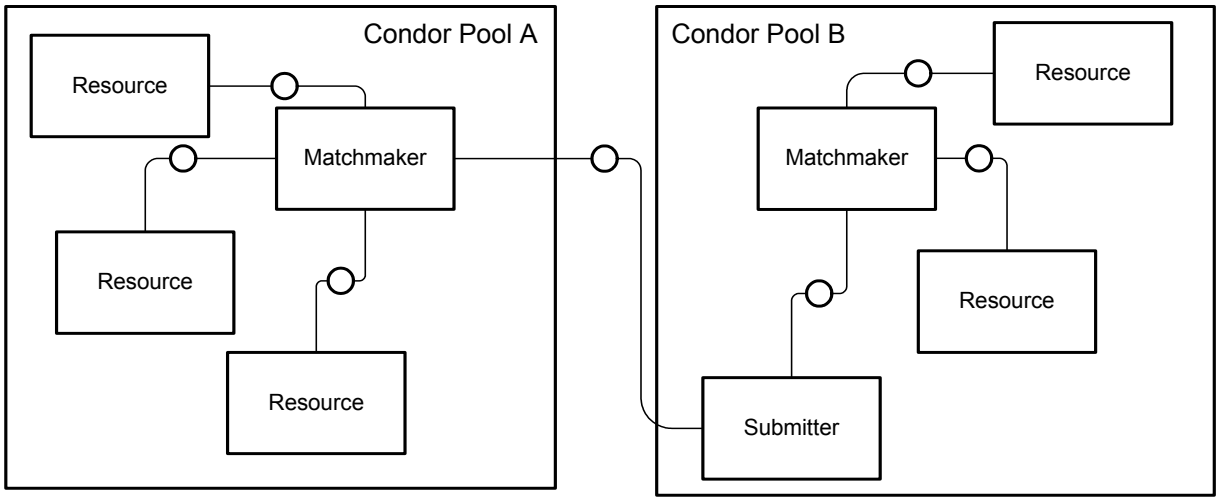


Abbildung 6: Condor Flocking

3.2 Condor-G

Condor-G stellt eine Symbiose von Condor und Globus dar. Globus liefert die Protokolle für sichere Interdomänen-Kommunikation (GRAM) und stellt den Zugriff auf entfernte Batch-Systeme zur Verfügung, während Condor als Job-Scheduler im Globus-Grid arbeitet, das heißt es stellt die Mechanismen zur Übermittlung von Jobs und Ressourcenallokation zur Verfügung.

Condor-G ist somit ein Condorsystem, welches dahingehend erweitert wurde, dass es mit dem GRAM-Protokoll umgehen kann. Dadurch ist es möglich, ein Globus-Grid genau so zu bedienen wie einen Condor-Pool. Jeder Cluster im Grid besitzt in der Regel eine eigene Batch-Queue. Condor-G stellt Jobs in diese Queues und administriert sie, das heißt, es zeigt dem Benutzer den Status der Jobs und stellt im Fehlerfall eine Fehleranalyse auf und gegebenenfalls den Job wieder in die Queue. Zudem stellt Condor-G auch eine eigene Queue, Prioritäten, Accounting und Logging zur Verfügung, wie dies auch im Condor-Pool unterstützt wird. Siehe dazu Abbildung 7.

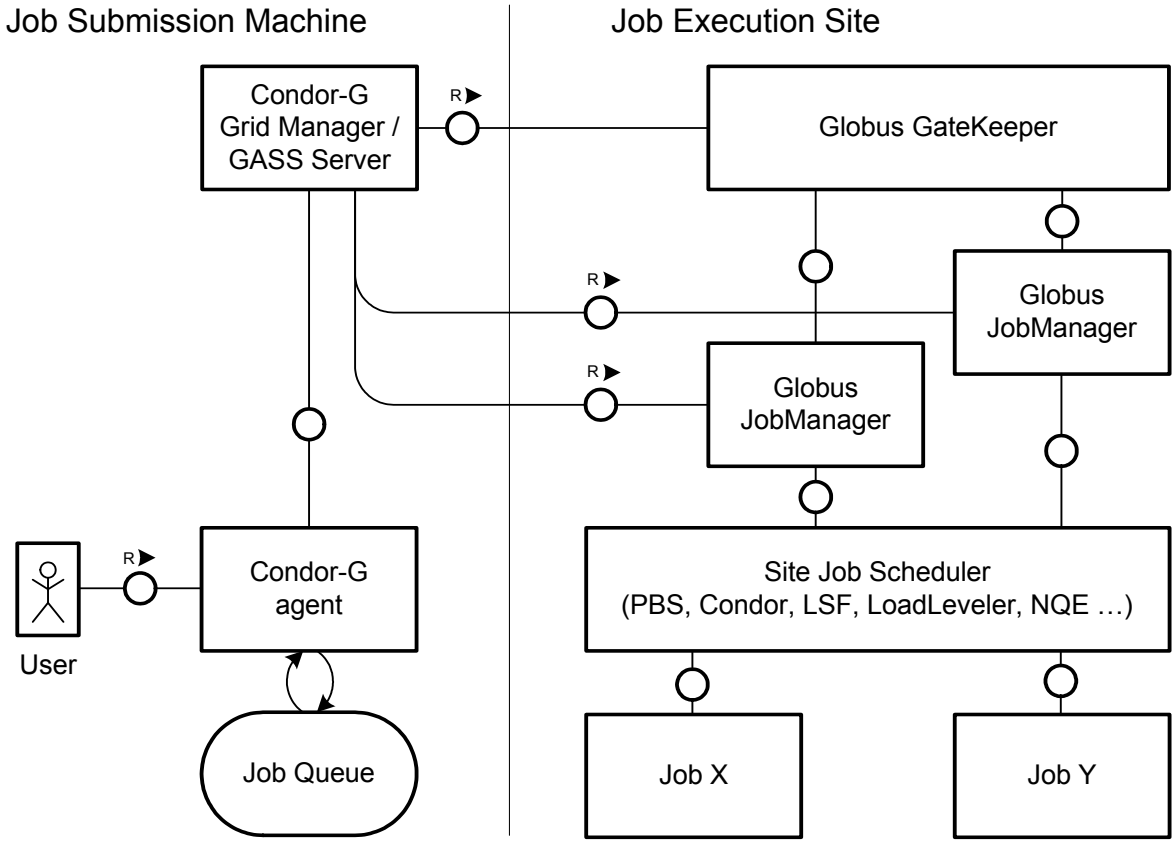


Abbildung 7: Condor-G

Leider hat das GRAM-Protokoll aufgrund seiner Universalität einige Nachteile. Da es nicht die Feinheiten aller Batch-Systeme unterstützen kann, ist das Protokoll relativ allgemein gehalten. Die Übermittlung eines Jobs erfolgt an eine Queue eines Batch-Systems. Die Informationen über diese Queue sind allerdings sehr gering. Es werden keine Informationen bzgl. der Länge der Queue oder der Verfügbarkeit dahinter liegender Ressourcen publiziert. So kann es vorkommen, dass ein Job an eine sehr lange Queue übergeben wird, obwohl andere Cluster mit kurzen Queues zur Verfügung stünden. In diesem Fall kann ein Benutzer also seinen Job in viele Queues stellen und warten, bis der erste Job davon gestartet wird. Oder aber man übermittelt den Job nur an eine (oder wenige) Queues mit der Gefahr, nicht schnellstmöglich zur Ausführung zu gelangen. Dieses Problem wird vom GlideIn-Mechanismus in Condor gelöst.

3.2.1 GlideIn

Mit *GlideIn* (oder auch *gliding in*) wurde ein Verfahren entwickelt, das es ermöglicht, die Rechenleistung eines Globus-Grid mit dem Komfort eines lokalen Condor-Pools zu kombinieren.

Das GlideIn erfolgt in drei Schritten:

Schritt 1: Der Nutzer startet mit Condor-G Jobs auf entfernten Maschinen eines Globus-Grid. Diese Jobs beinhalten die Standard-Condor-Dämonen und werden auf der Zielmaschine als gewöhnliche Jobs behandelt. Auf dem Rechner des Nutzers wird ein Matchmaker gestartet. Der Condor-G-Agent des Nutzers läuft weiterhin (Abbildung 8).

Schritt 2: Werden die Jobs und damit die Dämonen zur Ausführung gebracht, so bilden sie mit den lokalen Komponenten einen persönlichen Condor-Pool, auf den der Nutzer zugreifen kann (Abbildung 9 auf Seite 4-21).

Schritt 3: Condor-Jobs können nun an den Condor-G-Agenten übergeben und dann auf die Ressourcen verteilt und ausgeführt werden.

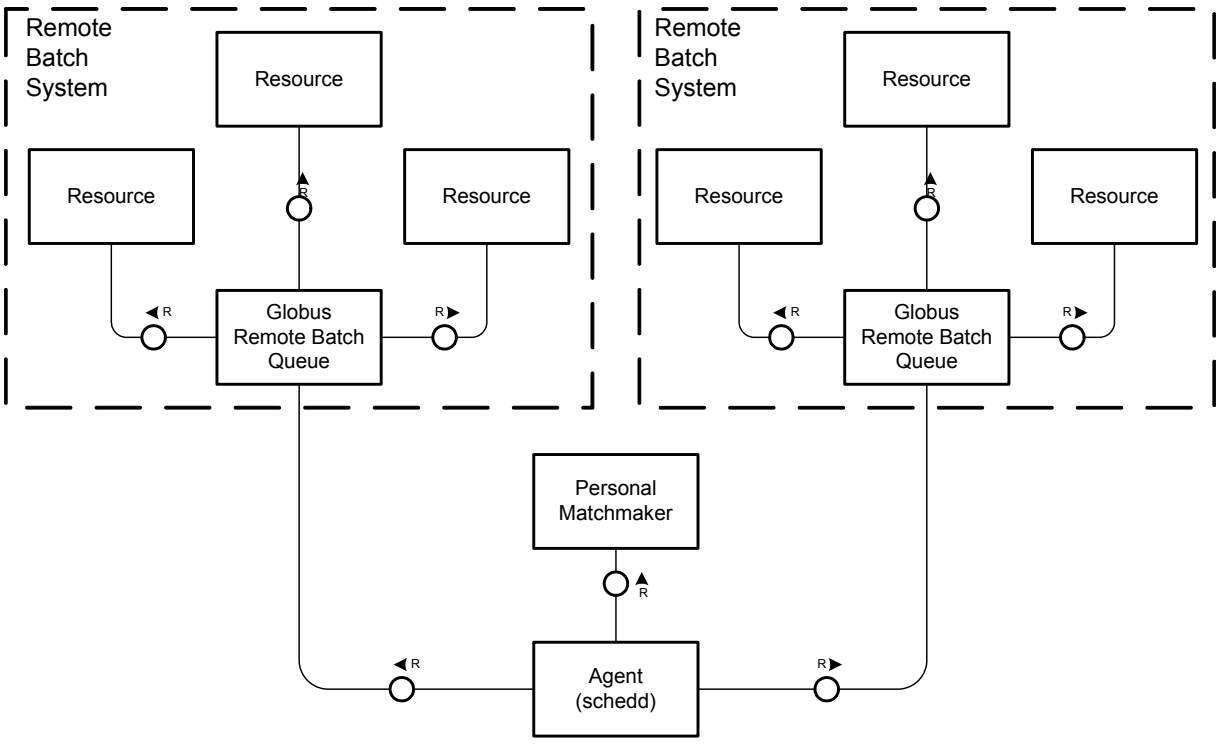


Abbildung 8: GlideIn, Schritt 1

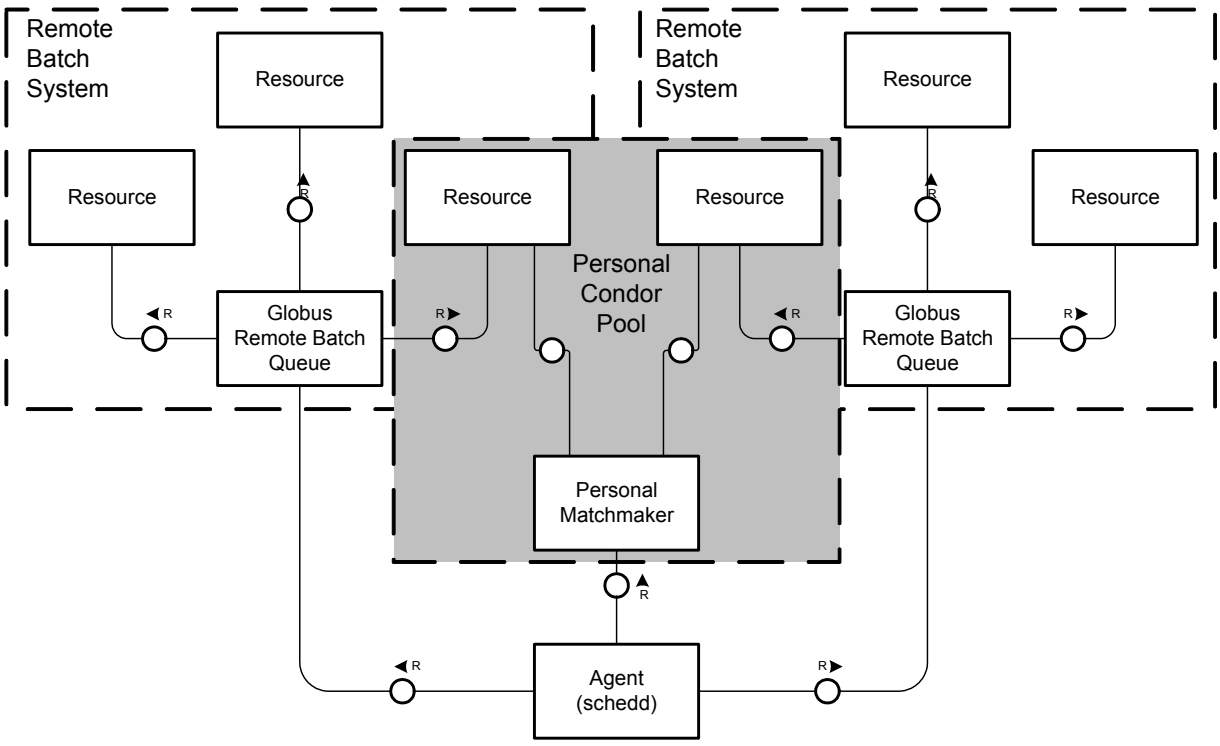


Abbildung 9: GlideIn, Schritt 2

3.3 Globus Site Job Scheduler

Eine weitere Möglichkeit, Condor im Grid einzusetzen ist als *Globus Site Job Scheduler*. Hier wird ein Condor-Pool dem Globus-Grid zur Verfügung gestellt und Jobs werden über das Grid an den Pool geschickt. Der Condor-Pool verhält sich hier wie eine beliebige Ressource im Grid (Cluster, Parallelrechner).

Condor übernimmt in einem solchen Szenario die Rolle des *Local Scheduler* unterhalb des Globus Job-Manager und ist damit eines der vielen möglichen Batchsysteme, die zusammen mit Globus eingesetzt werden können.

4 Einsatz in der Praxis

Wie im Abschnitt 2.1 auf Seite 4-3 beschrieben, startete Condor 1988 als Forschungsprojekt an der Universität von Wisconsin-Madison, an der es bis heute weiterentwickelt wird, und dient auch heute noch als Experimentierfeld für verschiedenste Forschungsprojekte im Bereich des High Throughput Computing (für eine Übersicht siehe [1]). Allerdings ist Condor schon seit seinen Anfängen auch als Produktivsystem im Einsatz und wird zur Zeit für eine ganze Reihe von Projekten in der Forschung und Industrie eingesetzt. Im Folgenden werden wir daher die häufigsten Einsatzfelder von Condor beschreiben, um dann kurz einige Anwendungsbeispiele von Condor zu präsentieren.

4.1 Einsatzfelder

Condor wurde ursprünglich dafür entwickelt, ungenutzte Rechenzeiten von Desktop Workstations nutzbar zu machen (siehe unter Anderem [5]), und eignet sich natürlich auch immer noch ideal für dieses Einsatzfeld, auch wenn durch immer weiter fallende Hardwarekosten diese Idee nicht mehr die frühere Bedeutung hat. Allerdings erwies sich im Laufe der Zeit, dass das gleiche Systemdesign, dass das „Stehlen“ von CPU-Zyklen möglich macht, auch geeignet ist, um unter anderem Cluster, Multiprozessor-Maschinen und weitverteilte Systeme zu kontrollieren. Im Folgenden werden daher die wichtigsten heutigen Einsatzfelder für Condor umrissen.

Die wohl einfachste Möglichkeit, Condor zu nutzen, ist der Einsatz auf einem einzigen Rechner. Condors Fähigkeiten, eine große Anzahl von lang laufenden Jobs komfortabel zu verwalten und ihre Ausführung von Umgebungsparametern des Rechners abhängig zu machen, lässt sich auch lokal gut einsetzen, um im Hintergrund (eventuell mit Hilfe des DAGMan-Metaschedulers) vollautomatisch langlaufende Berechnungen durchzuführen, ohne dass der Rechner dadurch für den interaktiven Einsatz unbenutzbar würde oder ständig laufen müßte.

Eine andere Domäne, bei der Condor erfolgreich mit anderen, zum Teil spezialisierteren Systemen (wie [10], [15], [11], [12], [14] und [16]) konkurriert, ist der Einsatz als Cluster Submission Tool. In diesem Zusammenhang ist auch die gute Unterstützung für MPI und PVM von Bedeutung, sowie die Möglichkeit, SMP-Maschinen unter Condor effizient auszulasten.

Auch im Bereich der weitverteilten Systeme und des Grid-Computing spielt Condor eine gewichtige Rolle. In den Abschnitten 3.1 auf Seite 4-17 und 3.2 auf Seite 4-18 wurde schon genauer beschrieben, welche Mechanismen Condor selbst für das Nutzen von Rechenzeit über administrative Grenzen hinweg bietet, bzw. wie es mit dem Globus System zusammenspielen kann.

4.2 Anwendungsbeispiele

Condor wurde in der Wissenschaft, aber auch in der Wirtschaft schon in zahlreichen Fällen eingesetzt. Zu den 150 offiziellen Nutzern oder Unterstützern von Condor im Jahre 2002 zählen unter anderem verschiedenste Universitäten, die US Airforce, das Department of Defence der USA, die NASA, IBM, Intel, Microsoft und Oracle. Condor wurde und wird dafür eingesetzt, um zum Beispiel 3D Film-Szenen zu rendern, Videos zu encoden, das Verhalten des Aktienmarktes zu simulieren, das menschliche Genom zu untersuchen, die Statik eines Hauses zu berechnen, elektronische Schaltungen zu simulieren und virtuelle Windtunnelexperimente mit Autos durchzuführen.

Trotz diesem vielfachen Einsatz finden sich allerdings nur wenig gut dokumentierte Condor-Installationen. Zwei davon, die Installation an der UW-Madison und bei Oracle, sollen im Folgenden kurz vorgestellt werden.

4.2.1 Einsatz an der UW-Madison

Der wohl am besten dokumentierte ([1], [8]) Einsatz von Condor findet im Department of Computer Sciences an der Universität von Wisconsin-Madison selbst statt. Das erste Produktivsystem wurde dort vor fast 15 Jahren installiert. Das dortige Condor-System bestand Anfang 2003 aus einem Pool mit mehr als 1000 CPUs, wobei weitere 1000 CPUs in diversen kleineren Condor-Pools in anderen Abteilungen angeschlossen sind. Im Schnitt stellt allein der große Pool dabei täglich vierzehn bis fünfzehntausend CPU-Stunden zur Verfügung.

Die universitätseigenen Condor-Pools werden für verschiedenste Aufgaben genutzt. Beispiele sind laut [1] die Untersuchung von lernenden Algorithmen, die Simulation von Phasentransitionen in der Physik, die Untersuchung des Discrete Element Modelling Problems, die Simulation von Prozessoren und die Forschung an Verwirbelungen in Diesel-Motoren.

4.2.2 Einsatz bei Oracle

Ein Beispiel für den Einsatz von Condor in der Industrie sind mehrere, über drei Kontinente verteilte Condor-Pools bei Oracle, die für die Entwicklung des Version 9 Data Servers geschaffen wurden: Als Pilotprojekt 1998 gestartet, bestehen die Pools laut [9] im Augenblick aus 1700 Knoten, und sollen noch um weitere 7000 Rechner erweitert werden. Angewendet wurden die Pools vor allem für umfangreiche Regressionstests und die tägliche Übersetzung des Datenbanksystems.

5 Zusammenfassung

Condor ist ein an der Universität von Wisconsin-Madison entwickeltes System, welches das Bündeln und Benutzen von heterogenen Rechenressourcen in Netzwerken verschiedenster Größen erlaubt. Ursprünglich entwickelt mit dem Ziel, ungenützte Rechenzeiten von Workstations zu nutzen, kann es heute auch als Scheduler für Cluster oder im Grid-Umfeld (via Condor-G und als Site Job Scheduler) eingesetzt werden. Für die Ausführung unter Condor geeignet sind dabei alle Rechenaufgaben, die keine interaktiven Eingaben erfordern und hinreichend viel Rechenzeit benötigen.

Das System zeichnet sich insbesondere durch die große Flexiblität bei der Anpassung an vorhandene Infrastruktur bzw. Anforderungen an das Scheduling von Jobs aus. Es kann auch mit verhältnismäßig langsamen und unzuverlässigen Netzwerken gut umgehen und bietet vielerlei Konfigurationsmöglichkeiten. Mittels verschiedener Laufzeitumgebungen (Universen) können eine Vielzahl von Programmen, zum Teil ohne Modifikationen, von Condor ausgeführt werden.

Besonders hervorzuheben sind bei Condor die mächtigen Features des Matchmaking, Checkpointing und der Remote System Calls. Ersteres ist eine äußerst flexible Möglichkeit, den Schedulingalgorithmus zu beeinflussen, letztere sorgen für eine Umgebung auf den Gastsystemen, die laufende Jobs sichern und auf andere Rechner

migrieren kann, bzw. stellen eine konsistente Umgebung auf allen Gastsystemen sicher.

Abbildungsverzeichnis

1 An einem Condor-System beteiligte Gruppen 4-5
2 Grundsätzlicher Aufbau eines Condor-Systems 4-6
3 Die Dämonen eines Condor-Systems 4-7
4 Ein Remote System Call im Condor-System 4-15
5 Condor Gateway Flocking 4-17
6 Condor Flocking 4-18
7 Condor-G 4-19
8 GlideIn, Schritt 1 4-20
9 GlideIn, Schritt 2 4-21

Quelltextverzeichnis

1 Beispiel-ClassAd einer Ressource 4-12
2 Beispiel-ClassAd eines Jobs 4-12

Literatur

[1] *Condor Project Homepage*
http://www.cs.wisc.edu/condor

[2] Condor Team, University of Wisconsin-Madison
Condor Version 6.4.7 Manual
http://www.cs.wisc.edu/condor/manual/v6.4

[3] Ian Foster, Carl Kesselman
The Grid: Blueprint for a New Computing Infrastructure
1999 Morgan Kaufmann Publishers, Inc., San Francisco

[4] Fran Berman, Anthony J.G. Hey, Geoffrey Fox
Grid Computing: Making the Global Infrastructure a Reality
2003 John Wiley & Sons, Ltd

[5] Michael J. Litzkow, Miron Livny, and Matt W. Mutka
Condor - A Hunter of Idle Workstations
In Proc. 8th Intl. Conf. Distributed Computing Systems, San Jose, California
1988 IEEE Computer Society Press

[6] M. Livny, J. Basney, R. Raman, T. Tannenbaum
Mechanisms for High Throughput Computing
Department of Computer Sciences, University of Wisconsin-Madison
1997

[7] Rajesh Raman, Miron Livny, Marvin H. Solomon
Matchmaking: Distributed Resource Management for High Throughput Computing
In Proc. 7th IEEE International Symposium on High Performance Distributed Computing
July 1998
http://www.acis.ufl.edu/~ivan/bibliographic/Library/DistributedComputing/Condor/hpdc98.pdf

[8] Neil Alger, Miron Livny
Interview with the visionary Miron Livny of University of Wisconsin
GRID Today Vol. 2 No. 6
http://www.gridtoday.com/03/0210/101048.html

[9] Surendra Reddy
Grid Computing @ Oracle
www.ppdg.net/mtgs/Troubleshooting/Oracle

[10] *Portable Batch System Homepage*
http://www.openpbs.org

[11] *Distributed Queueing System (DQS) Homepage*
http://www.scri.fsu.edu/ pasko/dqs.html

[12] *Generic Network Queuing System (GNQS) Homepage*
http://www.gnqs.org

[13] *Sun ONE Grid Engine Homepage*
http://wwws.sun.com/software/gridware

[14] *Load Sharing Facility (LSF) Homepage*
http://www.platform.com

[15] *Network Queuing Environment (NQE) Homepage*
http://www.cray.com/products/software/nqe

[16] *LoadLeveler Homepage*
http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/loadleveler.html

[17] *Fundamental Modeling Concepts Web Site*
http://fmc.hpi.uni-potsdam.de

The Cactus Framework

Thomas Hille, Martin Karlsch

(thomas.hille, martin.karlsch)@hpi.uni-potsdam.de

Today's physical simulations need an extraordinary amount of computational power. It is getting harder and harder to satisfy these needs with a single processor machine. A solution to this problem is to distribute the calculations among many connected processors/machines. However new applications must be created in consideration of this new distributed architecture and existing applications have to be modified. This is a task which requires the creators of the simulations to do a massive additional amount of work. How the Cactus framework helps to reduce this workload we will discuss now.

1 Overview

Cactus is an open source problem solving environment designed to provide a unified modular and parallel computational framework for physicists and engineers. The Cactus Code was originally developed to provide a framework for the numerical solution of Einstein's Equations, one of the most complex sets of partial differential equations in physics. These equations govern such cataclysmic events as the collisions of black holes or the supernova explosions of stars. The solution of these equations with computers continues to provide challenges in the fields of mathematics, physics and computer science. The modular design of Cactus enables people and institutes from all these disciplines to coordinate their research, using Cactus as the collaborating and unifying tool.

2 What is Cactus?

The name Cactus comes from the design of a central core (or Flesh) which connects to application modules (or Thorns) through an extensible interface. Thorns can implement custom developed scientific or engineering applications, such as the Einstein solvers, or other applications such as computational fluid dynamics or provide a range of capabilities, such as parallel I/O, data distribution, or checkpointing.

2.1 Purpose of Cactus

The large and varied computational requirements of relativistic problems, such as black hole collisions, make them a good example for demonstrating the need for Grid computing, and an ideal test bed for developing solutions. Developing the Cactus infrastructure was an important step to use several computers and processors to

distribute large computations. Cactus provides scientist a comfortable environment for multi-dimensional simulations. The framework is freely available for everybody, because it is an open source project. It is portable, generic and modular. The Cactus framework is characterized by the easy use for scientists.

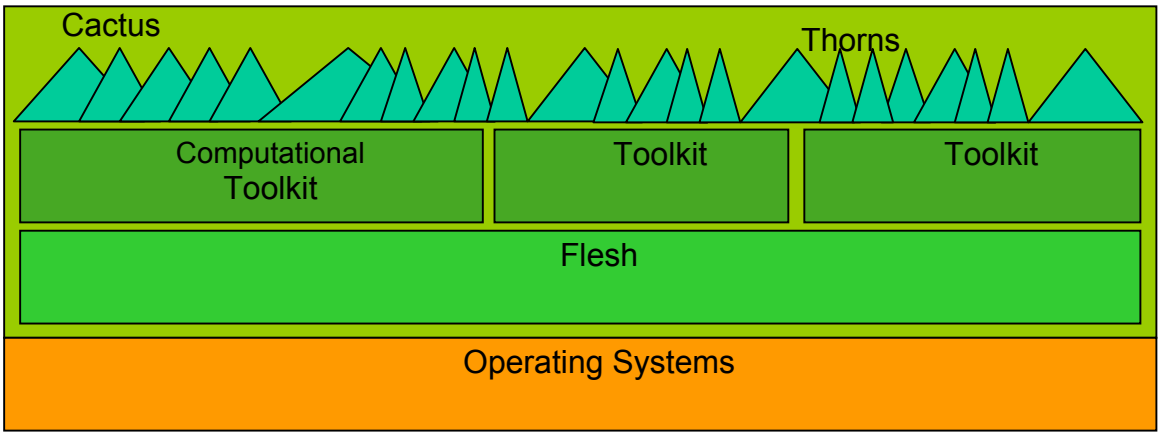


Figure 1 Cactus architecture overview [3]

2.2 History

In January 1997, Paul Walker and Joan Masso began to develop Cactus. Cactus 1.0 was a re-working of the Framework for uni-grid applications using a simple parallel uni-grid driver as a temporary replacement for DAGH [1]. (The parallelization software DAGH was the centerpiece of a toolkit called “Supertoolkit” developed by the “Black Hole Grand Challenge Alliance”. The “Framework” was a prototype framework designed and developed by Paul Walker at the Max Planck Institute for Gravitational Physics with the idea that different physics codes could be plugged in and that it would be an interface to DAGH to provide parallelism.)

In Cactus there were two types of objects, the “Flesh”, a core component providing parameter parsing, parallelization, and IO, and “Thorns” which are optional modules compiled in. In October 1997 Cactus 2.0 was released, providing for the first time the ability to dynamically pick which routines were run at run-time, and some ability to order them, this ability removed the necessity of modifying the Flesh to support new Thorns. In these early versions of Cactus all Thorns received all variables defined for a simulation. Cactus 3.0 removed this necessity, further enhancing the modularity. Cactus became the major workhouse for a collaboration of numerical relativists spanning several large projects in Europe and the US [NCSA, INRG]. This community requires very large scale, parallel computational power, and makes use of computational resources of all types, from laptops for development to heavy-duty production simulations on virtually every supercomputing architecture, including Linux clusters, “conventional” supercomputers and Japanese vector parallel machines. Hence complete portability and very efficient parallel computation and IO are crucial for its users.

Lessons learnt through its heavy use in this community led to the development of a still more modular and refined version of Cactus, Cactus 4.0, now the Flesh

contains almost no functionality. Anything can be implemented as a Thorn, not just physics routines, but all computational science layers.

3 Design of Cactus

Cactus has to be able to run on a wide range of different machines, so it requires a flexible and modular make system. Users are able to use the same source tree to build executables for different architectures or with different options on the same architecture without having to make copies of the source tree. The make system is able to detect the specific features of the machine and compilers in use, but also allows users to add configuration information that cannot otherwise be auto-detected in a transparent and concise way, without having to learn new programming languages.

Similarly it is easy to add new modules, and as much functionality as possible is delegated to modules rather than the core. Data associated with individual modules exist in separate name spaces to allow independently developed modules to co-exist without conflict. Functionally equivalent modules are inter-changeable without other modules which use or depend on them noticing the exchange. Many modules are able to be compiled into the executable at once and the desired one picked at program startup. The framework is able to trivially build applications that work in uniprocessor, SMP or MPP environments without code modification or placement of explicit conditional statements in their code to account for each of these execution environments. If the programmer wishes to build a serial implementation on a multiprocessor platform for testing purposes, the framework supports that build-time choice.

Cactus also is able to support legacy codes, and, in particular, to support the large number of programmers who use FORTRAN 77.

All the facts can be summed up in saying that Cactus is portable and modular, abstract the interfaces to lower-level infrastructure as much as possible, is easy to use and capable of supporting legacy codes.

In order to ensure code portability, the Flesh is written in ANSI C, Thorns, however, may be written in C, C++, FORTRAN 77 or Fortran 90. To make sure that Thorns are platform independent, the configuration script determines various machine-specific details, such as the presence or absence of certain non-ANSI C functions or the sizes of variables, which vary from platform to platform. In order to avoid problems with this, Cactus defines a set of types, such as CCTK_REAL, CCTK_INT, which are guaranteed to have the same size on all platforms and in all languages. This also means that runs can produce exactly the same results on different machines, and checkpoint files created on one machine can be restarted on any other.

Cactus is an Open Source Project. Everybody can access the code and extend the existing Thorns with own creations. The Cactus code is under CVS source control. The way how to get the code is described on the Cactus homepage[CUG, CACTUS].

3.1 The Flesh

The Flesh is the core of Cactus. It provides:

1. Basic Cactus data types. Variables which should be shared between the Flesh and Thorns must be declared as Cactus data types. This ensures portability and consistency between system as mentioned before. An example type is CCTKC_INT a 32bit signed integer on all systems.
2. Utility functions and types like hash maps, linked list and parameter querying functions. These are mostly independent of the rest of Cactus
3. The scheduler which coordinates the execution of the user defined calculation functions. The scheduler can be completely customized by entries in the scheduler.ccl configuration file which is described later. When the scheduler calls a scheduled function it passes a complete set of information about the grid to the callee, containing state and other information.
4. A set of abstract functions which can be overridden by Thorns. Other Thorns can call these functions if a Thorn overrides them. Important is that an abstract function can only be overridden ones. For example an abstract communication function which is overridden by a driver Thorn. (Driver Thorns are explained later.)
5. A table of registered functions. Each Thorn can register new functions at the Flesh easily. It is possible that more than one Thorn registers its function for a special function. E.g. an I/O output function is register by a function which stores the output to a file and one which displays the output on screen. If the I/O output function is called both registered functions are invoked. It is possible to configure the behavior through configuration files, too.
6. A database of existing grid variables. Grid variables are variables which the Thorn tells the Cactus infrastructure about, to get parallelization, IO, interpolation, communication, and checkpointing. These variables are registered at the Flesh, by the Thorns, but aren't created through it. They can have access modifiers like public, which means available to every other Thorn, restricted, which means available to friend Thorns and private, which makes them only usable by the owning Thorn. Grid variables can be arbitrarily sized grid arrays, a grid functions fixed to the size of the computational domain, or grid scalars. Also additionally properties like dimension or distribution type can be defined [8].
7. Parameter checking functions which check a parameter the user passed to a function for type correctness and if specified constrains. Constrains on variables can be defined in configuration files which are described later.

As easily seen the Flesh cannot work standalone. Thorns are needed for providing functionality. How Thorns work and what different kind of Thorns exists will follow later. But the Flesh is the key component which binds Thorns together offering a generic interface and needed base functions like scheduling.

3.2 Parallelization in Cactus

3.2.1 Ghostzones and I/O

How does parallelization in Cactus work if we are running in a distributed environment? The main concept behind it is called ghostzones. In Cactus all calculations are performed on a distributed grid of values. Each member of the physical grid holds a part of the computational grid. The basic assumption is that only the values on the edges of the computational grid parts must be shared between the physical actors. The shared parts are exchanged every iteration of the calculation. These shared zones are called ghostzones and are at the moment statically defined through configuration files. Let's see an example. Imagine a 2d data grid which is distributed among two processors/machines. In figure 2 you can see this setup. During each iteration we have to calculate the new value for the blue point. The result depends on the values of circumjacent red points. As you can see each machine owns one red point which the other cannot access. But still all red points are needed for an update. What happens now is that a ghostzone is defined. The values of the points in this zone are exchanged between the machines every iteration. (How this is done is described in 3.3.2 Driver Thorns). The two exchanged areas are marked green and blue in figure 2. After the exchange both processors/machines know all values contained in the ghostzone.

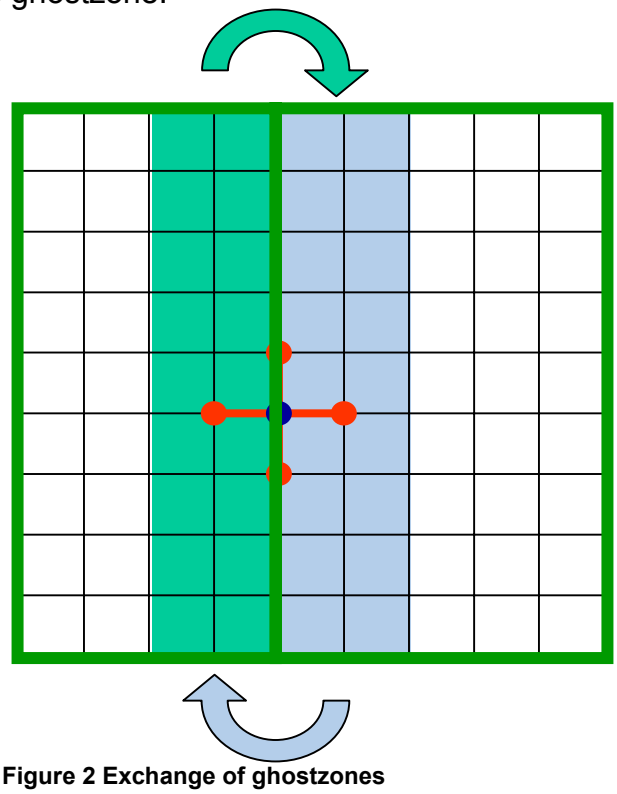


Figure 2 Exchange of ghostzones

Cactus offers also support for parallel I/O. Currently two different modes are offered through Thorns. In the first mode each processor/machine writes its own data file. When the computation is finished these files are recombined to one dataset. The second mode is that one processor/machine is in charge of gathering all calculated data after iteration and then writes it to disk. Which mode is used depends on the

used Thorn. I/O can be done through direct calls in the code or for grid variables through entries in configuration files. E.g. it is possible to specify an output file and mode (ASCII, HDF5, …) and also an interval.

3.2.2 Check pointing in Cactus

Since the state of the grid variables completely mirror the state of a calculation(iteration step state), it is enough to save all grid variables and current parameters to a dump file to create a checkpoint from which the calculation can be resumed later. Checkpointing is implemented through an I/O Thorn and available for every Cactus application which uses this Thorn. This also allows resuming the calculation on different machines. An interesting side effect is that checkpoint can be used for initial data studies.

3.3 The Thorns

The most important thing about the Thorns is that they provide all the functionality that the Cactus framework has to offer. The Flesh alone cannot do anything, because all code which is needed for computation, communication and parallelization is contained in Thorns. We subdivide the Thorns in three different categories.

- Driver Thorns
- Toolkit Thorns
- Application Thorns

The difference between these types is explained later.

Another important point is that the Thorns are virtually independent of each other. That means a Thorn which implements some computational functions has not to be aware of the Thorns which e.g. implement I/O related functions. This leads us to one of the key concept behind Thorns: the concept of the implementation. Relationships among Thorns are all based upon relationships among the implementations they provide. In principle it should be possible to swap one Thorn providing an implementation with another Thorn providing that implementation, without affecting any other Thorn.

An implementation defines a group of variables and parameters which are used to implement some functionality. For example the Thorn CactusPUGH/PUGH provides the implementation “driver”. This implementation is responsible for providing memory for grid variables and for communication. Another Thorn can also implement “driver”, and both Thorns can be compiled in at the same time. At runtime, the user can decide which Thorn providing “driver” is used. No other Thorn should be affected by this choice.

When a Thorn decides it needs access to a variable or a parameter provided by another Thorn, it defines a relationship between itself and the other Thorn's implementation, not explicitly with the other Thorn. This allows the transparent replacement, at compile or runtime, of one Thorn with another Thorn providing the same functionality as seen by the other Thorns. An important fact is that the actual communication between the Thorns is mostly handled by the Flesh.

3.3.1 Arrangements

Arrangements are a concept which you cannot avoid if you have to work with the Cactus toolkit. The main purpose is to collect related Thorns into a group. E.g. we have two Thorns which provide functions that help to solve an Einstein equation and we have three Thorns which contain different methods to interpolate between results. We would group these Thorns into two arrangements, namely CactusEinstein and CactusInterpolate. All Thorns provides by the toolkit are contained in such arrangements. In fact these arrangements are purely for organizational purposes. They make the life easier for developers which have only to select the right arrangement and not every Thorn which maybe needed for their simulation.

3.3.2 Driver Thorns

The driver Thorn is the most important Thorn if you want to write grid enabled applications. For any application which should use grid variables you need a driver Thorn. As we have already seen the Flesh knows about every grid variable in the system and holds them in a database for easy access. But the Flesh does not create them. This is the job of the driver Thorn. The Thorn allocates storage for grid variables. This is mostly done by reserving the needed memory but depends on the used driver. Also the driver must handle the distribution of the grid variables between different processors and the synchronization of ghostzones. Only one driver Thorn can be active at a time!

Currently only one working implementation of a driver Thorn exists. We will now describe how the communication of grid variables is handled in this implementation, named CactusPUGH. CactusPUGH uses MPI for communication. MPI means Message Passing Interface and is a specification for message passing libraries [9]. Actually different implementations of MPI are available. You have to choose at compile time which implementation CactusPUGH should use and if needed you must provide the libraries of the chosen implementation. Libraries are available for different platforms and different paradigms e.g. parallel computers, clusters and heterogeneous networks. In combination with Cactus the most widely used is MPICH, a portable MPI implementation mostly used on UNIX [10]. For other implementation, look at the links provided by [1]. The exchange of ghostzones and other variables is done through MPI calls.

The actual parallel driver can use whatever method it likes to decompose the grid across processors and exchange ghost zone information - each Thorn is presented with a standard interface, independent of the driver. The big advantage of Cactus is that Thorns can be easily swapped. That means if someday another driver implementation becomes available your application can use it without any changes. Imaginable are driver implementation through shared memory for fast communication if an application runs on a parallel computer with shared memory but without native MPI implementation. (Some supercomputers have a native extremely fast MPI implementation build in). Also the use of other middleware frameworks like CORBA is possible. This would open a whole new world for Cactus applications without changes at codelevel[4].

3.3.3 Toolkit Thorns

As the name suggests these Thorns provide tool functions which your application can use out of the box. The Cactus framework comes with a rich set of toolkit Thorns which provide a wide range of functions. The most important base toolkit is CactusBase it provides coordinate systems, interpolation, time simple I/O functions and boundary condition functions. For further calculations the most important toolkits are CactusEinstein which contains many functions to solve problems related to the Einstein equations, CactusElliptic which provides elliptic solvers and CactusWave for finding solutions to wave equations. For advanced IO, Thorns for output formats like HDF5 [11], IEEE, JPEG and others exits, too.

Additional Thorns e.g. simple webserver for remote steering of simulation, test and utilities are also included in the Cactus framework.

3.3.4 Application Thorns

Application Thorns contain the final code for solving a problem or realizing a simulation. Build on toolkit and driver Thorn the application Thorn needs nothing to know about how the communication works or how a special elliptic equation is solved. The Thorn writer can concentrate on the scientific problem which he wants to solve. Also the application Thorns provide the entry point for the simulation. Only these Thorns are executable.

3.4 Visualization

There are several visualization packages already existing for Cactus e.g. Xgraph, Gnuplot, LCA Vision, OpenDX etc. Depending on the kind of visualization a distinct I/O-Thorn is needed.

4 Cactus Configuration Language

Each Thorn provides three specification files written in the Cactus Configuration Language (CCL). These files designate which implementation the Thorn provides and the variables that the Thorn expects to be available, describe the parameters it uses, and specify routines from the Thorn which must be called.

To get a detailed view of the following three configuration files please read the appendix.

Interface.ccl

The interface.ccl consists of three parts.

1. A header block, which is defining the relationships between several Thorns
2. A block series, listing all global variables.
3. Details of interactions with other Thorns

Param.ccl

The param.ccl consists of a list of parameter object specification items (OSI), which are giving the type and the range of parameter separated by optional parameter data scoping items (DSI). The DSI detail the access to the parameter.

Schedule.ccl

The schedule.ccl consist of:

1. Assignment statements, which switch on storage for grid variables while program execution.
2. Schedule blocks, which schedule subroutines from a Thorn to be called at specific time while program execution in a given manner.
3. Conditional Statements, which allow assignment statements and schedule blocks to be processed depending on parameter values.

5 Bridging Cactus and Globus

Why do we want to do this?

The problem is that Cactus does not provide any functions for distribution of the binary executable, easy access to produced data or dynamic adding or removing resources like machines. Using Globus allows using many connected workstation instead of primarily supercomputer for calculations. For more information see [12]. If we use Cactus and Globus in conjunction user authentication can be done thorough the generic security services (GSS). We could transfer executables and parameter files used by Cactus application with help of the Globus access to secondary storage (GASS) manager and finally start our jobs through the Globus resource allocation & management (GRAM) tool. After our simulation is finished we can easily access the generated data using the GASS part of Globus again. As already mentioned the CactusPUGH driver can use any MPI implementation available, so it is possible to use the provided Nexus MPI to gain more advantages of Globus.

As we can see connecting Cactus to Globus offers many possibilities to make distribution and data gathering in a distributed environment much easier that with Cactus alone. Also it needs a considerable amount of work to setup a running Cactus and Globus[2].

6 Conclusion

As in other areas of computer science the influence and usage of middleware is growing. We think that Cactus is such a middleware product allowing scientist to concentrate on the problem of research instead of spending most of their time with task such as implementing the capabilities to distribute the simulation among several machines or ensuring platform independence. Also Cactus is much more than only a library. It offers many futures and tools which make scientist life easier such as the possibility for a wide range of data visualization. Another advantage of Cactus is, especially for people from the physical science area, that it can be programmed in

FORTRAN. In the fast changing environment of grid computing, an architecture is needed to flexibly support not yet know changes or new paradigms so that simulations can be ported without many changes to new infrastructures. Cactus fulfils this criterion through the abstraction it introduces between the layer of communication/distribution and the layer of computation/simulation.

A major disadvantage of Cactus is that you need at least a degree in physics or study physics to find/create usefully applications with Cactus. One reason for that is that most of the computational toolkits are very specific to small areas of physics [7].

References

[1] The Cactus homepage www.cactuscode.org
[2] Allen, Seidel, 99: The Cactus Computational Toolkit HPDC Presentation http://www.cactuscode.org/tutorials.html
[3] Cactus Team: Cactus Framework I-III Presentation
[4] Goodale, Allen, Radke, Seidel, Shalf, 2003: From the Laptop to the Grid
[5] National Center for Supercomputing Applications http://www.ncsa.uiuc.edu
[6] INRG @ Albert Einstein Institute http://jean-luc.aei-potsdam.de
[7] Talbot,Zhou,Higgins, 2000: Review of the Cactus framework http://sdcd.gsfc.nasa.gov/ESS/esmf_tasc/Files/Cactus_b.html
[8] CactusUserGuide http://www.cactuscode.org/guides/stable/UsersGuide/UserGuideHTML/
[9] The Message Passing Interface Standard http://www-unix.mcs.anl.gov/mpi/
[10] MPI Portable Implementation http://www-unix.mcs.anl.gov/mpi/mpich/
[11] HDF5 data format http://hdf.ncsa.uiuc.edu/HDF5/
[12] The Globus Toolkit http://www.globus.org/

Appendix

```
implements: <implementation>
[inherits: <implementation>, <implementation>]
[friend: <implementation>, <implementation>]

[<access>:]
<data_type><group_name>
[ [<number>] [TYPE=<group_type>] [DIM=<dim>]
  [TIMELEVELS=<num>]      [SIZE=<size in each
direction>]

[DISTRIB=<distribution_type>][GHOSTSIZE=<ghostsize>]
   [STAGGER=<stagger-specification>] [TAGS=<string>]
      {  [<variable_name>[,]<variable_name>]
      }
   [„<group_description>“]
]
```

Figure 3 The interface.ccl

```
<access>:

[EXTENDS|USES] <parameter_type>
[ [<integer]>] <parameter name> "<parameter
description>"
  [AS <alias>]
  [STEERABLE=<NEVER|ALWAYS|RECOVER>]
  [ACCUMULATOR=<expression>]
  [ACCUMULATOR-BASE=<parameter-name>]
  {
   <PARAMETER_VALUES>
  } <default value>
```

Figure 4 The param.ccl

```
schedule [GROUP] <function name|group name> AT|IN
<time> \[BEFORE|AFTER <function name>] [WHILE <variable>]
[AS <alias>]
{ [LANG: <language>]
  [STORAGE: <group>[timelevels],<group>[timelevels]...]
  [TRIGGER: <group>,<group>...]
  [SYNCHRONISE: <group>,<group>...]
  [OPTIONS: <option>,<option>...]
} "Description of function"
```

Figure 5 The schedule.ccl

High Performance Scheduler mit Maui/PBS

Ole Weidner, Jörg Schummer, Benedikt Meuthrath

{ole.weidner | joerg.schummer | benedikt.meuthrath}@student.hpi.uni-potsdam.de

Dieses Dokument soll einen Überblick über die Verwendung des Stapelverarbeitungssystems PBS in Verbindung mit dem Maui Scheduler in High Performance Computing Umgebungen geben. Des Weiteren wird eine Einordnung in den Gridcomputing Kontext diskutiert.

1 Begriffsdefinitionen

Um die in diesem Dokument verwendeten Terminologien eindeutig festzulegen, zuerst ein kurzes Glossar:

Cluster Ein Cluster ist eine Einheit von homogenen oder heterogenen Computersystemen, die über ein LAN miteinander verbunden sind und deren Ressourcen an einer zentralen Stelle verwaltet werden.

Job Ein Job bezeichnet eine auf einem Cluster ausführbare Einheit. Ein Job besteht aus dem eigentlichen auszuführenden Programm und eventuell dafür benötigten Daten sowie einem Jobscript.

Jobscript Ein Jobscript beinhaltet Ausführungsregeln für ein Programm sowie Informationen über die Ressourcen (CPU, Speicher, Swap, installierte Software, usw.), die für die Ausführung benötigt werden.

```
#!/bin/csh
#Demo JobScript for PBS
#PBS -l nodes=4:ppn=2
#PBS -l mem=2gb
#PBS -A user
#PBS -M user@foo.bar

ncups=`wc -l $PBS_NODEFILE | \
nawk ´{print $i}´`

/opt/SUNWhpc/bin/mprun -np $ncups \
 -Mf  $PBS_NODEFILE  /stburks/sun.A.4
\
> /stburks/sun.A.4.txt
```

Listing 1: Beispiel eines Jobscripts für PBS

Job Scheduler Der Job Scheduler entscheidet anhand der Daten, die der Resource Manager zur Verfügung stellt, wann welcher Job auf dem Stapelverarbeitungssystem zur Ausführung gebracht werden soll.

Resource Manager Die Aufgabe des Resource Managers ist es, Informationen über alle Ressourcen und alle Jobs im System in einer Datenbank zu verwalten. Diese Informationen werden hauptsächlich vom Scheduler verwendet um Scheduling-Entscheidungen zu treffen.

Reservierung Reservierungen zählen zu den Kernkomponenten in einer flexiblen, regelbasierten Scheduling-Umgebung. Reservierungen sind jobspezifisch und

bestehen aus einer Anzahl Ressourcen sowie einem Zeitrahmen, in dem die Ressourcen benötigt werden.

2 Das Portable Batch System

2.1 Einführung

Um Cluster- oder auch Gridscheduling zu betreiben, werden Informationen über das System und über die Jobs benötigt, die das System nutzen. Anhand dieser Daten können die Schedulingalgorithmen entscheiden, wie das System am effektivsten betrieben werden kann. In Clusterumgebungen werden Resource Manager eingesetzt um diese Daten aus dem System zu bekommen. Im Folgenden soll das Portable Batch System näher betrachtet werden, das unter anderem die Funktion eines Resource Managers erfüllen kann.

2.2 Evolution

Das Portable Batch System (PBS) ist ein dem POSIX Standard 1003.2d [1] vollständig genügendes Stapelverarbeitungssystem. Es wurde in den Jahren 1993 bis 1997 am NASA Ames Research Center [2] für große parallele SMP-Systeme mit dem Ziel entwickelt, das bis dahin verwendete Network Queueing System (NQS) zu ersetzten. Eines der wichtigsten Designziele war dabei, mit der Software ein möglichst breites Spektrum an Computerarchitekturen abzudecken, so dass sie sowohl auf lose gekoppelten Clustern heterogener Workstations als auch auf massiv parallelen Supercomputern zum Einsatz kommen kann. Trotz dieser breiten Abdeckung liegen die Stärken von PBS eher beim Einsatz auf SMP-Systemen als auf Clustern. Zum Beispiel ist die Resource Manager Komponente des Systems zwar in der Lage mehrere Prozessoren für einen parallelen Job zu reservieren, das Starten des Jobs und das Ausführen von Jobscripten für diesen Job findet jedoch nur auf einem einzigen Clusterknoten statt.

Aktuell wird PBS von der Firma Altair Grid Technologies, LLC [3] mit dem Ziel weiterentwickelt, die Unterstützung für den Clustereinsatz weiter voranzutreiben. Altair stellt zwei Versionen der Software zur Verfügung: die OpenSource Version OpenPBS und die kommerzielle Version PBS Pro. Die Hauptunterschiede bestehen im fehlenden Support, einer geringeren Fehlertoleranz und eingeschränktem Job Scheduling in der OpenSource Variante. Eine dritte Version existiert zurzeit als Betaversion, die von dem Unternehmen Cluster Resources Inc. [4] als Scalable PBS (OpenSource) zur Verfügung gestellt wird. Diese Version versucht die Nachteile von OpenPBS gegenüber PBS Pro auszugleichen und implementiert darüber hinaus eine erweiterte Scheduler-Schnittstelle, die es ermöglichen soll, den Scheduler mit zusätzlichen und exakteren Informationen über Ressourcen und Jobs zu versorgen. Dieses und die Tatsache, dass die Entwicklung an Scalable PBS sehr aktiv zu sein scheint, macht es zu dem zurzeit wohl interessantem OpenSource Stapelverarbeitungssystem.

2.3 Architektur

Bei dem Design von PBS hat man sich von Anfang an auf einen verteilten Aufbau geeinigt, der sich aus den drei Komponenten Job Server, Job Scheduler und Job Executor zusammensetzt. Da jede dieser Komponenten als eigenständiges Programm realisiert ist und für die Kommunikation zwischen diesen Programmen wohldefinierte Schnittstellen und Protokolle existieren, ermöglicht diese Architektur den leichten Austausch einzelner Komponenten. Dieses wird in einem späteren Abschnitt, der Integration des Maui Schedulers in das PBS System, noch genauer betrachtet.

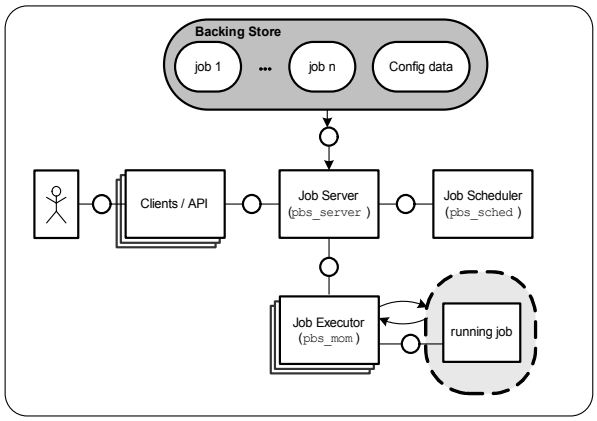


Abbildung 1: Architektur des Portable Batch Systems

Die zentrale Komponente des Systems bildet der Job Server (pbs_server). Er dient hauptsächlich als Kommunikationsknoten zwischen den Job Executoren, dem Job Scheduler und den Clients. Um diese mit den benötigten Informationen über Jobs und Ressourcen zu versorgen, hält und verwaltet der Server eine Job- und eine Ressourcenliste, die er mit Hilfe der einzelnen Job Executoren aktualisiert. In der Serverkomponente ist keinerlei Scheduling-Logik implementiert – das Anstoßen eines neuen Jobs auf einem der Job Executoren geschieht nur auf Anweisung des Job Schedulers. Weiterhin ist der Job Server für die Fehlertoleranz im Cluster zuständig. Dies geschieht durch redundante Speicherung der Jobs auf einem Backing Store, bis diese vollständig abgearbeitet sind. Scheitert die Ausführung eines bestimmten Jobs (z.B. durch Ausfall einer oder mehrerer Clusterknoten), kann diese wieder aufgenommen werden, ohne dass der Benutzer seinen Job neu an das System übertragen muss. In einer PBS Umgebung läuft der Job Server in der Regel nur auf einem Knoten. Dieser Knoten ist der einzige, mit dem die Benutzer des Clusters interagieren können.

Die Scheduling-Logik selbst ist in der Komponente Job Scheduler (pbs_sched) implementiert. Die von dem Scheduling-Algorithmus benötigten Daten bekommt der Job Scheduler von dem Job Server. Ist eine Scheduling-Entscheidung aufgrund dieser Daten getroffen, wird der Job Server angewiesen, diese auszuführen. Der Job Scheduler läuft genau wie der Job Server in der Regel auch nur auf einem Knoten – oft auf demselben wie der Job Server.

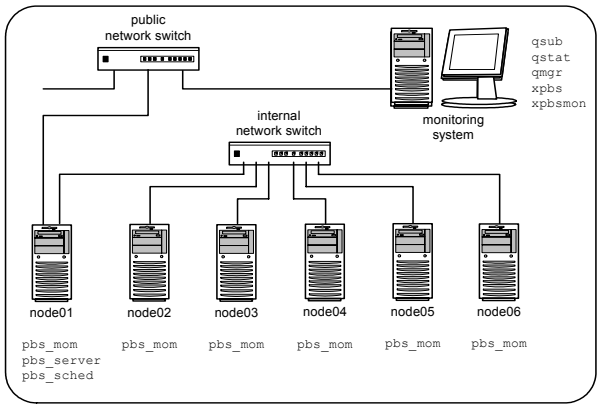


Abbildung 2: Beispielkonfiguration eines Clusters mit PBS

Die Abarbeitung der Jobs wird von den Job Executoren (pbs_mom) oder auch MOMs (Machine Orientated Miniserver) übernommen, die auf jedem Rechenknoten des Clusters ausgeführt werden. Ein Job Executor führt eine vom Job Server übermittelte Kopie eines Jobs

aus und gibt auf Anfrage Informationen über dessen aktuellen Status und die Ressourcenauslastung des Rechenknotens an den Job Server zurück. Da ein Einsatz von PBS in heterogenen Clusterumgebungen möglich sein soll, besteht eine hohe Plattformverfügbarkeit für die Job Executoren. Neben Versionen für x86 Architekturen unter Linux oder FreeBSD existieren Implementierungen für CRAY Systeme unter UniCos 8, 9, 10 oder MK2, IBM 590 und SP Systeme unter AIX 4.x oder 5L, SGI Systeme unter IRIX 5.x oder 6.x und SUN Sparc Systeme unter SunOS 4.1 oder Solaris. In den jeweiligen Implementierungen wird versucht, plattformspezifische Eigenschaften auszunutzen.

In der Abbildung 2 ist eine typische Beispiel-Konfiguration für einen Cluster mit PBS dargestellt. Auf *node01*, die über das öffentliche Netzwerk erreicht werden kann, werden der Job Server und der Job Scheduler ausgeführt. Die Abarbeitung der Jobs geschieht auf den dedizierten Rechenknoten *node2* bis *node6*, die über ein internes Netzwerk mit *node1* verbunden, von außerhalb aber nicht zu erreichen sind.

2.4 Benutzerinteraktion mit PBS

Um die Funktionalität eines Stapelverarbeitungssystems nutzen zu können, muss es Schnittstellen geben, über die der Benutzer mit dem System interagieren kann, d.h. über die Jobs an das System übermittelt werden können, über die Informationen über das System abgefragt werden können und über die das System verwaltet und administriert werden kann. Dazu stehen zwei Möglichkeiten zur Verfügung: zum einen gibt es eine Reihe von Benutzerschnittstellen in Form von Nutzerprogrammen und zum anderen eine Programmierschnittstelle (API).

Die Nutzerprogramme umfassen Werkzeuge zur Jobverwaltung, zum Einholen von Statusinformationen und zur Verwaltung und Konfiguration des Systems. Einige Befehle sind in Tabelle 1 aufgelistet. Für eine vollständige Liste sei auf die UNIX-man Seiten von PBS verwiesen.

Als graphische Benutzerschnittstelle stehen dem Benutzer die Programme xpbs und xpbsmon zur Verfügung. Beide sind unter dem X-Window System lauffähig. xpbsmon stellt den Cluster grafisch dar und stellt Statusinformationen zu jedem Rechenknoten zur Verfügung. xpbs ermöglicht es dem Benutzer, Jobscripte „zusammenzuklicken“, die ansonsten „von Hand“ mit einem Texteditor erstellt werden müssen, sowie die komfortable Übermittlung von Jobs an das System. Dies ist zwar sehr hilfreich, bei komplexeren Scripten ist allerdings eine manuelle Konfiguration unumgänglich, da xpbs nicht alle Konfigurations-Parameter unterstützt.

Programm	Funktion
`qalter`	ändert Attribute eines Jobs
`qdel`	löscht einen Job
`qhold`	hält einen Job an
`qrerun`	führt einen Job erneut auf
`qstatus`	gibt Jobinformationen aus
`qsub`	schickt einen Job an den Server
`qmgr`	stellt eine Verwaltungsschnittstelle für PBS zur Verfügung

Zusätzlich zu den zum Umfang von PBS gehörenden Clientanwendungen, gibt es noch einige Werkzeuge Dritter, wie zum Beispiel das an der University of Alberta entwickelte PBSWeb [5], das eine ähnliche Funktionalität wie xpbs über eine HTML Schnittstelle zur Verfügung stellt.

Als API steht die C-Bibliothek libpbs.a zur Verfügung, die auch von dem Standard PBS Nutzerprogrammen genutzt wird. Die Bibliothek stellt Schnittstellen für Jobverwaltung, Ressourcenverwaltung und Scheduling zur Verfügung. Die Prototypen der Bibliotheksfunktionen sind in den Headerdateien pbs_ifl.h und pbs_error.h definiert. Außerdem existieren Wrapper für die Sprachen Python, TCL und BaSL[1], einer PBS-eigenen Erweiterung der Sprache C, mit der sich sehr einfach Scheduler beschreiben lassen. Eine Beispielanwendung, die nichts weiter macht, als zu versuchen, sich zu einem PBS Job Server zu verbinden, lässt sich mit Hilfe der PBS API in C folgendermaßen realisieren:

```
#include pbs_ifl.h
  #include pbs_error.h

  main( int argc, char* argv[] ) {
    int connect;

    connect=cnt2server("localhost");
    if( connect <= 0 ) {
      fprintf( stderr, "connect to %s
\
        failed: (error=%d)",
        pbs_server, pbs_errno);
    } else {
      fprintf( stderr, "connected!" );
    }
    return 0;
  }
```

Listing 2: Einfaches Beispiel zur Benutzung von libpbs.a

Tabelle 1: Ausgewählte PBS Nutzerprogramme

Die Bibliothek abstrahiert von den Schnittstellen und Protokollen mit denen PBS intern arbeitet, in dem „low-level"-Aufrufe, wie zum Beispiel die Benutzung von Sockets bei der Verbindung zu einem Server in Funktionen gekapselt werden und somit dem Anwender verborgen bleiben.

2.5 Integration des MAUI Schedulers

Da PBS ein vollständiges Stapelverarbeitungssystem ist, bringt es auch wie bereits oben erwähnt seinen eigenen Scheduler mit – die Job Scheduler Komponente pbs_sched. Der Standardscheduler der OpenSource Version ist ein modifizierter FIFO Scheduler. Im Vergleich zu anderen Schedulern wie zum Beispiel Maui (siehe Abschnitt III.), sind in dem FIFO Scheduler aktuell geforderte Algorithmen, wie zum Beispiel FairShare oder Backfilling nicht implementiert. Wenn man nun ein Clustersystem möglichst effizient nutzen möchte, ohne die kommerzielle PBS Pro Version einzusetzen, empfiehlt es sich also, nur den FIFO Scheduler auszutauschen. Dies ist aufgrund der beschriebenen Komponentenstruktur von PBS sehr einfach möglich. Solange der neue Scheduler die Schnittstellenspezifikation zum Job Server einhält, kann er anstatt des Standardschedulers ausgeführt werden, ohne dass dies den anderen Komponenten des PBS Systems bekannt sein muss. In der Praxis sieht es tatsächlich so aus, dass anstatt des FIFO Scheduler Prozesses pbs_sched einfach der Maui Prozess maui gestartet wird. Auf der Seite von PBS besteht kein Konfigurationsbedarf.

Der Maui Scheduler ist in der Lage, erweiterte Scheduling-Regeln in den Job Scripten zu verarbeiten. Daher bringt er eine eigene Sammlung an Clientprogrammen zur Jobverwaltung mit, die benutzt werden müssen, falls man auf diesen erweiterten Regelsatz zurückgreifen möchte. Falls dies nicht der Fall ist, können weiterhin die Clientprogramme von PBS benutzt werden.

[1] BaSL: Baisc Scheduling Language

3 Der Maui Scheduler

3.1 Einführung

Der Maui Scheduler [6] ist ein Job-Scheduler für Cluster und Supercomputer. Obwohl er ein Job-Scheduler ist, und somit einen hohen Job-Durchsatz fokussiert, zielt er ebenfalls darauf ab, die Ressourcenauslastung des zugrunde liegenden Systems zu optimieren.

Zu Mauis Hauptmerkmalen zählen die Unterstützung von Ressourcen-Reservierungen, die bei vergleichsweise einfachen Stapelverarbeitungssystemen (siehe Abschnitt 2: „Das Portable Batch System“) nicht vorhanden ist, sowie die hochgradige Konfigurierbarkeit, die es erlaubt Maui an die unterschiedlichsten Systemanforderungen anzupassen.

Maui ist komplett in C geschrieben und dementsprechend auf (fast) allen Unix-Derivaten (z.B. Linux, AIX, OSF/Tru-64, Solaris, HP-UX, IRIX, FreeBSD) lauffähig.

3.2 Evolution

Die Entwicklung des Maui Scheduler begann 1995 an der Brigham Young University als Teil einer Master-Arbeit auf dem Gebiet Scheduling-Optimierung. Zum praktischen Einsatz kam er erstmals 1996 am Maui High Performance Computing Center (MHPCC), welches ihm dann auch zu seinem Namen verhalf. Inzwischen ist er sehr bekannt und weit verbreitet; so wurde er z.B. schon im Jahr 2000 vom National Center for Supercomputing Applications (NCSA) auf einem Origin 2000 Cluster mit insgesamt 1500 Prozessoren eingesetzt.

Maui wurde lange Zeit als Sourceforge-Projekt geführt (Maui ist OpenSource). Die Entwicklung wurde dann jedoch von Cluster Resources Inc., einer Firma die unter anderem Maui für spezielle Umgebungen anpasst und auch Schulungen zu Maui anbietet, fortgesetzt und findet bis heute dort statt.

3.3 Maui Molokini Edition

Wie auch Maui ist Maui Molokini Edition (MME) [7] ein Job-Scheduler für High Performance Computing Systeme. Obwohl der Name es suggeriert, gibt es keine nähere Verbindung zwischen den beiden Schedulern.

MME wird auf Mauis früherer Sourceforge-Entwicklungsseite von einem Team entwickelt, das komplett unabhängig vom Maui-Entwicklungsteam arbeitet; tragischerweise wird MME dort auch oft nur „The Maui Scheduler“ genannt, was die Verwirrung um einen möglichen tieferen Zusammenhang zwischen den beiden Projekten komplettiert.

Zu den Hauptunterschieden zwischen MME und Maui zählt, dass MME komplett in Java geschrieben ist, während Maui in C implementiert wurde. Des Weiteren wird der viel jüngere MME momentan nur auf einem einzigen Cluster am MHPCC eingesetzt, während Maui bereits sehr stark verbreitet ist.

3.4 Architektur

In einem jobverarbeitenden System stellt Maui de facto nur den Scheduler dar, d.h. nur diejenige Komponente, die aus einer oder mehreren Job-Queues entscheidet, welcher Job als nächstes ausgeführt wird und welche Ressourcen er dazu benutzen darf.

Zur Jobverarbeitung wird daher noch ein Resource-Management-System (RMS) benötigt. Zum einen erhält Maui von diesem Informationen über den Zustand der Ressourcen, wie z.B. die Prozessorauslastung oder die verfügbare Festplattenkapazität. Zu den Aufgaben einer Resource-Management-Umgebung gehört es zum anderen aber auch, verschiedene Job-Queues bereitzustellen, in welche die Endbenutzer Jobs einfügen können. Maui erhält zwar Informationen über die Queues und führt Statistiken über deren Benutzung, aber verwaltet werden diese dennoch vom RMS. Zuletzt ist es auch noch Aufgabe des RMS die Jobs, welche von Maui zur Ausführung ausgewählt wurden, auf den entsprechenden Knoten des Systems wirklich auszuführen.

Die Aussage, dass Maui auf allen Unix-Derivaten lauffähig ist, ist demnach mit Vorsicht zu genießen, da tatsächlich das Resource-Management-System auf den dem Cluster oder Supercomputer zugrunde liegenden Knoten lauffähig sein muss.

Derzeit bietet Maui volle Unterstützung für die folgenden Resource-Management-APIs bzw. Resource-Management-Systeme:

- PBS Scheduling API - OpenPBS and PBSPro
- LoadLeveler Scheduling API - LoadLeveler (IBM)
- LSF Scheduling API - LSF (Platform)
- Wiki FlatText Scheduling API (Wiki)

Teilweise oder nur mit Einschränkungen unterstützt werden:

- SGE Scheduling API - Grid Engine (Sun)
- BProc Scheduling API - BProc (Scyld)
- SSS XML Scheduling API

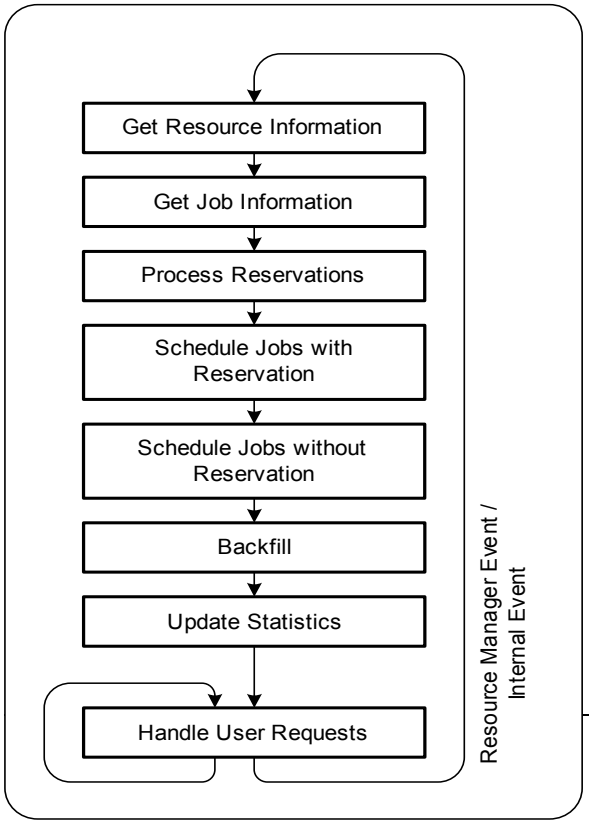


Abbildung 3: Der Schedulingablauf von Maui

Maui unterstützt aber nicht nur jedes dieser RMS „für sich“, sondern lässt auch beliebige Kombinationen derselben (z.B. auf heterogenen Cluster-Umgebungen) zu.

3.5 Der Scheduling-Zyklus

Maui beginnt seinen Scheduling-Zyklus [8] damit, die Resource-Manager zu kontaktieren um den Status aller Ressourcen zu erfragen.

Danach werden Reservierungen aktualisiert, d.h. es werden neue Reservierungen eingefügt oder bestehende gelöscht. Anschließend werden Jobs, für deren Ressourcenanforderungen

Reservierungen vorgenommen wurden, gestartet.

Anschließend werden alle Jobs, für die keine Reservierung vorgenommen wurde, verarbeitet. Dies beinhaltet das Erstellen einer Liste aller lauffähigen Jobs und die Priorisierung dieser Jobs in Form einer Sortierung ebendieser Liste. Ob ein Job lauffähig ist, wird anhand seiner Ressourcenanforderungen und der Verfügbarkeit der Ressourcen entschieden. Nachdem die Priorisierung vorgenommen wurde, werden die Jobs in der Reihenfolge ihrer Priorität gestartet.

Falls Maui entsprechend konfiguriert wurde, wird nach dem „normalen“ Starten der Jobs das so genannte Backfilling durchgeführt. Backfilling bezeichnet ein außerplanmäßiges Starten von Jobs zur besseren Auslastung der Ressourcen (siehe H: „Der Backfill-Algorithmus“).

Dann werden verschiedene Statistiken, wie z. B. über die Auslastung bestimmter Ressourcen, aktualisiert. Abschließend werden Benutzeranfragen bearbeitet; das beinhaltet z. B. das Abfragen ebensolcher Statistiken. Dies wird solange durchgeführt, bis ein Resource-Manager-Event oder ein interner Event eintreten. Ein Resource-Manager-Event kann z. B. das Einreichen eines neuen Jobs, die Fertigstellung eines Jobs, oder das Hinzufügen neuer Ressourcen darstellen. Ein interner Event kann z.B. eine Schedule-Anfrage sein, wie sie mittels des *schedctl*-Kommandos gestellt werden kann.

3.6 Job-Priorisierung

Job-Prioritäten in Maui werden weder vom Benutzer, welcher den Job gestartet hat, noch vom Administrator direkt als Zahlenwerte angegeben; stattdessen setzt sich die endgültige Priorität indirekt aus 6 Komponenten zusammen. Dies sind “Requested Resources”, “FairShare”, “Target”, “Bypass”, “Service” und eine direkte Priorität. Zu letzterer ist zu erwähnen, dass sie nur als eine dieser 6 Komponenten gewertet wird und die anderen 5 Komponenten keinesfalls überschreibt. Nur falls die anderen 5 Komponenten für zwei bestimmte Jobs eine gleiche Prioritätsstufe ergäben, bestimmt diese Komponente die Priorität direkt.

- Die Komponente Requested Resources erlaubt es, Jobs aufgrund ihrer Ressourcenanforderungen unterschiedlich zu priorisieren.
- FairShare lässt eine differenzierte Priorisierung von Jobs unter Berücksichtigung der Benutzung von Ressourcen in der Vergangenheit zu.
- Die Komponente Target erlaubt es, Jobs zu bevorzugen, die schon relativ lange Zeit in einer Queue verbringen.
- Mittels der Komponente Bypass können solche Jobs bevorzugt werden, die schon oft von Jobs, die durch Backfilling gestartet wurden, „überholt“ wurden.
- Service gestattet es, die Priorität bestimmter Jobs, die ansonsten sehr weit unten in der Prioritätsliste stünden, anzuheben.

Wie bereits erwähnt, ist als untergeordnete Differenzierung noch die Angabe einer direkten Priorität möglich. Dies ist z.B. dann sinnvoll, wenn derselbe Benutzer zur selben Zeit 2 sehr ähnliche Jobs startet, aber die Reihenfolge der Ausführung dennoch beeinflussen möchte.

Die Berechnung der endgültigen Priorität eines Jobs erfolgt dann anhand der ermittelten Werte der Prioritäts-Komponenten für den Job sowie einer Gewichtung dieser Komponenten. Diese 6 Gewichte werden einmalig in der Maui-Konfiguration festgelegt und sind für alle Jobs gültig. Es ergibt sich folgende Formel:

```
priority =
  resourceweight   * resourcefactor  +
  fairshareweight  * fairsharefactor +
  targetweight     * targetfactor    +
  bypassweight     * bypassfactor    +
  serviceweight    * servicefactor   +
  directspecweight * directspecfactor
```

Hier sind die auf der linken Seite stehenden *weight*-Bestandteile die in der Maui-Konfiguration festgelegten Werte und die auf der rechten Seite stehenden *factor*-Bestandteile die jobspezifisch errechneten Werte.

3.7 Das FairShare-Konzept

FairShare stellt ein Konzept dar, das es erlaubt, Ressourcen zwischen bestimmten Gruppen von Jobs in einer fairen Art und Weise aufzuteilen. In Maui wird dieses Konzept als eine der 6 Prioritäts-Komponenten von Jobs realisiert; im Gegensatz zur Prioritäts-Komponente ‚Requested Resources' werden aber zusätzlich zu den aktuellen Ressourcen-Anforderungen des Jobs einer bestimmten Gruppe auch zurückliegende Ressourcen-Anforderungen von Jobs derselben Gruppe berücksichtigt.

Zur Anwendung des Konzepts müssen zumindest zwei Angaben gemacht werden; zum einen muss die Ressource, die zwischen den Jobs aufgeteilt werden, und deren ‚Konsum' von bestimmten Job-Gruppen somit aufgezeichnet und verfolgt werden soll, spezifiziert werden und zum anderen muss eine Gruppierung der Jobs erfolgen. Diese Gruppierung kann z.B. nach Job-Besitzer oder nach Job-Queue erfolgen.

Wie bereits erwähnt, ist ein Hauptmerkmal des FairShare-Konzepts die Berücksichtigung des Ressourcen-Konsums über einen Zeitraum. Hierbei ist es üblich, Ressourcen-Anforderungen, die länger zurückliegen, weniger stark zu gewichten als solche, die weniger lange zurückliegen. Für diesen Anwendungsfall muss noch ein Faktor angegeben werden, um den zurückliegende Aufzeichnungen von Ressourcen-Konsum an Gewicht verlieren.

Die Umsetzung des FairShare-Konzepts erfolgt dann, wie bei allen anderen Prioritätskomponenten auch, durch Anpassung der Priorität der Jobs. Dies wird so realisiert, dass die Priorität von Jobs, welche mehr Ressourcen konsumieren, als es die FairShare-Festlegung vorsieht, herabgesetzt wird. Umgekehrt gilt, dass, falls eine bestimmte Job-Gruppe nur geringen Gebrauch von der angegebenen Ressource macht, die Prioritäten der Jobs dieser Gruppe heraufgesetzt werden.

3.8 Der Backfill-Algorithmus

Backfilling bezeichnet ein außerplanmäßiges Starten von Jobs zur besseren Auslastung der Ressourcen. Außerplanmäßig bedeutet in diesem Fall, dass die vorgenommene Priorisierung der Jobs nur bedingt berücksichtigt wird, d.h. es kann sein, dass Jobs mit niederer Priorität vor Jobs mit vergleichsweise hoher Priorität ausgeführt werden.

Zwei Umstände sollen als Vorüberlegung zum Backfill-Algorithmus dienen:

1. Jeder Job in Maui hat eine Start-Zeit.
2. Jeder Job in Maui hat ein sogenanntes Wallclock-Limit.

Das Wallclock-Limit gibt die Zeit an, die der Job maximal für seine Ausführung beanspruchen darf, bevor er von Maui beendet wird.

Bei der Abarbeitung der Prioritätsliste, die Maui innerhalb seines Scheduling-Zyklus erstellt, wird die Liste der Reihe nach durchgegangen, und solange Maui auf lauffähige Jobs trifft, werden diese ausgeführt. Eine strikte Einhaltung der

Prioritätsliste würde voraussetzen, dass, sobald ein Job angetroffen wird, der nicht lauffähig ist, an diesem Punkt das Starten von Jobs angehalten wird. Dass ein Job nicht lauffähig ist, kann dann der Fall sein, wenn die angeforderten Ressourcen durch einen gerade gestarteten, höher priorisierten Job blockiert werden.

Eine solche strikte Einhaltung der Job-Prioritäten scheint zwar fair zu sein, da niemals ein Job mit niederer Priorität vor einem mit höherer Priorität gestartet wird; es wird aber schnell klar, dass die Ressourcen wesentlich besser ausgelastet werden könnten. Ebendies wird durch den Backfill-Algorithmus realisiert.

Wenn Maui nun auf einen Job in der Prioritätsliste stößt, dessen angeforderte Ressourcen blockiert werden, so kann zumindest eine potentielle Start-Zeit des Jobs ermittelt werden, da die Start-Zeit und das Wallclock-Limit des blockierenden Jobs bekannt sind. Wenn nun ein nieder priorisierter Job, der aufgrund seiner Ressourcenanforderungen jetzt lauffähig ist und diese potentielle Start-Zeit aufgrund seines Wallclock-Limits nicht nach hinten verschieben kann, so wird dieser gestartet.

Folgendes Beispiel-Szenario soll die Vorgehensweise beim Backfilling verdeutlichen. Die Ausgangssituation (Abbildung 4) im System ist die, dass Job A bereits gestartet ist und sein Wallclock-Limit in 3½ Stunden ausläuft. Des Weiteren sind zwei Reservierungen vorgenommen worden.

Die priorisierte Liste von Jobs sieht wie folgt aus:

1) Job B – benötigt 1 Knoten für 1½ Stunden
2) Job C – benötigt 2 Knoten für 2 Stunden
3) Job D – benötigt 1 Knoten für 2½ Stunden

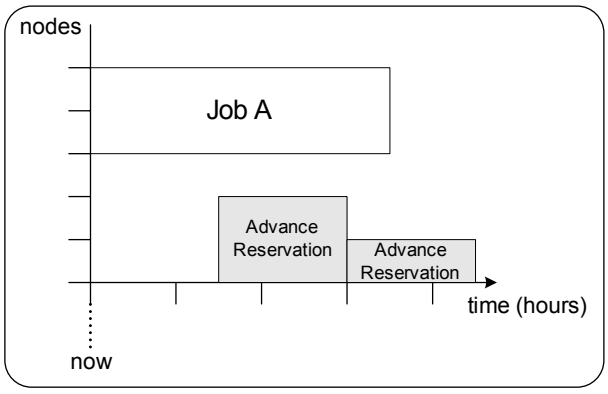


Abbildung 4: Backfilling - Ausgangssituation

Job B wird sofort auf Knoten 1 (der unterste im Diagramm) ausgeführt, da auf diesem in den nächsten 1½ Stunden keine Job-Ausführung vorgesehen ist. Job C kann nun aber nicht sofort ausgeführt werden, da nur Knoten 3 (der 3. von unten im Diagramm) 2 Stunden lang frei ist. Es wird also stattdessen eine Reservierung auf Knoten 2 und 3, die nämlich beide in 3 Stunden wieder verfügbar sind, vorgenommen.

Danach wird mit Job D fortgefahren, und da dieser ohne die Start-Zeit des höher priorisierten Job C zu verschieben – jetzt auf Knoten 3 lauffähig ist, wird er gestartet. Dies führt zu einer Endsituation, wie sie in Abbildung 5 zu sehen ist.

Die Ressourcen werden also insofern besser ausgelastet, als dass ohne den Einsatz von Backfilling Knoten 3 in den nächsten 2½ Stunden nicht verwendet würde.

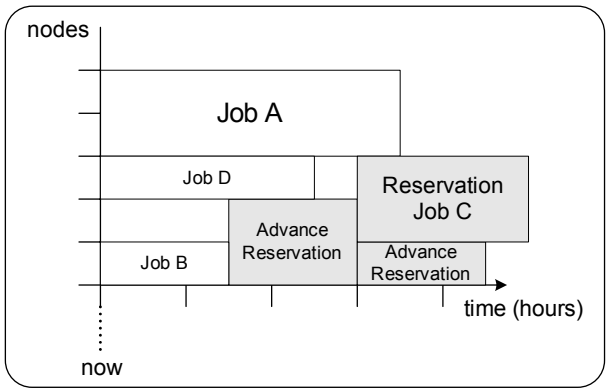


Abbildung 5: Backfilling - Endsituation

Ein Nachteil des Backfill-Algorithmus besteht darin, dass die Wallclock-Limits, die ja der Benutzer, welcher den Job an Maui übergibt, selbst spezifiziert, sehr ungenau sein können. Das heißt, dass sich Jobs, die momentan Ressourcen beanspruchen, schon lange Zeit vor dem Auslaufen ihres Wallclock-Limits beenden. Dies führt dazu, dass die Reservierung für einen Job, der beim Backfilling ‚überholt' wird, eigentlich für einen zu späten Zeitraum angelegt wird, und der höher priorisierte Job doch hätte früher starten können.

Dieser Aspekt ist beim Einsatz von Backfilling auf jeden Fall zu berücksichtigen; jedoch überwiegt in den meisten Fällen die bessere Auslastung der Ressourcen und es wird entschieden, Backfilling zu verwenden.

4 Einordnung in den Grid-Kontext

4.1 Einleitung

Das so genannte High Performance Scheduling wird meist als Synonym für Application Scheduling benutzt. Application Scheduler sind eine Art von Schedulern, die man braucht, wenn man ein heterogenes, weiträumig verteiltest Grid performant betreiben möchte.

Ein Application Scheduler braucht sowohl Resource Scheduling als auch Job Scheduling auf den einzelnen Knoten im Grid, um vernünftige Schedules errechnen und ausführen zu können. Dabei ist ein Application Scheduler (oder auch „Grid Scheduler") eine Art Metascheduler, d.h. er „scheduled" den Einsatz von Schedulern. Ein Beispiel, an das wir uns halten, ist der Metascheduler Silver [9], welcher als Unterbau den Maui Scheduler und das PBS verwenden kann.

Der Silver Scheduler ist zu Teilen implementiert und ist als Demonstration auf der „Supercomputing 2000 Conference" und dem „Grid Forum IV" aufgetreten. Die Beispiele, die zum Silver gegeben werden sind entweder schon implementiert oder sind geplant. Da der Schwerpunkt nicht auf dem Silver Scheduler sondern auf allgemeinen Ansätzen und Konzepten liegen soll, wird an der Stelle nicht weiter differenziert.

Die im Folgenden vorgestellten Ansätze und Konzepte wurden auf diversen Konferenzen vorgestellt und diskutiert [10].

4.2 Application Scheduler

Während Resource Scheduler dahingehend optimiert sind, die vorhandenen Ressourcen optimal auszulasten und Job Scheduler dahingehend, möglichst viele Jobs pro Zeiteinheit zu erledigen (High-Throughput-Scheduler), legt ein Application Scheduler mehr den Schwerpunkt auf die Performanceansprüche einer einzelnen Anwendung, wobei diese unter Umständen vom Programmierer angegeben werden (können). Dabei ist die Aufgabe eines „Application Schedulers" eine Auswahl an zur Verfügung stehenden Ressourcen zu bestimmen, auf denen die einzelnen Tasks der Anwendung ausgeführt werden sollen. Aber sowohl die Verteilung der benötigten Daten auf die Knoten, wo sie gebraucht werden, als auch die Anforderung der Kommunikation zwischen den einzelnen Tasks muss berücksichtig oder sogar selber durchgeführt werden.

4.3 Performanz und Vorhersagen

Die Informationen, welche Ressourcen derzeit zur Verfügung stehen, bekommt der Application Scheduler vom Resource Scheduler, der auf jedem einzelnen Knoten des Grids läuft und dort die Auslastung beobachtet und sie möglichst optimiert.

Zugleich steht dem Application Scheduler ein Vorhersagemodell zur Verfügung, anhand dessen er versucht grobgranular die Auslastung in näherer Zukunft vorherzusagen. Hierbei handelt es sich nicht um Vorhersagen im Millisekundenbereich sondern eher im Bereich von Stunden oder sogar Tagen. Die Vorhersagen stützen sich meist auf Modelle, die aus Beobachtungen über einen längeren Zeitraum erfolgt sind. Ein Beispiel um zu erklären, wie es ungefähr funktioniert: Studenten, die sich die Ressourcen eines Computerlabors teilen, versuchen häufig die am wenigsten ausgelasteten Server für ihre Experimente zu nutzen. Nicht die momentane Auslastung interessiert sie, sondern eine Abschätzung darüber, wie die Auslastung in nächster Zukunft, in der sie ihr Programm ausführen werden, sein wird. Sie benutzten dafür Informationen, die das System anbietet (z.B. unter UNIX „load average“, wie viele Nutzer derzeit eingeloggt sind, wer diese Nutzer sind, welche Programme derzeit laufen, etc) um vorherzusagen wie performant ihr Programm laufen wird.

Um jegliche Art von Vorhersagen die Performanz betreffend machen zu können, braucht man eine Basis an Daten, welche die Auslastung der einzelnen Knoten beschreiben. Da diese Datenbasis nicht statisch vorliegt, was in der Natur eines Grids begründet ist, muss sie dynamisch verwaltet werden, dass heißt es müssen regelmäßig Aktualisierungen vorgenommen werden.

Damit der Grid Scheduler aus diesen Daten die Performanz einer Anwendung errechnen kann, braucht er ein Performanzmodell. Entweder muss der Programmierer dieses Modell angeben oder der Scheduler legt grob für verschiedene Arten von Anwendungen ein Performanzmodell fest. Meist gibt es eine Mischform der Ansätze bei welcher der Programmierer Performanzziele angibt (z.B. maximaler Ressourceverbrauch, Rückkehrzeit der Anwendung, Erreichen von definierten Teilzielen zu einem bestimmten Zeitpunkt etc.) und der Scheduler versucht mit Hilfe der Performanzvorhersagen diese Ziele zu erreichen.

Der Silver Scheduler bietet für diesen Zweck eine pro Job vorhandene und/oder systemweite „Job Utility“ Funktion, welche bestimmt wann und wo ein Job ausgeführt werden soll. Dabei werden Aspekte wie definierte Startzeit des Jobs, Rückkehrzeit eines Jobs und Ressourcekosten beachtet.

Im Bereich des Grid Computing werden auch oftmals diese Vorhersagen und das Sammeln der nötigen Daten vom so genannten „Network Weather Service“ zur Verfügung gestellt. Dieser Service erstellt in seiner derzeitigen Implementierung (Rich Wolski, Neil T. Spring, Jim Hayes, UCSD and University of Tennessee) kurzzeitige Performanzvorhersagen auf Basis von Performanzmessungen in der Vergangenheit. Das Ziel ist es eine dynamische Charakterisierung des Grids und Vorhersagen über die Performanz auf Grid-Anwendungsebene von einer Auswahl von Netzwerk- und Computerressourcen zu ermöglichen.

4.4 Ressourcenverwaltung

Um die für eine Anwendung ausgewählten Ressourcen zu nutzen und anderen vorhandenen Schedulern mitzuteilen, welche Ressourcen benötigt werden, muss die Reservierung von Ressourcen für einen bestimmten Zeitraum möglich sein. Diese

Reservierungen ermöglichen es den Resource bzw. Job Schedulern geeignete Scheduling Algorithmen für ihren lokalen Gridknoten anzuwenden. Außerdem steigern sie die Vorhersagbarkeit der Gridcharakteristik.

Auf der anderen Seite müssen die einzelnen Knoten Garantien über die Qualität ihrer angebotenen Ressourcen in dem reservierten Zeitraum geben können, da eine Reservierung sonst hinfällig wäre.

Silver erlaubt so genannte „Advanced Reservations“ dadurch, dass das Multischeduling via Maui/PBS diese unterstützen. Bei einer Ressourcenreservierung für eine Grid Anwendung reicht Silver diese an das PBS weiter, damit Maui diese Information für die Berechnung eines geeigneten Schedules benutzen kann. Mit Bestätigung der Reservierung ist gleichzeitig die Garantie über die Ressourcen für den angegeben Zeitraum gegeben.

Statt der anfangs erwähnten Möglichkeit, die benötigten Ressourcen durch den Programmierer festlegen zu lassen, wäre es auch möglich zu versuchen aus der vorliegenden Anwendung die nötigen Informationen zu gewinnen. Hierbei gibt es zwei denkbare Ansätze. Zum einen gibt es den Ansatz die Anwendung durch einen gewichteten Ablaufgraphen darzustellen. Dies würde auch etwaige Nebenläufigkeit darstellen, was die Möglichkeit mit sich bringt, die nebenläufigen Programmteile auf verschiedenen Knoten des Grids auszuführen. Aus dem gewichteten Ablaufgraphen versucht man dann, die benötigten Ressourcen in bestimmten Zeiträumen zu errechnen.

Ein anderer Ansatz ist, eine Anzahl an Programmcharakteristiken festzulegen und die an den Grid Scheduler gesendete Anwendung auf diese Charakteristiken hin zu überprüfen. Aus den so gewonnenen Erkenntnissen werden dann die benötigten Ressourcen errechnet.

4.5 Ziele des Schedulings

Das Ziel eines jeden Grid Schedulers ist einen möglichst besten Schedule zu finden und diesen umzusetzen. Eine mögliche Lösung, dies mit den bis hierhin vorgestellten Konzepten und Ansätzen als Vorraussetzung zu erreichen, wäre eine Auswahl an in Frage kommenden Gridknoten zu treffen, eine Performanzvorhersage der Anwendung unter Beachtung eventueller Performanz- und Ressourcekriterien des Programmierers auf diesen zu erstellen, diese miteinander zu vergleichen und auf der „besten“ Auswahl die Anwendung dann auszuführen. Dabei können auch unter Umständen mehrere Knoten in einer Auswahl zusammengefasst werden und somit die Anwendung später ausführen. Dabei muss beachtet werden, dass die Erhebung der nötigen Daten, die Berechnung der Vorhersagen, der anschließende Vergleich der ausgewählten Ressourcen und schließlich die Ausführung der Anwendung nicht mehr Kosten im Sinne der Performanz- und Ressourcenkriterien verursacht, als wenn man die Anwendung nach einem beliebigen Algorithmus „scheduled“.

Eine andere Lösung wäre eine Reihe von Algorithmen vorher zu bestimmen, welche die Ressourceansprüche einer Anwendung beachten würden und mit dem Ziel einer minimalen Ausführungszeit diese dann „schedulen“. Dies ist allerdings nicht flexibel und geht nur bedingt auf die Dynamik des zugrunde liegenden Systems ein.

Der Silver Scheduler optimiert seine Entscheidungen, welche auf regelmäßig aktualisierten Daten über die globale und lokale Auslastung des Grids, der Knoten und der Netzwerkauslastung oder eventuellem Ausfall einzelner Komponenten

basieren, bezüglich des Schedules durch neuerliche Evaluierung in bestimmten Zeitintervallen.

Quellen

[1] POSIX Standard 1003.2d, Webseite: http://standards.ieee.org/catalog/olis/posix.html
[2] NASA Ames Research Center, Webseite: http://www.arc.nasa.gov/
[3] Altair Grid Technology, LLC, Webseite: http://www.altair.com
[4] Cluster Resources, Inc., Webseite: http://www.supercluster.org
[5] PBBWeb, Webseite: http://www.cs.ualberta.ca/~pinchak/PBSWeb/
[6] The Maui Scheduler Homepage, Webseite: http://www.supercluster.org/maui/
[7] MME Webseite: http://www.cs.ualberta.ca/~pinchak/PBSWeb/
[8] David Jackson, Quinn Snell, Mark Clement: „Core Algorithms of the Maui Scheduler“, Webseite: http://link.springer.de/link/service/series/0558/papers/2221/22210087.pdf
[9] The Silver Scheduler Homepage, Webseite: http://www.supercluster.org/silver/
[10] Supercomputing Conference 2002
IEEE International Conference on Cluster Computing
IEEE Symposiums on High Performance Distributed Computing
International Parallel (and Distributed) Processing Symposium
International Conference on Parallel Processing
“Maui Scheduler Documentation” at High Performance Computing Center North, Website: http://www.hpc2n.umu.se/doc/maui/index.html

Bandbreiten-Monitoring mit NWS

Alexander Ritter, Gregor Höfert

ghoefert@rz.uni-potsdam.de

Dieses Dokument beschreibt den Aufbau und die Funktionsweise von NWS. NWS ist ein Tool zur Messung verschiedener Charakteristika eines Netzwerkes. Es werden die mathematischen Modelle zur Erstellung von Vorhersagen sowie ein mögliches Szenario betrachtet.

Inhaltsverzeichnis:

1 Begriffseinführung 3
1.1 NWS 3
1.2 Bandbreite 3
1.3 Latenz 4
2 Architektur von NWS 4
2.1 Anforderungen 4
2.2 Realisierung 4
3 Die Forecaster-Methoden 6
3.1 CPU-Forecasting 6
3.2 Netzwerk-Forecasting 7
3.3 Forecaster Allgemein 7
3.4 Mathematische Grundlagen 8
3.5 Komplexere Modelle 11
3.6 Dynamische Modellwahl 11
4 NWS in Anwendung 12
4.1 Installation und Starten von NWS 13
4.2 NWS-Tools 14
5 Vergleichbare Messmethoden 15
6 Auswertung der Messungen 16
7 Ausblick (NWS vs. Globus) 19
8 Literatur 21

1 Begriffseinführung

In diesem Abschnitt werden zum Verständnis der vorliegenden Arbeit Begriffe eingeführt, die fortwährend genutzt werden.

1.1 NWS

NWS steht als Abkürzung für Network Weather Service. Die Idee für NWS wurde 1997 an der UCSB (University of California in Santa Barbara) von Rich Wolski und Martin Swany geboren. Sie wollten ein System entwickeln, welches nicht nur allerhand Messungen durchführen kann, sondern auch gezielte Vorhersagen erstellen kann. Dass heißt, NWS sollte für den Einsatz im Netzwerk und speziell für das sich entwickelnde Grid-Computing genutzt werden können.

Messungen, die mit NWS durchgeführt werden können, sind CPU-Messungen, Messungen zur Speicherauslastung, sowie Bandbreiten- und Latenzmessungen. Anhand dieser Messungen können dann gezielte Vorhersagen erstellt werden. Dieser Vorgang wird durch diese Ausarbeitung genauer im Abschnitt 3 vorgestellt.

Das aktuelle Release des Systems ist die Version 2.6 vom 18.06.2003. Details zur Architektur finden Sie im Abschnitt 2, zur Installation und Benutzung im Abschnitt 4.

1.2 Bandbreite

Der Begriff der Bandbreite wird häufig benutzt und stammt ursprünglich aus der Funktechnik. Als Definition der Bandbreite gibt man die Datenmenge an, die pro Zeiteinheit über das Netz verschickt werden kann. Bei NWS kann man die Größe des zu übersendenden Datenpaketes explizit festlegen. Nähere Details dazu erfahren Sie im Abschnitt 4.

Abbildung 1 zeigt einen Verlauf der Messung. Die genaue Bandbreite wird dann über die Formel des effektiven Durchsatzes bestimmt. Dabei wird die Größe des Datenpaketes durch die Transferzeit minus der Round Trip Time dividiert.

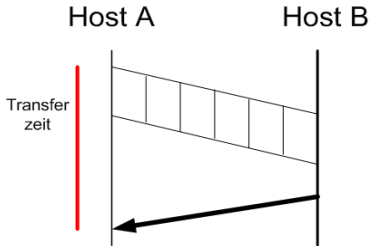


Abbildung 1

Unter Round Trip Time versteht man dabei die Zeitspanne, die erforderlich ist, um ein Signal von einer Quelle über das Netzwerk zum Empfänger zu senden und die Antwort des Empfängers wiederum über das Netzwerk zurück zum Sender zu transportieren.

1.3 Latenz

Die Latenz stellt die minimale Übertragungsverzögerung beim Senden einer Nachricht dar. Im Allgemeinen handelt es sich dabei um das Zeitintervall vom Ende eines Ereignisses bis zum Beginn der Reaktion auf dieses Ereignis. NWS approximiert die Latenz als die Hälfte der eben beschriebenen Round Trip Time für eine kleine Nachricht, die 4 Byte groß ist.

2 Architektur von NWS

2.1 Anforderungen

Bevor die Architektur von NWS betrachtet wird, so sollte man sich fragen, welchen Anforderungen ein solches System genügen muss. Dabei kommt man zu dem Ergebnis, dass die erstellten Vorhersagen möglichst genau sein sollten, um diese auch entsprechend nutzen zu können.

Außerdem sollte die Benutzung von Ressourcen durch das System minimal gehalten werden, um nicht unnötigen Overhead zu produzieren.

Ein weiterer wichtiger Punkt ist die Allgegenwärtigkeit im zu messenden System und die dauerhafte Ausführung. Es nützt nämlich nichts, wenn ab und zu einzelne Messungen durchgeführt werden und anhand dieser Vorhersagen erstellt werden. Diese sind dann nicht zu gebrauchen, da sie fehlerhaft und ungenau sind.

Ein letzter Punkt ist die Portabilität, das System sollte natürlich auf verschiedenen Plattformen lauffähig sein.

2.2 Realisierung

Die Architektur von NWS besteht aus 4 Komponenten: Name Server Process, Persistant State Process (Memory Server), Sensor Process und Forecaster Process.

Alle Prozesse sind zustandslos und können auf unterschiedliche Knoten des Systems verteilt werden. Eine Kommunikation zwischen den Prozessen ist trotzdem jederzeit möglich.

Name Server Process

Der Name Server Process stellt einen Verzeichnisdienst zur Verfügung, bei dem sich alle anderen Prozesse anmelden und registriert werden. Außerdem bietet der Name Server Process Tools an, mit denen das System verwaltet werden kann. Pro System existiert dabei nur ein Name Server.

Persistant State Process

Der Persistant State Process speichert die empfangenen Messdaten in Dateien. Dabei wird der Dateiname aus der/den zu messenden Maschinen, der Art der Messung sowie dem Sensor-Port gebildet.

Der Persistant State Process kann auf verschiedenen Knoten des Systems verteilt sein, d.h. die Messdaten werden auf verschiedenen Rechnern des Systems gespeichert.

Sensor Process

Der Sensor Process hat die Aufgabe, Messdaten zu verschiedenen Ressourcen zu sammeln. Die gesammelten Daten werden sofort an den Persistant State Process gesandt. Zusätzlich zu den Messdaten wird dabei ein Zeitstempel übertragen und gespeichert. Mit dessen Hilfe ist es dann möglich, z.B. Messungen von verschiedenen Ressourcen mit dem gleichen Zeitstempel zu vergleichen.

Die beiden wichtigsten Sensoren sind der CPU-Sensor und der Netzwerk-Sensor, auf die gesondert im nächsten Abschnitt (**Abschnitt 3**) eingegangen wird.

Forecaster Process

Der Forecaster Process verarbeitet die gespeicherten Messwerte aus dem Persistant State Process. Dazu benötigt er jeweils Messpaare, die einen Zeitstempel besitzen. Für die Berechnung der Vorhersagen existieren verschiedene statistische Modelle. Es wird jeweils das Modell ausgewählt, welches den kleinsten spezifischen Fehler besitzt. Näheres dazu erfahren Sie im **Abschnitt 3**.

Das Zusammenspiel der Komponenten verdeutlicht die Abbildung 2.

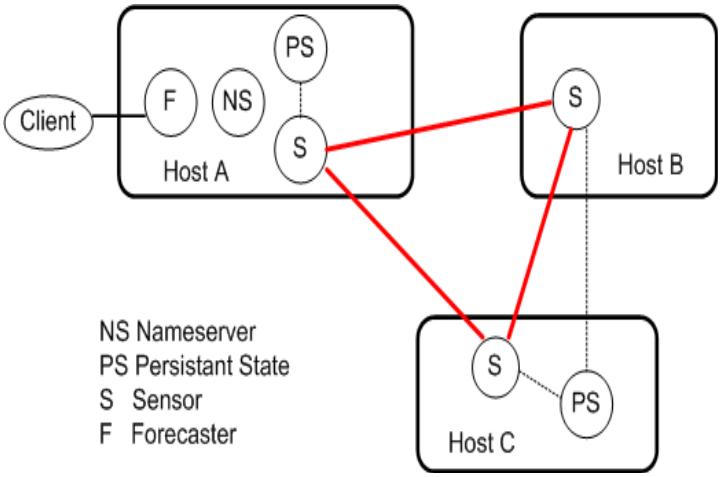


Abbildung 2

3 Die Forecaster-Methoden

3.1 CPU-Forecasting

Die CPU-Verfügbarkeit wird hier festgelegt als der Zeitanteil der CPU-Belegung, die ein neuer, vollpriorisierter Standard User Prozess erhalten kann.

Um diese Verfügbarkeit vorhersagen zu können, benutzt der Forecaster drei Messarten: Die beiden Systembefehle `vmstat` und `uptime` sowie eine eigene Aktive Probe. Die Aktive Probe ist ein Prozess, der genau wie oben beschrieben ein vollpriorisierter Standard User Prozess ist und welcher eine kurzzeitige rechenintensive Aufgabe auf dem Prozessor durchführen lässt. Die Rechendauer ist voreingestellt auf 3 Sekunden, kann aber ebenfalls je nach System-Dynamik variieren.

Die beiden Systembefehle vmstat und uptime sollen die Verfälschung der Messung möglichst gering halten; sie sind sehr systemnahe Befehle – demgegenüber hat die Aktive Probe viel Einfluss auf das zu messende System, bringt aber auch das beste Ergebnis. Das Ergebnis der Aktiven Probe wird als “Grundwahrheit” betrachtet und die Ergebnisse der anderen Werkzeuge werden über einen BIAS-Wert (für Abweichung oder Voreinstellung) dementsprechend angepasst.

Allerdings wird die Aktive Probe in zeitlich angepassten Zeitintervallen ausgelöst, die je nach Dynamik des Systems von 15 Sekunden bis 60 Minuten variieren können (voreingestellt sind 60 Sekunden); die Befehle vmstat und uptime werden standardmäßig alle 10 Sekunden abgefragt.

Eine Beispielausgabe von vmstat:

```
procs ---------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy  id  wa
 1  0      0  82492   4012 105656    0    0   377    15  157  220 10  8  82   0
```

Unter procs r steht die Anzahl der Prozesse mit “runnable”-Status, d.h. es läuft zurzeit nur ein User-Programm. Die CPU-Auslastung bescheinigt diesem einen Prozess daher 10 % der Prozessorzeit, 8 % gehen zu Lasten des Systems und die restlichen 82 % der Zeit wartet der Prozessor.

Beispielausgabe des uptime-Befehls:

```
09:02:02 up 55 min, 1 user, load average: 0.46, 0.20, 0.09
```

Der Befehl uptime gibt die maximale, die durchschnittliche und die minimale Prozessorlast an.

3.2 Netzwerk-Forecasting

Auch für die Netzwerklastbestimmung gibt es eine Aktive Probe; passive Werte sind in diesem Fall zwecklos. Die Aktive Probe misst die Latenzzeit einer Punkt-zu-Punkt-Verbindung durch Hin- und Zurücksenden eines 4 Byte-Pakets. Die Hälfte der so gemessenen Round-Trip-Time (RTT) eines Pakets ist die Latenzzeit (=RTT/2).

Um die Bandbreite zu messen werden signifikant große Datenpakete von 16, 32 oder 64 Kbyte über das Netz versendet und die Zeit bis zum entsprechenden ACK der Gegenstation gemessen (siehe Abbildung 1).

3.3 Forecaster Allgemein

Der Forecaster soll gute kurz- und mittelfristige Vorhersagen für die Bandbreite, Latenzzeit, CPU-und RAM-Verfügbarkeit ermöglichen; außerdem soll er einfach zu erweitern sein und eine schnelle Berechnung zur Vorhersage bereitstellen.

Die Daten der Sensoren werden wie im Beispielaufbau in Abbildung 2 an den Persistant State Server geschickt. Wird der Forecaster-Prozess aufgerufen, so extrahiert er die zu analysierenden Daten vom PS-Server und speichert die Datenreihen wieder dort ab, ergänzt um zusätzliche Vorhersagewerte. Dazu wird ein numerisches Berechnungsmodell genutzt, das intern aus mehreren alternativen statistischen Vorhersagemethoden immer die Methode mit der kleinsten Abweichung zu den Vergangenheitswerten auswählt. Diese Methoden können einfach als Module eingebunden werden. Zusammen mit dem Java NWS Sensor ist so eine 'live connectivity'-Analyse möglich.

Am interessantesten ist am NWS-Forecaster der mögliche Einsatz als Entscheidungsgrundlage für den Globus-Scheduler und andere Scheduler zum verteilten Rechnen.

Schematischer Aufbau des Forecasters:

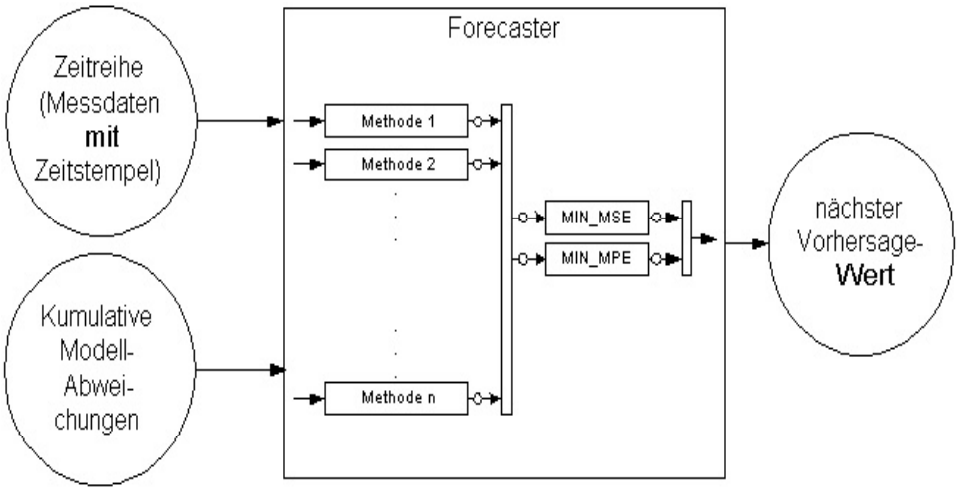


Abbildung 3

Auflistung der bekanntesten und am häufigsten verwendeten Methoden:

Vorhersage- Methode	**Beschreibung**
RUN_AVG	Laufender Durchschnitt
SW_AVG	Sliding-Window Durchschnitt
LAST	Letzte Messung verwenden
ADAPT_AVG	Adaptives Window Durchschnitt
MEDIAN	Median der letzten 21 Messungen
ADAPT_MED	Adaptives Window Median
TRIM_MEAN	α-getrimmter Durchschnitt
GRAD	Stochastischer Gradient
AR	Autoregression
MIN_MAE	Adaptives Minimum des abs. Fehlers

3.4 Mathematische Grundlagen

Kurze Definitionen:
Vorhersagewert(t) =Vorhersagemethode(Messwert(t), alteMessreihe(t))
Fehler(t) = Messwert(t) – Vorhersagewert(t-1)

Am Anfang der Vorhersage gibt es meist nicht genug Werte, um eine bessere Vorhersage zu erstellen; in diesem Fall nimmt man einfach den vorhergegangenen Wert auch als nächsten Wert an:
LAST(t) = SW_AVG(t,0)

Durchschnittsbasierte Modelle:

Der laufende Durchschnitt ist der Durchschnitt aller bisherigen Messwerte.

RUN_AVG(t)= $$\frac{1}{(t+1)}\sum_{i=1}^{t} Messwert(t)$$

Da diese Berechnung u.U. zu viele Werte benötigt, greift man auf die letzten K Werte zurück, man erhält den Sliding-Window-Durchschnitt:

SW_AVG(t, K) = $$\frac{1}{(K+1)}\sum_{i=t-K}^{t} Messwert(t)$$ mit Fenstergröße K

Beim adaptiven Durchschnitt variiert die Fenstergröße K und wird dynamisch angepasst:

ADAPT_AVG(t) = $$\begin{cases} SW_{AVG}(t, K(t)-1) & falls \quad \min_{i=-1,0,1} Fehler_i(t) = Fehler_{-1}(t) \\ SW_{AVG}(t, K(t)) & falls \quad \min_{i=-1,0,1} Fehler_i(t) = Fehler_0(t) \\ SW_{AVG}(t, K(t)+1) & falls \quad \min_{i=-1,0,1} Fehler_i(t) = Fehler_1(t) \end{cases}$$

mit K(t+1) = $$\begin{cases} K(t)-1 & falls \quad \min_{i=-1,0,1} Fehler_i(t) = Fehler_{-1}(t) \\ K(t) & falls \quad \min_{i=-1,0,1} Fehler_i(t) = Fehler_0(t) \\ K(t)+1 & falls \quad \min_{i=-1,0,1} Fehler_i(t) = Fehler_1(t) \end{cases}$$

Gradienten-basiertes Modell:

Das nächst-komplexere Modell beruht auf einer Gradienten-Berechnung, wie sie zum Beispiel auch im Kernel implementiert ist, um bei TCP/IP per Gradienten Filter die Round-Trip-Time dynamisch zu bestimmen.

GRAD(t, g) = (1 – g) * GRAD(t – 1, g) + g * Wert(t)

Der Wert g (= gain) bestimmt die Akkuratheit, mit der GRAD den Durchschnitt der Zeitreihe und die Verzögerungszeit berechnet, bevor GRAD gegen einen stabilen Vorhersagewert konvergiert. Vernünftige Werte für g sind $0 < g < 1$, bspw. g=0,05.

Median-basierte Modelle:

$Sort_K$ = aufsteigend Sortierte Folge der letzten K Messwerte
$Sort_K(j)$ = j.ter Messwert der Reihenfolge

MEDIAN(t, K) =
$$\begin{cases} Sort_K((K+1)/2) & falls\ K\ ungerade \\ \dfrac{Sort_K(K/2)+Sort_K(K/2+1)}{2} & falls\ K\ gerade \end{cases}$$

Neben der einfachen Median-Berechnung gibt es noch die adaptive Median-Berechnung, bei der die Anzahl der betrachteten Werte K variiert:

ADAPT_MED(t) =
$$\begin{cases} MEDIAN(t, K(t)-1) & falls \min\limits_{i=-1,0,1} Fehler_i(t) = Fehler_{-1}(t) \\ MEDIAN(t, K(t)) & falls \min\limits_{i=-1,0,1} Fehler_i(t) = Fehler_0(t) \\ MEDIAN(t, K(t)+1) & falls \min\limits_{i=-1,0,1} Fehler_i(t) = Fehler_1(t) \end{cases}$$

$$mit\ Fehler_i(t) = (Messwert(t) - MEDIAN(t, K(t)+i))^2$$
$K(t)$ ist Messwert bei Zeit t
$K(t+1)$ wird wie für ADAPT AVG berechnet

α -getrimmter Median:

Man setze α als Trimmwert ein, sodass K – 2*α*K Werte innerhalb eines K-großen Windows mit (0 < α < 0,5) liegen.

Dann definieren wir: $T = \lfloor \alpha * K \rfloor$

TRIM_MEAN(t, K, T) =
$$\sum_{j=T+1}^{K-T+1} \left(\frac{1}{(K-2*T)} Sort_K(j)\right)$$

Hierbei wird auch α dynamisch ähnlich K in ADAPT_AVG und ADAPT_MED dynamisch angepasst, allerdings gibt es keine offensichtliche Korrelation zwischen α und K.

Autoregression:

Versuche belegen, dass Internetverkehr gut durch autoregressive, integrierte Methoden mit wechselndem Durchschnitt (ARIMA) modelliert werden kann, was allerdings einen enormen Rechenaufwand zur Vorhersage erfordern würde. Pure Autoregression der p-ten Ordnung (kurz AR) ist nicht so komplex wie ARIMA, da nur lineare Gleichungen zu lösen sind, die sich rekursiv lösen lassen. Aufgrund des hohen Zeitverbrauchs von O(p * n) für n Messungen wird die AR nur mit Sliding Window angewendet:

$$AR(t,p) = \sum_{i=0}^{p} a_i * Wert(t-i)$$

3.5 Komplexere Modelle

Die Semi-Nonparametrische Zeitreihen-Analyse (SNP) nach Gallant und Tauchen ist eine ganze Sammlung von stationäre Autoregressionsberechnungen mit und ohne Gaus'schem Rauschen, Autoregressive bedingte heteroscedastische Modelle (ARCH), generelle non-lineare Prozesse mit heterogenen Innovationen und weiteren Modellen.

SNP ist zweiphasig: Anhand der alten Zeitreihe werden alle Modelle durchgerechnet und mit dem Modell mit der geringsten Abweichung wird der nächste Vorhersagewert errechnet.

Tatsächlicher Vergleich ergibt:

Methode	Durchschn. Fehler	Kosten pro Vorhersage
SNP	0,437	1090 ms
NWS	0,450	7,29 ms

3.6 Dynamische Modellwahl

Alle Modelle werden beibehalten und einzeln ausgewertet mit

Fehler(t) = Messwert(t) – Vorhersagewert(t-1)

$$MSE(t) = \frac{1}{(t+1)} \sum_{i=0}^{t} (Fehler(i))^2$$

$$MPE(t) = \frac{1}{(t+1)} \sum_{i=0}^{t} |Fehler(i)|/Wert(i)$$

Anhand der niedrigsten Fehlerabweichung wird entweder MSE (medium square error = durchschnittlicher quadratischer Fehler) oder MPE (medium procentage error = durchschnittlicher prozentualer Fehler) zur Bestimmung der nächsten Vorhersagemethode gewählt (siehe Abbildung 3).

Beispieltabelle für die durchschnittlichen quadratischen und prozentualen Fehler der einfachen Forecaster-Methoden bezüglich einer hier nicht aufgeführten Zeitreihe:

Vorhersagemodell	MSE	MPE
RUN_AVG	0,5274	0,0927
SW_AVG	0,5041	0,0902
LAST	0,7892	0,1066
ADAPT_AVG	0,5214	0,0925
MEDIAN	0,5386	0,0901
ADAPT_MEDIAN	0,5337	0,0896
TRIM_MEAN	0,5130	**0,0893**
GRAD	**0,4903**	0,0895
AR	0,7139	0,0992
MIN_MSE	0,5136	0,0901
MIN_MPE	0,5417	0,0906

Hier würden also nach MSE die Gradienten-Berechnung oder nach MPE der α-getrimmte Median ausgewählt. Da MSE und MPE nicht direkt vergleichbare Werte liefern, kann man entweder den Forecaster selbst wählen lassen oder direkt vorgeben, nach welcher Fehlerart die Modellwahl stattfinden soll.

4 NWS in Anwendung

Dieser Abschnitt soll einen kurzen Einblick liefern, wie NWS zu installieren und zu benutzen ist, d.h., welche Tools zur Administration zur Verfügung stehen. Eine ausführliche Beschreibung befindet sich in der Dokumentation zu NWS.

4.1 Installation und Starten von NWS

Der Download der aktuellsten NWS-Version von http://nws.cs.ucsb.edu/download sollte am Anfang der Installation stehen. Danach entpackt man das Packet und konfiguriert es bspw. als einfacher User mit Angabe des Zielpfades ./configure –prefix=$HOME/nws_inst. Danach installiert man das Paket entweder mit make all oder mit make <component>, d.h. entweder das gesamte Paket oder nur eine einzelne Komponente, bspw. einen Sensor.

Als hilfreich hat es sich erwiesen, eine Initialisierungsdatei .nwsrc anzulegen. In dieser Datei kann man Standardeinstellungen für das System definieren. So z.B. die Bezeichnung des Name Server und Memory Server sowie zu benutzende Ports. Standardmäßige Ports sind 8095 für den Name Server, 8055 für den Memory Server, 8060 für den Sensor und 8070 für den Forecaster.

Auch die Verzeichnisse zur Speicherung der Messdaten können in der Initialisierungsdatei festgelegt werden.

Nach der abgeschlossen Installation kann man das System hochfahren.

Als erstes startet man dazu den Name Server, anschließend Memory Server, Sensor sowie Forecaster Process. Die Befehle dazu lauten:

nws_nameserver [-c seconds] [-e file] [-f file] [-l file] [-p port] &

nws_memory [-C cache entries] [-d directory] [-e file] [-f file] [-N Nameserver] [-L] [-p port] &

nws_sensor [-c yes/no] [-a name] [-e file] [-l file] [-M Memoryserver] [-N Nameserver] [-L] [-p port] &

nws_forecast [-D] [-e file] [-l file] [-N Nameserver] [-L] [-p port] &

Um nun z.B. Bandbreitenmessungen durchzuführen, muss mindestens auf einem zweiten Rechner ein korrekt eingestellter Sensor gestartet werden.

Die Tools zur Administration werden im folgenden Abschnitt kurz aufgezeigt.

4.2 NWS-Tools

Die folgende Tabelle gibt einen Überblick über die vorhandenen Tools, sowie eine kurze Erläuterung dieser.

Tool	*Beschreibung*
add_forecast	Leitet Vorhersagenaus Messdaten her
ctrl_mem	Kontrolliert den Speicherplatz
html_hosts	Produziert eine HTML-Tabelle mit Messwerten zu Latenz und Bandbreite zw. 2 Rechnern
nws_extract	Bietet leichten Zugang zu Messdaten und Vorhersagen
nws_insert	Bietet die Möglichkeit, Messdaten anderer Programme für NWS zu portieren
nws_search	Listet Objekte auf, die beim Name Server registriert sind
nws_host	Kontrolliert laufende NWS Hosts und kann Befehle senden
start_activity	Kann NWS Aktivitäten starten (Messungen)
halt_activity	Kann NWS Aktivitäten beenden
nws_hostadmin	Assestiert bei der Administration entfernter NWS Hosts
nws_ping	Testet die Verbindung zu anderen NWS Hosts, liefert Bandbreite und Latenz
whattime	Wandelt die Zeitstempel zu Datum und Uhrzeit um

Bsp. zum Start einer Bandbreitenmessung zwischen 4 Hosts:

1. Festlegen einer Konfigurationsdatei bspw. *Myclique* mit Inhalt:

```
name:colors                  #Name der Aktivität
controlName:clique           #clique-Schlüsselwort für Messungen zw. Hosts
skillName:tcpMessageMonitor  #was gemessen werden soll hier: TCP
member:orange.ufo.edu        #Mitglied der Gruppe
member:yellow.ufo.edu        #Mitglied der Gruppe
member:red.ufo.edu           #Mitglied der Gruppe
```

member:green.ufo.edu *#Mitglied der Gruppe*
size:32 *#Größe des Datenpaketes=32 kByte*
period:120 *#Zeitintervall=2 min*

Zum Start der Messung muss noch folgender Befehl aufgerufen werden:
start_activity -f Myclique red.ufo.edu

Damit wird bewirkt, dass die Aktion, die in der Datei *Myclique* formuliert ist, ausgeführt wird. red.ufo.edu ist dabei der beheimatende Host der Aktivität.

Weitere Ausführungen zu den Tools finden Sie entsprechend in der NWS Dokumentation.

5 Vergleichbare Messmethoden

An dieser Stelle möchten wir als vergleichbare Messmethode auf das Tool **Pinger** eingehen, das wie schon am Namen sichtbar, das Kommando *ping* benutzt. Die Grundlage für das Kommando ist das ICMP-Protokoll mit Typ 0 und 8 - *echo request* und *reply* , welches dafür benutzt wird, die Erreichbarkeit von Rechnern zu überprüfen.

Die Architektur von Pinger ist in Abbildung 4 ersichtlich. Es besteht aus drei Komponenten, der *remote monitoring site*, *monitoring site* und der *analysis and archive site*.

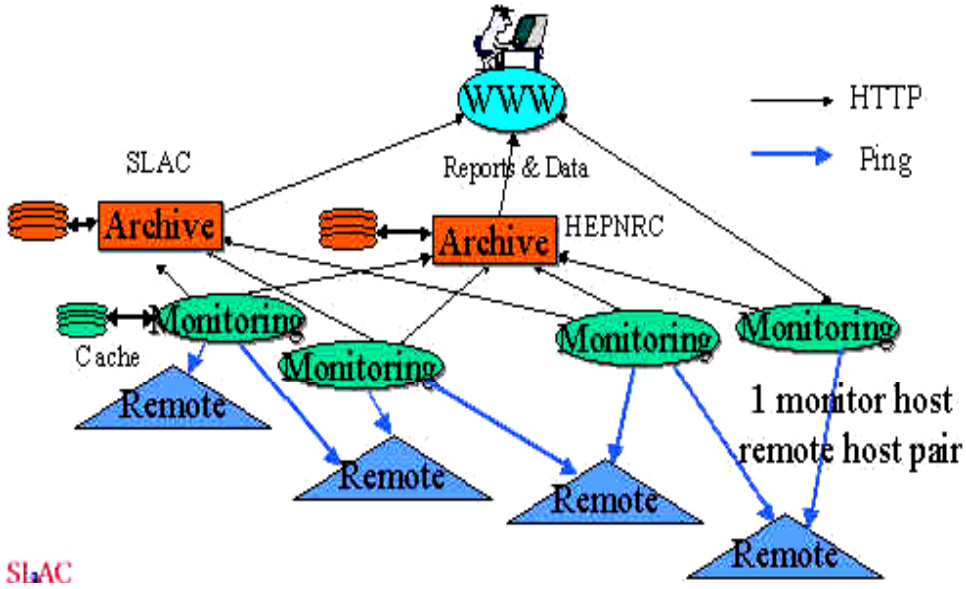


Abbildung 4

Die Messungen können wie anhand Abbildung 4 ersichtlich immer zwischen einer *monitoring site* und einer *remote monitoring site* durchgeführt werden. Per HTTP

werden die Messdaten dann an die *archive and analysis site* geschickt.

An dieser Stelle werden schon die Nachteile von Pinger sichtbar. Es ist nicht möglich, wie bei NWS, Messungen zwischen beliebigen Rechnern durchzuführen.

Außerdem kann dieses Tool lediglich die Bandbreite messen, obwohl man mit kleinen Paketgrößen sicher auch eine Latenzmessung simulieren könnte. Ein weiterer Nachteil von Pinger ist die eigentliche Berechnung der Bandbreite. Zur Messung wird ein Paket an einen Zielrechner geschickt und die Zeit gemessen, bis das *reply* für dieses Paket eintrifft. Diese Zeitspanne wird anschließend durch zwei dividiert. Dabei können sehr schnell Fehler und Ungenauigkeiten auftreten, da das Paket auf dem Rückweg eine andere Route gewählt haben könnte als auf dem Hinweg.
Eine Vorhersage-Möglichkeit ist mit diesem Tool nicht gegeben. Die *archive and analysis site* bietet aber Möglichkeiten zur Auswertung gespeicherter Messergebnisse, beispielsweise grafischer Natur.

Ein Test dieses Tools wurde durch uns an dieser Stelle nicht durchgeführt. Dies lag zum einen an fehlender Zeit und zum anderen an den sicherheitstechnischen Beschränkungen des HPI. Ein *ping* in das CORBA-Testbed ist von außerhalb nicht möglich.

6 Auswertung der Messungen

Die an den Messungen beteiligten Rechner waren:
ccm1.ccm.hpi.uni-potsdam.de //CORBA-Testbed HPI
ccm9.ccm.hpi.uni-potsdam.de //CORBA-Testbed HPI
hoedic.haiti.cs.uni-potsdam.de //Institut für Informatik (anstelle Uranus Cluster)
origin.aei.mpg.de //Cluster des Albert-Einstein-Insituts Potsdam

Der Aufbau unseres Systems gestaltet sich wie folgt (nicht alle Messungen werden nachfolgend als Grafik aufgeführt):
Nameserver: ccm1
Memoryserver: ccm1
Forecaster: ccm1
Sensoren: ccm1, ccm9, hoedic, origin

Die Prozessor-Verfügbarkeit und -Nutzung auf CCM1:

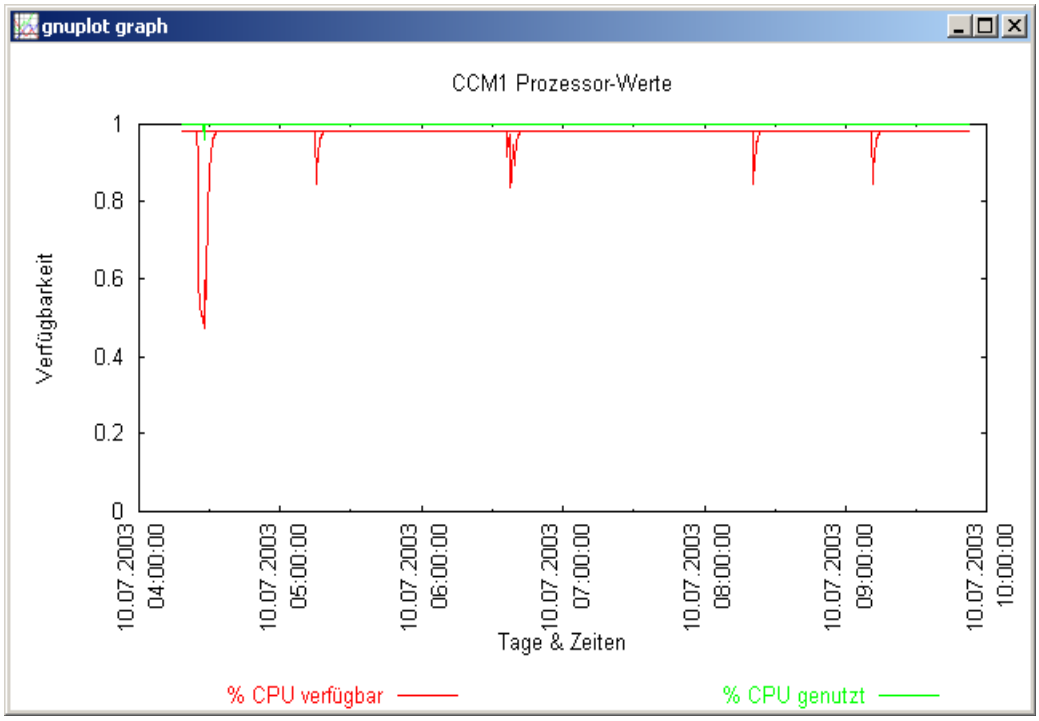


Abbildung 5

Bandbreitenmessung von CCM1 nach CCM9 und Hoedec.Haiti:

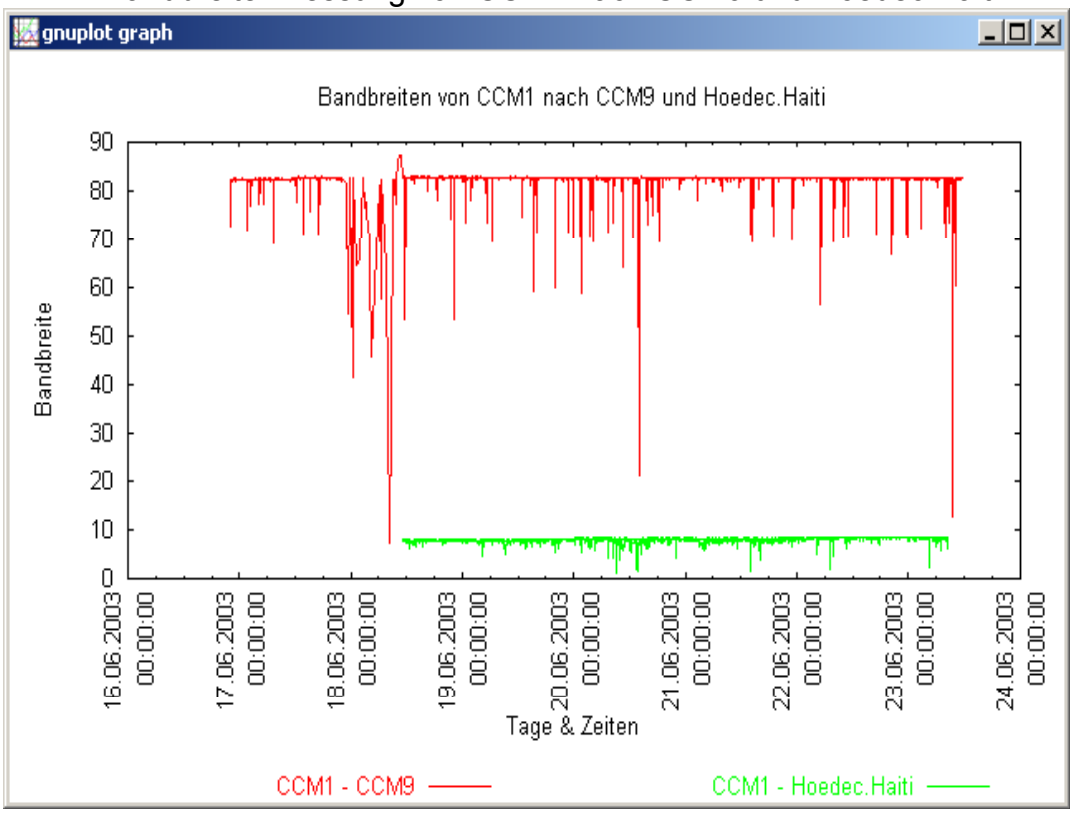


Abbildung 6

Latenzzeitmessung von CCM1 nach CCM9 und Hoedec.Haiti:

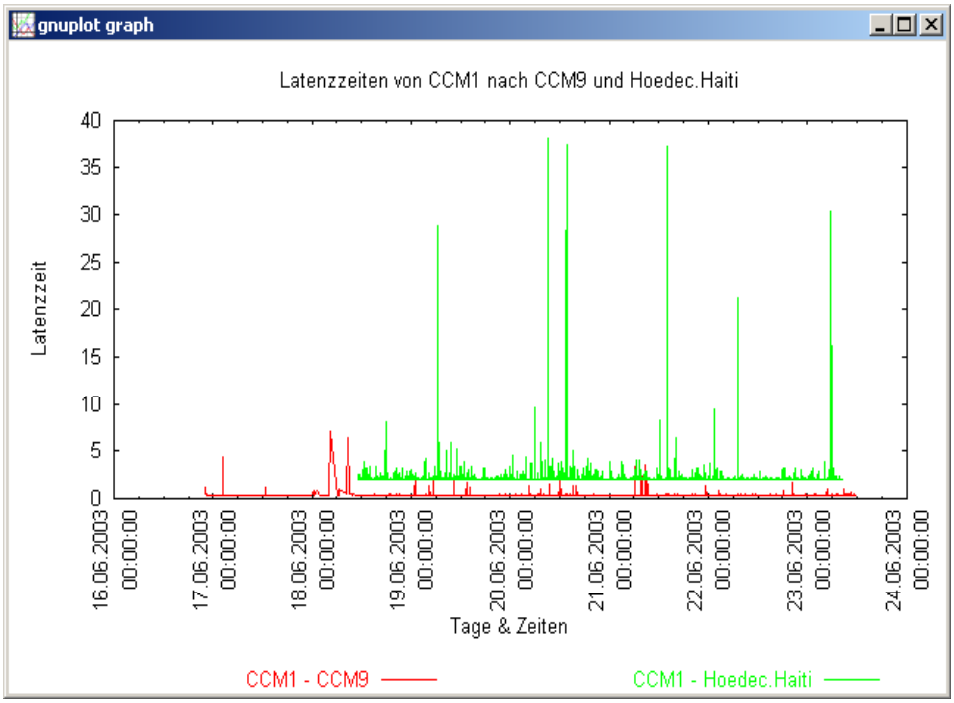


Abbildung 7

Zusammenstellung mit Bandbreite und Latenz von CCM1 nach CCM9 und Hoedec.Haiti:

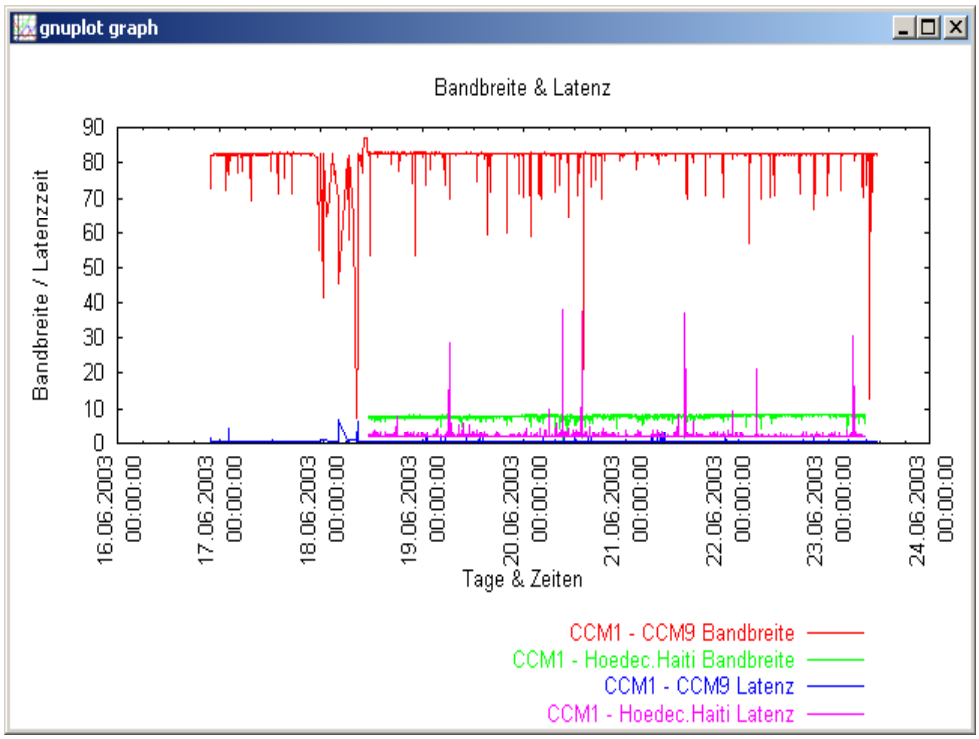


Abbildung 8

7 Ausblick (NWS vs. Globus)

In dem letzten Abschnitt unserer Arbeit möchten wir einen kurzen Ausblick geben, wie NWS zusammen mit Globus genutzt werden kann.

Das Ziel der Nutzung sollte sein, die NWS-Vorhersagen, die auf Abruf von registrierten Rechnern erstellt werden können, mit der Globus-Funktionalität zu vereinen.

Die prinzipielle Funktionsweise des so entstehenden Systems ist in Abbildung 9 ersichtlich. Es existieren zwei ldap-Server, ein Dämon für das Rechnernetzwerk, z.B Cluster, und ein Dämon für den NWS-Knoten. Ldap-Anfragen des Clusters werden dabei an den Dämon des NWS-Knoten übermittelt und mit Hilfe eines shell-backend in NWS Anfragen konvertiert und an das NWS-System geschickt. Die Registrierung und Verwaltung, bspw. Cliquen-Erstellung, wird ebenfalls mit den NWS-Tools erreicht. Die Anfragen müssen dabei jeweils entsprechend konvertiert werden.

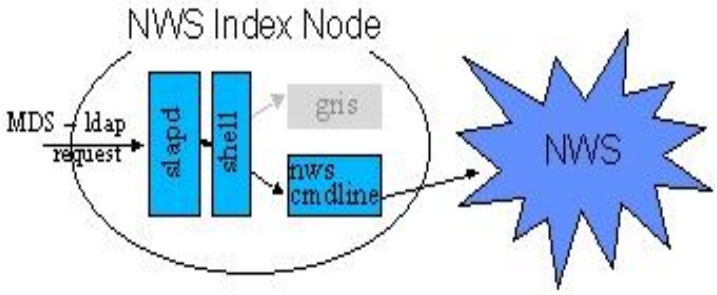


Abbildung 9

Die Netzwerk-Konfiguration wird verdeutlicht in Abbildung 10.

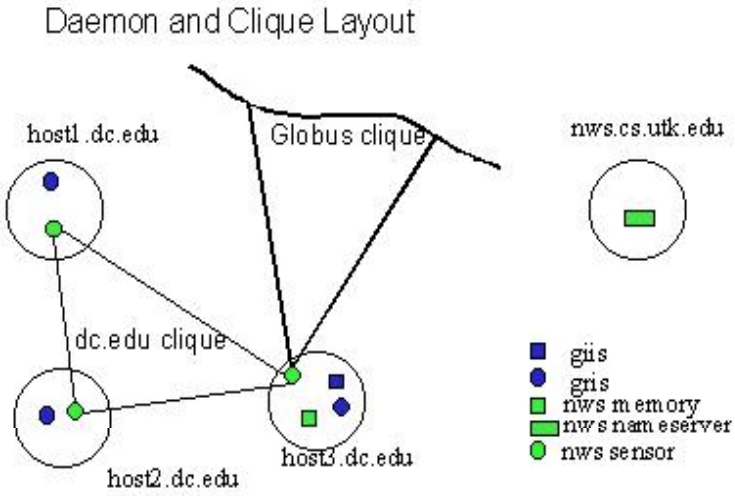


Abbildung 10

Es existiert eine globale Globus Clique, die viele kleine Cliquen enthält (vgl. *dc.edu clique*). Auf jedem Knoten dieser Clique läuft ein NWS-Sensor sowie ein Grid

Resource Information Service. Auf dem Knoten, der mit der Globus Clique verbunden ist, läuft zusätzlich noch ein NWS-Memory Server zum Speichern der gesammelten Messdaten und der Grid Index Information Service, der Informationen zu allen Ressourcen der Clique enthält.

Der Nameserver kann sich auf einem beliebigen Knoten der globalen Clique befinden.

Anfragen auf NWS via Globus kann man z. B. mit Hilfe folgender Befehle erreichen:

grid-info-search -b "service=NWS, o=Grid" "<-filter>“

grid-info-search -h nws.cs.utk.edu -p 390 / -b "service=NWS, o=Grid" "<-filter>“

Wobei mögliche Filter wie folgt definiert werden:

source-hostname=bolas.isi.edu

destination-hostname=nws.cs.utk.edu

(&(source-hostname=pitcairn.mcs.anl.gov) (destination-hostname=burns.ecs.csun.edu))

8 Literatur

- http://nws.cs.ucsb.edu
- http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html#pinger
- http://www-fp.globus.org/retreat00/presentations/nws_wolski
- http://citeseer.nj.nec.com/cs
- http://w3.siemens.de/solutionprovider/_online_lexikon/start.htm

The Paradyn Parallel Performance Measurement Tool

Jens Ulferts, Christian Liesegang
Hasso-Plattner-Institut der Universität Potsdam
{jens.ulferts, christian.liesegang}@student.hpi.uni-potsdam.de

Abstract

Diese Ausarbeitung beschäftigt sich mit dem Problem, Performance Bottlenecks (Flaschenhälse) in großen parallelen Anwendungen zu lokalisieren. Das Überwachen und Sammeln von Informationen über das Laufzeitenverhalten stellt allerdings ein großes Problem da. Dies Ansammeln enormer Datenmengen kann selber das Laufzeitverhalten einer zu überwachenden Anwendung beeinflussen. Weiter müssen diese Datenmengen aufbereitet und sinnvoll für den Anwender visualisiert werden. Ein neuer Ansatz stellt das dynamische Instrumentieren dar, welches dem Anwender ermöglich, zur Laufzeit Entscheidungen über die Art und Menge der zu sammelnden Daten zu treffen. Mit Hilfe des W^3-Such-Modells kann dies erreicht werden. Paradyn ist eine Anwendung die das dynamische Instrumentieren und das W^3-Such-Modell in sich vereint.

1 Einführung

Durch den Zusammenschluss mehrerer Computer zu einem Grid sind Berechnungen ausführbar, die von einem einzelnen Computer nicht möglich sind oder inakzeptabel lange brauchen. Der Vorteil ist jedoch nur gegeben, wenn die Software, die auf dem Grid läuft, auch an die Bedingungen des Grids angepasst ist und die gebotenen Möglichkeiten optimal nutzt.

Um die Software anzupassen und zu optimieren, benötigt man zunächst Informationen über das Verhalten des Programms in dem System. Die Informationsfülle steigt allerdings mit zunehmender Rechenleistung und Verteilung des Programms rapide an. Ein Mensch ist nicht mehr in der Lage diese Informationen zu sammeln und ungefiltert zu nutzen. Hierfür werden Performance Debugger, spezielle Werkzeuge, benötigt.

2 Konzepte und Ansätze

Diese Werkzeuge untersuchen das Programm auf „Bottlenecks“(Flaschenhälse). Diese sind Software- oder Hardwarekomponenten, an denen ein signifikanter Anteil der gesamten Ausführungszeit verbracht wird. Der Begriff Bottleneck ist wertneutral. Die in einer Komponente verbrachte Zeit kann gerechtfertigt und optimal sein, ein Performance Tool wird diese Komponente nichtsdestotrotz als Bottleneck identifizieren. Innerhalb eines Programms können mehrere Bottlenecks auftreten. Bottlenecks treten beispielsweise in CPU, Synchronisationsobjekten oder einzelnen Codefragmenten auf. Für die Verständnisbildung bei dem Benutzer bezüglich des Systems werden vom Performance-Werkzeug drei Arbeitsschritte ausgeführt.

Zunächst werden an geeigneten Stellen Informationen gesammelt, durch Filtern und Aufbereiten reduziert und anschließend dem Benutzer präsentiert. Dabei treten verschiedene Probleme auf, die bei der Bearbeitung beachtet werden müssen. Für die Messung werden weitere Systemressourcen benötigt. Dadurch können die Ergebnisse verfälscht werden. Des Weiteren entstehen beim Sammeln der Informationen große Datenmengen. Dieser Sachverhalt begründet wiederum die Störung und erklärt ebenso das Problem bei der Suche nach den interessierenden Informationen.

Das Verständnis des Benutzers stellt den Anwendungszweck eines Performance

Werkzeugs dar. Deshalb muss eine geeignete Darstellung für die gesammelten Informationen gefunden werden. Wegen der Vielgestaltigkeit der untersuchten Programme und der unterschiedlichen Auffassungsweisen der Benutzer ist es schwierig, eine zweckmäßige Darstellung für das Verhalten zu finden.

Um die drei Schritte durchzuführen, werden von den verschiedenen existierenden Performance Debuggern unterschiedliche Vorgehensweisen praktiziert.

2.1 Sammeln

Informationen können an verschiedenen Stellen des Systems entnommen werden. Softwareseitig stehen die zu analysierende Software direkt und das Betriebssystem zur Verfügung. Auch von Seiten der Hardware kann nützliches Wissen erhoben werden. Die drei Quellen können entweder durch Software, Hardware oder einer Mischung aus beiden, so genannten Hybridsystemen, erschlossen werden. Generell muss auf eine möglichst geringe Störung des Systems geachtet werden, da die erhaltenen Informationen ansonsten nicht mehr aussagekräftig sind und das Systemverhalten nicht mehr widerspiegeln.

Ist die zu untersuchende Software die Quelle, so bieten sich verschiedene Möglichkeiten, die mit dem Entwicklungsprozess stark zusammenhängen

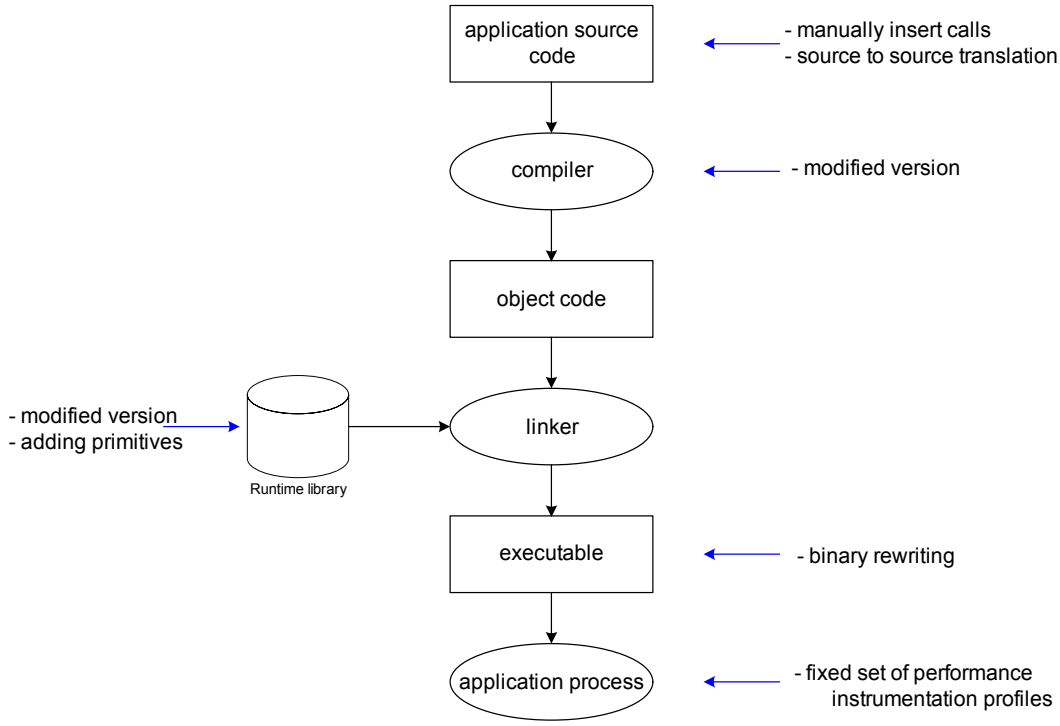


Abb. 1: Ansatzpunkte im Programmcode

Der erste Ansatzpunkt ist der Quellcode. Hier kann der Programmierer spezielle Codefragmente für die Performancemessung manuell einfügen. Da dies zeitaufwendig ist, ermöglichen manche Performace Debugger eine „source to source translation“. Dabei werden anhand von Regeln Systemaufrufe eingefügt, die für die Analyse genutzt werden können. Die „source to source translation“ muss allerdings für jede Sprache angepasst werden.

Durch das Modifizieren des Compilers, der aus dem Quellcode die binären Dateien erzeugt, können ebenfalls Systemaufrufe eingefügt werden, und Informationen, die zur Compilezeit bereitstehen, bieten einen guten Einblick in das Systemverhalten. Ebenso können Systemaufrufe durch Anpassen der verwendeten Runtime Library integriert werden. Hier können auch noch weitere Funktionen hinzugefügt werden, die vorher der Sprache nicht zur Verfügung standen. Auch diese beiden Ansätze sind an die jeweiligen Sprachen gebunden. Auch ist fraglich, ob die Compiler und Bibliotheken für eine Modifikation zur Verfügung stehen, da es sich dabei häufig um kommerzielle Produkte handelt.

Ein sprachunabhängiges Verfahren stellt das „binary rewriting“ dar. Nach dem Compilieren wird der erhaltene Binärcode mit weiterem Binärcode angereichert, der ein Sammeln von Informationen ermöglicht.

Die bisher vorgestellten Ansätze benötigen Entscheidungen über die zu sammelnden Daten vor dem Ausführen des Programms. Einige Performance Debugger verfolgen den Ansatz nach dem Laden der zu untersuchenden Software in den Speicher, noch ein Programm in den Speicher zu laden, das das Softwareverhalten auf Übereinstimmung mit vorher definierten Profilen überprüft.

Neben der Software selbst kann auch das System, das die Software enthält, zur Performance-Messung genutzt werden.

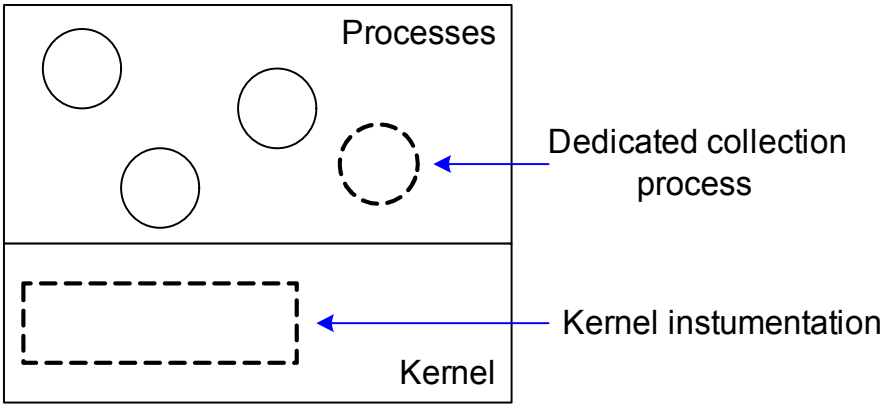


Abb. 2: Ansatzpunke im System

Durch Modifizieren des Betriebssystem-Kernels steht eine Vielzahl an unterschiedlichen Informationen zur Verfügung. Allerdings ist die Modifikation sehr zeitaufwendig und fehleranfällig. Auch ist fraglich, ob der Quellcode des Betriebssystems verfügbar ist. Wichtiger ist allerdings noch, dass die Software nicht mehr auf dem angedachten Betriebssystem läuft, die Aussagekraft der gesammelten Daten also fraglich ist.

Als weitere Messungsmöglichkeit kommen während der Ausführungszeit laufende Programme in Betracht, die das Systemverhalten untersuchen und dabei schon bestehende Funktionalitäten des Betriebssystems nutzen. Die Programme benötigen jedoch wiederum Systemressourcen und verändern durch ihre Anwesenheit das Systemverhalten.

Auch auf Hardware kann zur Performance-Messung zurückgegriffen werden. Dabei handelt es sich um speziell für diesen Zweck integrierte Hardwarekomponenten, die Ereignisse auf anderen Komponenten aufzeichnen. Hardware- Messungen verursachen in der Regel wenig Störungen, geben also ein unverändertes Bild des Systems wieder. Allerdings sind nicht alle Geschehnisse für Hardware sichtbar. Entscheidender ist jedoch, dass zwar häufig die Wirkung beobachtbar ist, die Ursache dazu sich jedoch nicht durch Hardware identifizieren lässt.

Um diese Schwachstellen zu umgehen, werden Hybridsysteme, eine Mischung aus Hard- und Software, eingesetzt. Die Ursache wird durch Software registriert, die Wirkung durch Hardware.

2.2 Analysieren

Aufgrund der dabei entstehenden Datenmengen ist es nicht möglich, sämtliche potentiell zur Verfügung stehenden Informationen zu sammeln. Stattdessen beschränken sich Performance Tools auf Teilaspekte. Welche Leistungsmerkmale analysiert werden, muss bei den meisten Werkzeugen vor dem Programmstart oder sogar noch während der Erstellung des Programms statisch festgelegt werden. Dabei werden Zähler oder Trigger integriert, durch die die erforderlichen Daten

entnommen werden. Dies gilt beispielsweise für alle Werkzeuge, die den Quellcode, den Compiler, die Laufzeitbibliothek oder die Ausführungsdateien verändern.

Auf der anderen Seite existieren Werkzeuge, bei denen dynamisch während der Ausführung entschieden werden kann, welche Daten gesammelt werden. Je nach Bedarf kann der Benutzer das Sammeln einzelner Leistungsmerkmale aktivieren oder unterbinden.

Die gesammelten Daten werden nun zusammengefasst und aufbereitet, um anschließend in eine für den Benutzer verständliche Darstellung gebracht zu werden. Bei allen Werkzeugen stellt sich die Frage, welche Daten wichtig für das Verständnis des Systemverhaltens sind. Werden zu viele nebensächliche Informationen gesammelt, so wird das Systemverhalten verfälscht, sodass ein falsches Bild des Verhaltens abgelesen wird. Bei zu wenig gesammelten Daten kann aufgrund dieser kein begreifbares Abbild erzeugt werden.

Daher werden zumeist zu viele Informationen gesammelt und, nachdem die wichtigsten Merkmale identifiziert wurden, anschließend verfeinert. Wird auf statische Verfahren zurückgegriffen, so ist diese Vorgehensweise bei Grid Software, die eine lange Ausführungszeit besitzt, sehr zeitaufwendig.

2.3 Darstellen

Für die Darstellung der gesammelten Informationen existiert eine Vielzahl von Möglichkeiten. Eine dieser Möglichkeiten sind Metriken. Dabei wird eine Ressource mit einem Zahlenwert in Verbindung gebracht, beispielsweise die prozentuale Benutzung der CPU. Die Anwendungsmöglichkeiten von Metriken sind Vielseitig, neben der angesprochenen prozentualen CPU-Last lässt sich zum Beispiel auch die Ausführungszeit der Software ermitteln und mit kalkulierten Zeiten vergleichen. Dabei ist es häufig sinnvoll, sich mehr als eine Metrik anzeigen zu lassen, um die verschiedenen Einflussfaktoren zu ermitteln.

Neben Metriken, einfachen Zahlenwerten, bieten Werkzeuge häufig auch Graphen an. Aufgrund des zeitabhängigen Charakters offenbaren Graphen viele durch Metriken nicht in diesem Maße nachvollziehbare Zusammenhänge. Auch hier ist es hilfreich, mehrere Graphen in Verbindung zu bringen. Manche Werkzeuge bieten die Ansatzpunkte für neue Metriken und Graphen durch Kombination schon existierender Darstellungen oder durch das Bereitstellen einer Bibliothek, die genutzt werden kann.

Neben diesen beiden Varianten bieten manche Werkzeuge auch die Algorithmus-Animation an. Der Benutzer erstellt zu seinem Programm einen formalen Graphen, der das Verhalten der Software widerspiegelt. Anschließend legt er fest, welche Abschnitte der Software auf welchen Teil des Graphen abgebildet werden sollen. Bei der Darstellung durch das Performance-Werkzeug wird dadurch ersichtlich, wie viel Zeit in welchem Teil des Algorithmus verbracht wird. Allerdings fällt es zuweilen schwer, einen geeigneten formalisierten Graphen zu erstellen, und der Zeitaufwand ist ebenfalls signifikant.

Einen anderen Ansatz stellen die Search Based Tools dar. Bei diesen existieren zwei Varianten. In der ersten wird dem Benutzer direkte Hilfe bei der Verbesserung der Ausführungszeit angeboten. Das Werkzeug untersucht den Quellcode auf

unzweckmäßige Konstrukte und gibt alternative Implementierungen vor. Werkzeuge mit dieser Arbeitsweise sind sprachabhängig und bieten nur die Möglichkeit, ihnen bekannte Probleme zu verbessern. Die zweite Variante verlangt vom Programmierer, Zusicherungen zum Zeitverhalten einzelner Codeblöcke zu schreiben. Werden diese nicht eingehalten, so liegt offensichtlich ein Problem vor. Dieses Verfahren erfordert erfahrene Programmierer und ist aufgrund der zusätzlichen Arbeit sehr zeitaufwendig. Beide Varianten bieten nur eine grobe Granularität bei der Problemanalyse, die erste wegen der Anzahl bekannter Probleme, die zweite wegen der Anzahl manuell integrierter Zusicherungen.

Es existiert keine Darstellung, welche das Laufzeitverhalten jeder Software für jeden Benutzer optimal erläutern kann. Aus diesem Grund ist eine Palette unterschiedlicher Darstellungen hilfreich und Erfahrung seitens des Benutzers vonnöten, welche Darstellung das Verhalten der momentan untersuchten Software am zweckmäßigsten abbildet.

3 Paradyn

3.1 Überblick

Das Paradyn-Performance-Measurement-Werkzeug wird an der Universität von Wisconsin seit 1994 besonders für den Anwendungsbereich großer umfangreicher und komplexer paralleler Anwendungen entwickelt.

Es dient dem Auffinden von Bottlenecks, die das Laufzeitverhalten einer Anwendung entscheidend beeinflussen können und basiert auf den Theorien von K. Hollingsworth[1]. Seit Mai 2003 steht es nun in der Version 4.0 für Unix, Windows, Sun- und IBM-Plattformen zur Verfügung.

Der Quellcode ist zum freien Download unter http://www.cs.wisc.edu/~paradyn/ erhältlich und für rein wissenschaftliche Verwendungszwecke entstehen keine Lizenzkosten.

3.2 Prinzipien und Charakteristiken

Im Folgenden werden nun einige grundsätzliche Prinzipien vorgestellt, die während der Entwicklung von Paradyn eine wichtige Rolle spielten.

3.2.1 Skalierbarkeit

Paradyn muss in der Lage sein, Anwendungen zu messen, die über einen langen Zeitraum laufen und/oder auf großen parallelen Umgebungen ausgeführt werden. Daher muss es möglich sein, gezielt nur bestimmte Bereiche einer Anwendung zu instrumentieren und zu messen, bzw., die beobachteten Bereiche, während die Anwendung läuft, zu verändern, um die gesammelten Daten noch zur Laufzeit auswerten zu können.

3.2.2 Automatische Suche

Ziel von Paradyn ist es, die Teile eines Programms zu identifizieren, welche die meisten Ressourcen verbrauchen, und den Anwender genau zu diesen Abschnitten hinzuführen. Paradyn soll selber die Suche nach diesen laufzeitbeeinflussenden Bereichen übernehmen und dies nicht einzig dem Anwender überlassen. Mit Hilfe eines speziellen Suchmodells soll dies erreicht werden.

3.2.3 Heterogene Umgebungen

Parallele Rechenumgebungen reichen von Clustern von Workstations bis hin zu mächtigen parallelen Rechnern. Heterogenität erscheint in vielen Formen, beispielsweise als Rechnerarchitekturen oder Betriebssysteme. Durch Abstraktion der Eigenheiten jeder Umgebung kann die Portierung auf eine neue Plattform leicht erreicht werden.

3.2.4 Programmiersprachen-Unterstützung

Performance-Probleme, die auf der niedrigsten Software- und Hardwareebene auftreten, müssen durch die einzelnen Abstraktionsebenen nach oben befördert werden, damit ein Anwender mitgeteilt bekommt, wo in der Terminologie seiner Programmiersprache das Problem auftritt.

3.2.5 Offene Interfaces

Paradyn stellt ein einfaches Interface bereit, welches es erlaubt, neben den mitgelieferten Visualisierungen, wie Zeit-Histogramme, auch fremde Anzeigemöglichkeiten von anderen Anwendungen, wie zum Beispiel Pablo[4], zu importieren.

Genauso wichtig ist die Fähigkeit, neue Arten von Daten erheben zu können, wie zum Beispiel cache miss, Netzwerkverkehr oder Paging-Aktivitäten. Paradyns Instrumentierung ist durch einfache Konfigurationsdateien so konfigurierbar, dass im Prinzip jede Art von Performance-Daten erhoben werden könnten.

3.2.6 Einfache Anwendung

Es soll wenig Mühe bereiten, Paradyn zu installieren und zu bedienen. Ziel ist es, mit möglichst wenigen Handgriffen Paradyn einsatzbereit zu machen.

Das Überwachen von Anwendungen soll mit möglichst wenig Veränderung der zu überwachenden Anwendung im Vorfeld möglich sein. Durch dynamisches Instrumentieren ist ein Modifizieren der Anwendung im Voraus überflüssig. Es wird auch ermöglicht, die Instrumentierung selbst während einer laufenden Anwendung noch zu verändern.

3.3 Paradyns Bestandteile

Paradyn besteht aus einem Hauptprozess, einem oder mehreren Daemon- und beliebig vielen Visualisierungsprozessen.

Der Hauptprozess beinhaltet mehrere Threads, welche den Performance Consultant, Visualisation Manager, Data Manager und den User Interface Manager repräsentieren. Die Kommunikation wird mittels Interfaces ermöglicht. Die Abbildung 3 zeigt die Architektur von Paradyn.

Der Data Manager verarbeitet Datensammelanfragen von anderen Threads, pflegt diese und liefert Messdaten von den Paradyn Daemons zurück an die anfragenden Threads. Weiter verwaltet der Data Manager Informationen über Metriken und Ressourcen für eine derzeit überwachte Anwendung. Das User Interface verwendet Tcl/Tk und ermöglicht es dem Benutzer, graphisch gestützten Zugriff auf Paradyn selbst und die Messdaten zu erhalten. Der Performance Consultant überwacht die automatische Suche nach Performanceproblemen und fragt selbständig Daten beim Data Manger ab. Der Visualization Manager startet einen Visualisierungsprozess und ist für die Kommunikation zwischen den externen Visualisierungsprozessen und den Paradyn- Modulen verantwortlich.

Die Paradyn Daemons beinhalten die plattformabhängigen Fragmente von Paradyn. Der Instrumentation Manager implementiert die dynamische Instrumentierung. Die Daemons bearbeiten Anfragen nach Performancedaten der Anwendung von Paradyn. Sie kapseln die Plattformabhängigkeiten und verwenden ein Remote Procedure Call Interface und ermöglichen es so, auch heterogene Anwendungen zu bearbeiten.

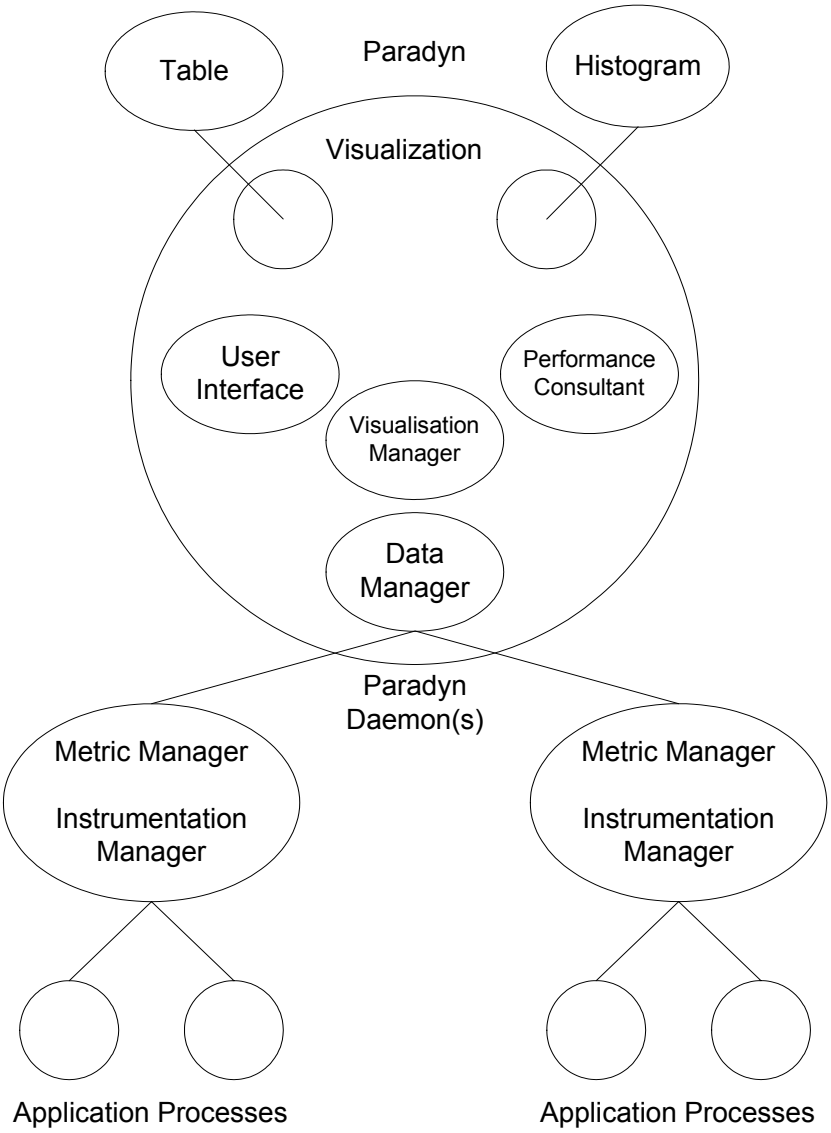


Abb. 3: Architektur von Paradyn [1, Seite 73]

4 Dynamische Instrumentierung

Paradyn verwendet dynamische Instrumentierung, um nur die Teile einer Anwendung zu instrumentieren, welche wichtig sind, um das aktuelle Performanceproblem zu finden. Dynamische Instrumentierung modifiziert eine Anwendung erst, wenn diese gestartet ist, und fügt neue Instrumentierung hinzu bzw. löscht diese wieder, während die zu überwachende Anwendung läuft.

4.1 Dyn. Instrumentierungs-Interface

Die Paradyn Daemons übersetzen Instrumentierungsanfragen in Programmanweisungen, welche in die laufende Anwendung eingefügt werden müssen. Dieses Übersetzen einer Anfrage in Programmcode erfolgt in zwei Schritten. Als erstes wird die Anfrage vom Metric Manager in eine abstrakte maschinenunabhängige Beschreibung des zu Messenden übersetzt. Diese Beschreibung nennt man Metrik. Anschließend übersetzt der Instrumentation Manager diese in eine plattformabhängige Maschinenanweisung. Counter und Timer sind die zwei Typen von Instrumentierungen, welche in eine Anwendung eingefügt werden können. Counter sind einfache Zähler, welche die Häufigkeit des Auftretens einer Anweisung zählen, während Timer die Zeit messen, welche die Ausführung einer bestimmten Aktion dauerte.

4.2 Points, Primitives und Predicates

Points, Primitives und Predicates stellen eine einfache, maschinenunabhängige Sammlung von Operationen dar, welche zusammen eine einzelne Instrumentierung ergeben.

Points sind Stellen innerhalb des Codes einer Anwendung, an denen das Einfügen von Instrumentierung überhaupt möglich und sinnvoll ist. Primitives sind einfache Operationen, die den Wert eines Counters oder Timers verändern können. Predicates sind boolesche Ausdrücke, die das Ausführen von Primitives ermöglichen oder verhindern.

Derzeit werden folgende Points von Paradyn unterstützt: Prozedur-Eingang, Prozedur-Ausgang und individuelle Call-Statements. Weiter stehen zur Zeit sechs Primitives zur Verfügung, das sind set counter, add to counter, substract from counter, set timer, start timer und stop.

Ein Beispiel wie, Predicates und Primivites zusammen eine Instrumentierung bilden, zeigt die folgende Abbildung 4.

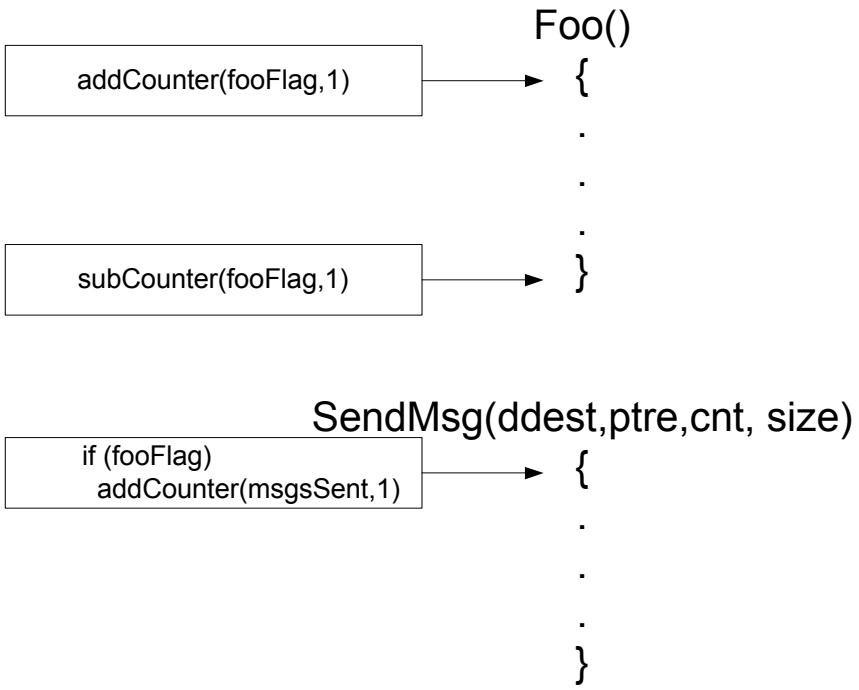


Abb. 4: Beispielinstrumentierung [1, Seite 42]

Wie eine Metrik in entsprechende Instrumentierung umgewandelt wird, ist durch eine Metrik-Definition definiert, welche in speziellen Konfigurationsdateien (PCL) von Paradyn enthalten sind. Diese PCLs können leicht verändert und erweitert werden, wodurch es möglich wird, weitere Metriken hinzuzufügen und Paradyn auch auf weitere Plattformen zu portieren. Die Definition geschieht mit Hilfe einer Pseudo-Programmiersprache.

4.3 Metrikdefinition

Eine Metrik ist eine Schablone, die beschreibt, wie das Sammeln von Daten für verschiedene Kombinationen von Ressourcen geschehen soll. Eine Metrik besteht aus einer Ansammlung von Codefragmenten, welche Primitives und Predicates erstellen, um die gewünschte Metrik zu erzeugen.

Damit sie möglichst modular und kompakt ist, wird eine Metrik in zwei Teile geteilt: eine base metric und eine Vielzahl von Ressourcenbeschränkungen (Constraint). Die base metric beschreibt, wie eine Metrik berechnet werden soll. Eine Ressourcenbeschränkung definiert, wie und wann eine Metrik für eine spezielle Ressource angewandt wird. Sie werden in der Regel in Predicates umgewandelt.

```
syncFuncs = { Barrier, Lock, Mutex };

metric syncOps {
  name "Synchronization Operations";
  units operationsPerSecond;
  foldOperator sum;
                aggregationOperator sum;
  constraint MSGTagPredicate;
  constraint processPredicate;
  constraint procedurePredicate;

  base is counter {
  foreach func in syncFuncs {
    append at func->entry constrained [
     addCounter(syncOps, 1);
    ]
  } }}


  constraint procedurePredicate /Procedure is
counter {
    foreach i in $constraint->calls {
     append at i->preCall [
      setCounter(procedurePredicate, 0);
     ]
     prepend at i->postCall [
      setCounter(procedurePredicate, 1);
     ]
    }

    append at $constraint->entry [
      setCounter(procedurePredicate, 1);
    ]
    prepend at $constraint->return [
     setCounter(procedurePredicate, 0);
    ]
  }
```

Abb. 5: Metrik- und Constraintdefinition [1, Seite 45]

4.4 Instrumentierungs-Generation

Der Instrumentation Manager kapselt das Wissen um die architekturspezifische Implementation der Instrumentierungen. Er durchsucht die Anwendung nach geeigneten Instrumentierungspunkten. Dabei werden Prozedureinstiegs- und

-ausstiegspunkte sowie Prozeduraufrufe als geeignete Punkte markiert und gespeichert. Der Instrumentation Manager führt auch die finale Übersetzung einer Metrik und der dazugehörigen Instrumentierung in Maschinensprache durch.

Wenn Paradyn gestartet ist und eine Anfrage gestellt wird, etwas zu beobachten, so wird diese Anfrage an einen Paradyn Daemon geleitet. Der Metrik Manager ermittelt die entsprechende Metrik und leitet diese an den Instrumentierungs-Manager weiter, welcher nun die Metrik in kleine Codefragmente übersetzt, sogenannte Trampolines, und diese in das Programm einfügt. Man unterscheidet zwei Arten von Trampolines, base trampolines und minitrampolines.

Für jeden Point mit aktiver Instrumentierung wird eine base trampoline in das Programm eingefügt. Dies geschieht dadurch, dass die Maschinenanweisung an dieser Stelle durch einen Sprung (branch) zu dieser base trampoline ersetzt wird.

Eine base trampoline besitzt „Schlitze“ (slots), in denen mini-trampolines aufgerufen werden können. Diese slots befinden sich sowohl vor als auch nach der umpositionierten ursprünglichen Anweisung. Mini-trampolines beinhalten den Code, um ein bestimmtes predicate auszuwerten oder ein einzelnes primitive zu evaluieren. Der Einsatz von mini-trampolines erfordert die Erzeugung von korrekten Maschinenanweisungen, welche vom Instrumentation Manager erzeugt und dann in die Anwendung mit Hilfe einer Variation des UNIX ptrace-Interfaces transferiert werden. Bei dem Transfer muss natürlich auch das korrekte Speichern und Laden der betroffenen Register gewährleistet sein.

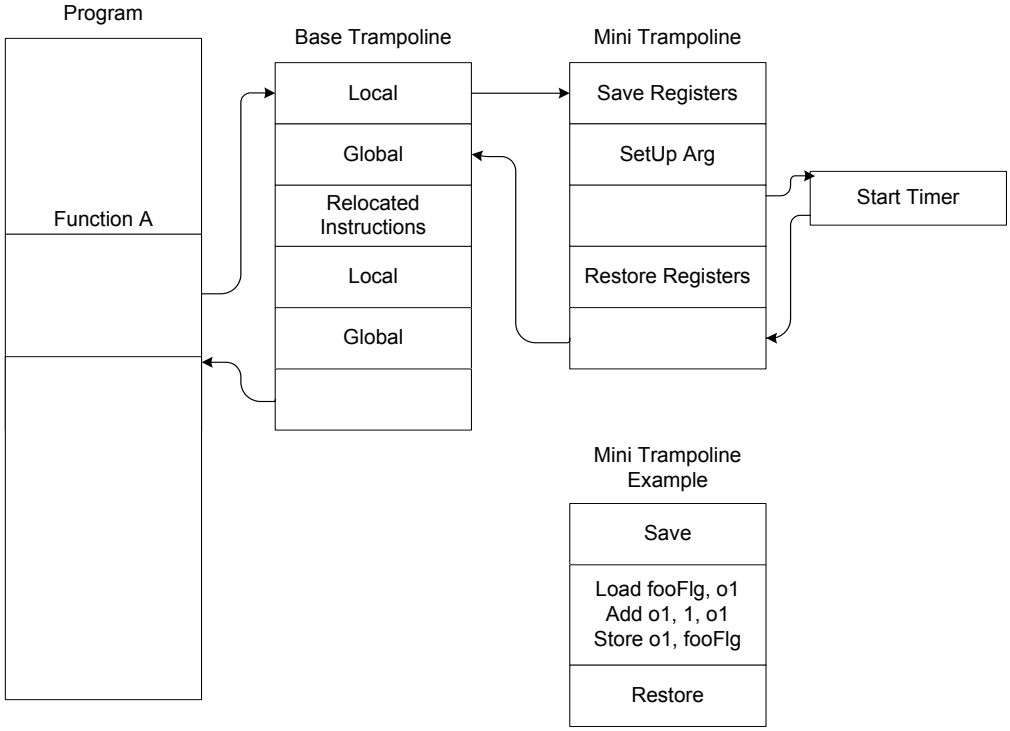


Abb. 6: Einfügen von Instrumentierung in ein Programm [1, Seite 48-49]

4.5 Sammeln von Daten

Nachdem die Instrumentierung erfolgt ist, fließen die gesammelten Daten zurück

an den Benutzer. Periodisch werden die Werte der Counter und Timer vom Paradyn Deamon an den Data Manager zurückgeliefert.

Da Paradyn Messungen an Hand einfacher Counter und Timer vornimmt, ist es leicht, auch externe Informationsquellen zu verwenden. So liefern viele Betriebssysteme bereits Informationen, zum Beispiel über virtuellen Speicher und CPU-Last. Viele Maschinen besitzen auch Counter auf Hardwarebasis, welche ebenfalls benutzt werden können.

4.6 Interne Nutzung von dynamischer Instrumentation

Unter dem Entdecken neuer Ressourcen versteht man die Bestimmung, welche Ressourcen von einer Anwendung benutzt werden, und das Abbilden dieses Wissens auf einen hierarchischen Baum, den Ressource Tree.

Der Großteil dieses Baumes kann bereits aufgebaut werden, wenn die Anwendung, welche überwacht werden soll, Paradyn bekannt gegeben wird. Allerdings gibt es einige Ressourcen und Informationen, die erst zur Laufzeit bekannt sind, wie zum Beispiel welche Dateien verwendet werden und wann diese gelesen und geschrieben werden. Um diese Ressourcen ausfindig zu machen, wird ebenfalls dynamische Instrumentierung verwendet. So wird zum Beispiel beobachtet, welche und wie viele Dateien geöffnet sind.

5 W3-Suchmodell & Performance Consultant

Wichtigstes Ziel von Paradyn ist es, dem Benutzer zu helfen, Performanceprobleme in einer Anwendung ausfindig zu machen. Paradyn verwendet für diese Suche ein wohldefiniertes Suchmodell. Die Suche wickelt der Performance Consultant mit Hilfe der gesammelten Daten der dynamischen Instrumentierung ab.

5.1 Das W^3-Suchmodell

Das W^3-Suchmodell abstrahiert die Aspekte einer parallelen Anwendung, die die Performance beeinflussen können, dadurch, dass im Grunde drei einfache Fragen gestellt werden:

- **Wieso** läuft meine Anwendung so langsam?
- **Wo** tritt das Performanceproblem auf?
- **Wann** genau geschieht dies?

Die Frage nach dem Wieso wird mit Hilfe so genannter Hypothesen beantwortet, auf die die Anwendung getestet wird. Die Frage nach dem Wo ermöglicht uns, genau zu isolieren, welche Ressource, zum Beispiel der Speicher, eine bestimmte Prozedur oder ein Synchronisationsobjekt das Problem verursacht. Um zu erfahren, wann das Problem auftritt, wird versucht, das Problem auf einen bestimmten Abschnitt der Programmabwicklung festzulegen.

Die Suche nach diesen Problemen ist ein iterativer Prozess, in dem nach und nach die Fragen beantwortet werden.

5.1.1 Die „Why"-Achse

Die Frage nach dem Wieso wird mit Hilfe der Why-Achse beantwortet.

Die Why-Achse repräsentiert die breite Palette an typischen Problemen, die in einer parallelen Anwendung auftreten können. Potentielle Probleme werden als so genannte Hypothesen und Tests dargestellt. Hypothesen sind die fundamentalen Typen von Performanceproblemen, welche generell in parallelen Anwendungen auftreten können, unabhängig von der betrachteten Anwendung. Zum Beispiel kann eine Hypothese sein, dass wir ein Synchronisationsproblem haben. Eine Handvoll dieser Hypothesen reicht oft aus, um die typischen Probleme solcher Anwendungen zu beschreiben.

Hypothesen bilden eine Art Hierarchie, so wird eine „allgemeine" Hypothese, wenn sie sich als wahr herausstellt, nach und nach verfeinert. Diese Hierarchie bildet einen azyklischen Suchgraphen, welcher traversiert wird. Die Abbildung 7 zeigt einen solchen Graphen.

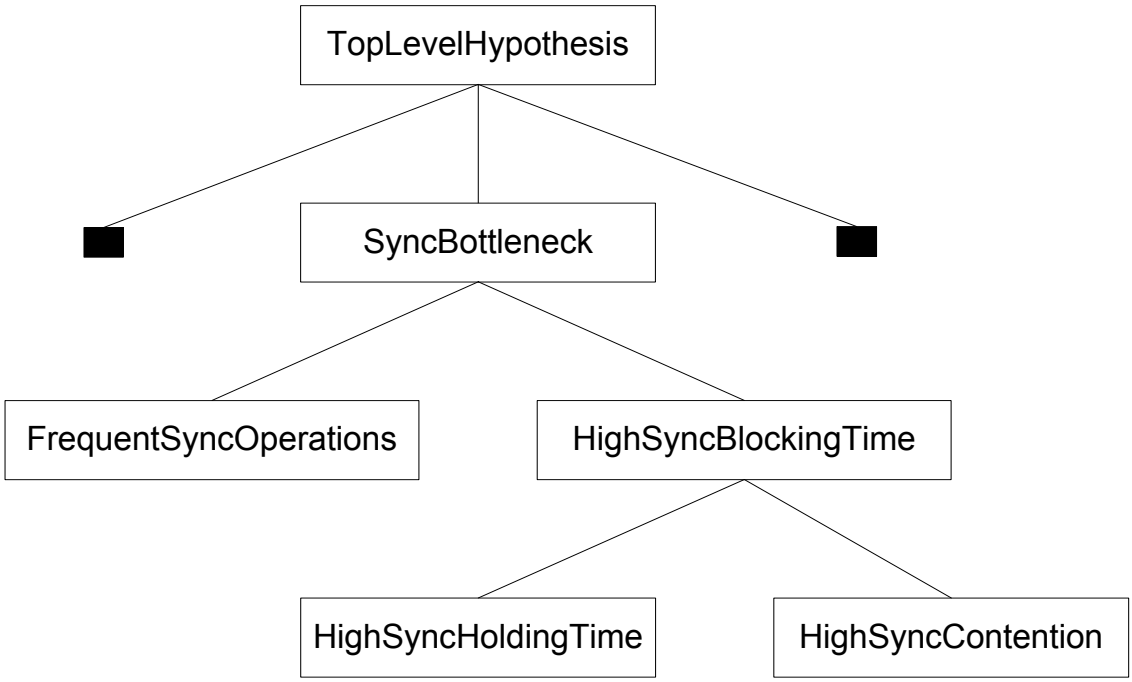


Abb. 7: Die „Why"-Achse [1, Seite 18]

Tests sind boolesche Funktionen, die evaluieren, ob eine Hypothese zutrifft. Dabei wird geprüft, ob für diese Hypothese ein bestimmter Schwellenwert überschritten wird. Am Beispiel der Synchronisation wird evaluiert, ob die Anwendung zum Beispiel 20% ihrer Laufzeit mit dem Warten auf Synchronisationsobjekte verbringt. Ist dies der Fall, gilt die Hypothese als wahr und wird nun verfeinert.

5.1.2 Die “Where”-Achse

Die zweite Frage, die beantwortet werden muss, ist die Frage, wo meine Anwendung das Problem verursacht.

Das Suchen auf der Where-Achse zeigt auf, welche Anwendungskomponente der Verursacher ist. Beim Beispiel der Synchronisation könnte ein Synchronisationsobjekt unter Hunderten das Problem verursachen. Wieder wird iterativ durch den Graphen, der die Where-Achse darstellt, gesucht. So wird vom allgemeinen Synchronisationsobjekt die Suche verfeinert, bis genau das verursachende Objekt gefunden worden ist.

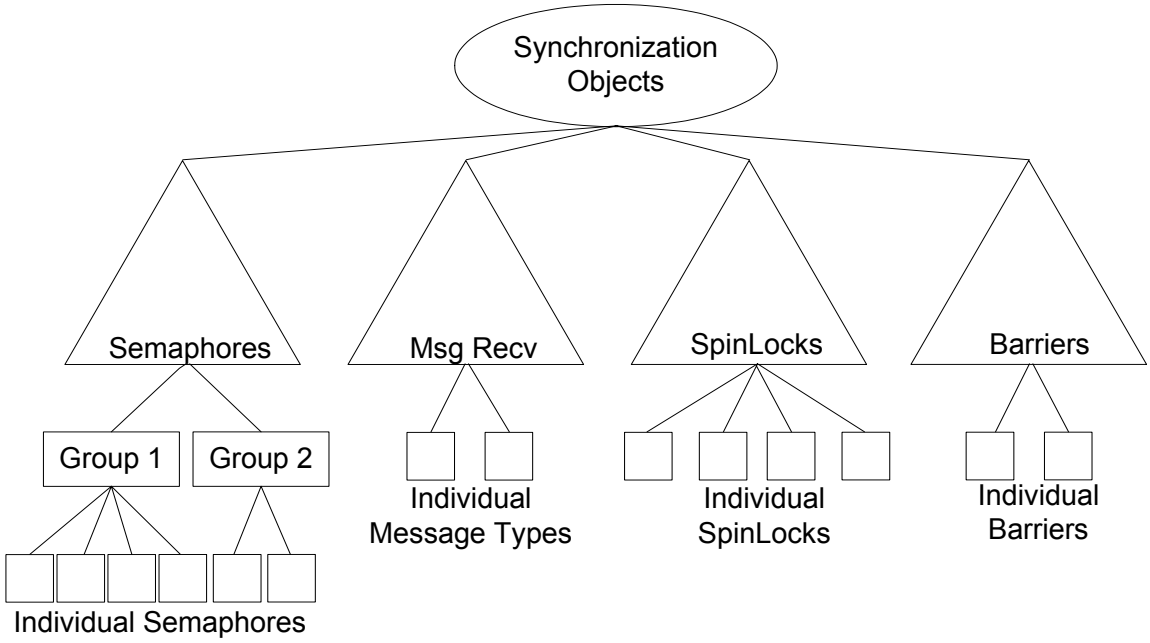


Abb. 8: Die „Where“-Achse [1, Seite 20]

5.1.3 Die “When”-Achse

Die dritte Frage lautet: „Wann genau in meiner Anwendung tritt das Problem auf?“

Anwendungen haben getrennte Phasen der Ausführung, und die When-Achse zeigt einen zeitlichen Verlauf, in dem zu unterschiedlichen Zeitpunkten verschiedene Performance-Probleme auftreten können. So kann zum Beispiel ein einfaches Programm drei Phasen aufweisen, eine Initialisierungsphase, eine Berechnungsphase und eine Phase, in der die Ergebnisse ausgegeben werden.

Innerhalb jeder einzelnen Phase verändert sich das Performanceverhalten nur unwesentlich, allerdings kann der Unterschied zu anderen Phasen erheblich sein, da sich das Verhalten einer Anwendung komplett ändern kann. Die Suche entlang der When-Achse beschreibt also den Test der Hypothesen zu verschiedenen Phasen der Programmausführung.

5.2 Der Performance Consultant

Das Performance Consultant Module von Paradyn spürt Performanceprobleme mit Hilfe des W^3-Suchmodells auf. Die Fähigkeit, automatisch nach Problemen zu

suchen, ist die wichtigste Funktionalität des Performance Consultant. Die Suche bzw. das Verfeinern der Suche entlang der drei Achsen erfolgt ohne das Zutun des Benutzers.

Die möglichen Verfeinerungen erfolgen mit Hilfe von vordefinierten geordneten Listen, über die der Performance Consultant verfügt und die auf die einzelnen Objekte auf diesen Achsen angewandt werden. Nach und nach werden diese Listen abgearbeitet; allerdings hat der Anwender die Möglichkeit, selber Verfeinerungen vorzunehmen, um so die Suche in eine bestimmte Richtung zu lenken. Berichterstattung über die aktuell laufende Suche und darüber, welche Ergebnisse bis jetzt vorliegen und was gerade getestet wird, liefert der Search History Graph(SHG).

Der SHG speichert die Ergebnisse der Suche entlang der Achsen, wobei jeder Knoten des Baumes einen einzelnen Schritt in dieser Suche darstellt und die Farbe Auskunft über den aktuellen Zustand des Knotens liefert. So bedeutet Pink zum Beispiel, dass der Knoten gerade getestet wird. Die Farben der Kanten geben an, auf welcher Achse die Verfeinerung stattfindet.

6 Schlussfolgerung

Das parallel-performance-measurement-Werkzeug Paradyn, vereinigt einige moderne Ansätze zum Auffinden von Performance Problemen.

Das dynamische Instrumentieren bietet die Möglichkeit, die Messkosten erheblich zu senken, da nur zeitweise gezielt Daten gesammelt werden, während das W^3-Suchmodell, welches im Performance Consultant integriert ist, die dynamische Instrumentierung beeinflusst. Das Zusammenwirken dieser beiden Technologien ermöglicht die automatische Suche nach Problemen, wobei Laufzeit und der Umfang einer verteilten Anwendung dabei keine Rolle spielen.

Weiter liefert Paradyn detaillierte Informationen auch über zeitliche Aspekte des Auftretens von Problemen und ist in der Lage, die auf Maschinenebene gefunden Probleme, so auf die Hochsprache abzubilden, in der die Anwendung geschrieben wurde, sodass der Anwender die Probleme an Hand seines eigenen Codes aufgezeigt bekommt.

Der modulare Aufbau und die Verwendung von Interfaces erlauben ein leichtes Hinzufügen eigener Metriken und das Portieren von Paradyn auf neue Plattformen und Architekturen.

In der Praxis offenbaren sich jedoch auch einige Schwächen. So läuft das Programm zuweilen instabil und die Kompatibilität scheint nicht mit sämtlichen Sprachkonstrukten gewährleistet zu sein, sodass man Paradyn zur Unterstützung der Optimierung einer parallelen Anwendung zwar nutzen kann, jedoch vorher prüfen sollte, ob die verwendete Sprache unterstützt wird und testen, ob die Verwendung von Paradyn bereits in frühen Phasen der Entwicklung gelingt. Eventuell bedarf es sonst einem Ausweichen auf alternative Performance Debugger oder manuelles Prüfen.

Quellen

[1] J. K. Hollingsworth: „Finding bottelnecks in large scale parallel programs“, 1994
[2] Paradyn Project: „Paradyn Parallel Performance Tools“, http://www.cs.wisc.edu/~paradyn/
[3] Ion Foster, Carl Kesselman: „The Grid: Blueprint for a New Computing Infrastructure“, 1998
[4] University of Illinois Pablo Research Group, http://www-pablo.cs.uiuc.edu/

Grid - Applikationen in der Praxis

Steffen Bach, Michael Blume, Helge Issel

{steffen.bach|michael.blume|helge.issel}@hpi.uni-potsdam.de

Das Ziel dieser Ausarbeitung ist es, die praktische Relevanz des Themas Grid zu beleuchten. Das Augenmerk liegt auf wirklich vorhandenen industriellen und wissenschaftlichen Anwendungen, die auf grid-ähnlichen Infrastrukturen basieren. Dazu wurden mögliche Applikationen in biowissenschaftlichen, ingenieursbasierten, datenorientierten, physikalischen und kommerziellen Anwendungen unterteilt. Aus jeder dieser Kategorie wurden ein bis zwei bedeutende Projekte ausgewählt und untersucht.

Inhaltsverzeichnis

1 Einleitung 3
2 UK e-Science 4
2.1 Einführung 4
2.2 UK e-Science Programm 4
2.3 Pilotprojekte 4
2.4 Kernprogramm 7
3 Biowissenschaftliche Anwendungen 12
3.1 BIRN 12
3.2 MCell 15
3.3 myGrid 18
4 Ingenieursbasierte Anwendungen 20
4.1 NASA IPG 20
4.2 GEODISE 23
5 Datenorientierte Grid-Applikationen 26
5.1 DAME 26
6 Physikalische Grid-Applikationen 29
6.1 EU AVO / NVO / UK AstroGrid 29
7 Kommerzielle Anwendungen 35
7.1 On-Demand Services 35
7.2 Sun Grid Engine 36
8 ClimatePrediction.net 39
9 Zusammenfassung 40
10 Quellen 41

1 Einleitung

Diese Ausarbeitung erfolgt im Rahmen des Seminars Grid Computing des Hasso-Plattner-Institutes und der Universität Potsdam. Im Verlaufe dieses Seminars wurden der Begriff Grid und eine große Anzahl damit in Zusammenhang stehender Technologien und Entwicklungen erläutert und untersucht. Das Ziel dieser Ausarbeitung ist es, die praktische Relevanz des Themas Grid zu beleuchten. Aus diesem Grund setzen wir voraus, dass ein Leser dieses Dokumentes schon ausreichend Erfahrung und Wissen im Gebiet Grid besitzt und die relevante Terminologie in genügendem Umfang beherrscht, so dass wir auf eine gesonderte Einführung in das Thema Grid an dieser Stelle verzichten. Vielmehr soll das Augenmerk auf wirklich vorhandene industrielle und wissenschaftliche Anwendungen, die auf grid-ähnlicher Infrastruktur basieren, gelegt werden. Dazu wurden mögliche Anwendungsgebiete in fünf Kategorien unterteilt, aus denen wir jeweils 1 bis 2 bedeutende Projekte ausgewählt haben, die wir im Folgenden untersuchen.

2 UK e-Science

2.1 Einführung

Als allererstes wird hier das UK e-Science[1] näher beleuchtet, welches mehrere Pilotprojekte aus fast allen Gebieten beinhaltet und deswegen keiner Kategorie klar zugeordnet werden kann.

Der Begriff „e-Science“ wurde von Dr. John Taylor (Generaldirektor aller Research Councils des “UK Office of Science and Technology” (OST)[2]) eingeführt. Die Original-Definition lautet:

„e-Science is about global collaboration in key areas of science and the next generation of infrastructure that will enable it.”

Bei e-Science dreht es sich also um eine globale Zusammenarbeit in Kerngebieten der Forschung und die dazu notwendige neue Infrastruktur. Als Infrastruktur hierfür dient das bereits mehrfach erwähnte Grid.

2.2 UK e-Science Programm

Die über 170 Millionen € schwere e-Science Initiative überspannt alle Research Councils des OST, wobei jedes einzelne einen bestimmten Anteil an dieser Summe erhielt. Der Löwenanteil (rund 26 Millionen €) ging dabei an das Forschungszentrum für Partikelphysik und Astronomie, um die notwendige Infrastruktur für die 2005 beginnenden LHC („Large Hadron Collider“)[3] – Experimente bereit zu stellen. Auch die zentralen Testlaboratorien in Daresbury und Rutherford (CLRC – „Central Laboratory of the Reseach Councils”)[4] bekamen 7 Millionen € um ihre Experimente grid-fähig zu machen. Die doppelte Summe, also 14 Millionen € wurde in die Entwicklung eines neuen nationalen Computersystems mit einer Rechenleistung von 7 Teraflop gesteckt. Schließlich wurden 50 Millionen € für das so genannte Kernprogramm („Core Programme“) veranschlagt.

2.3 Pilotprojekte

Wie bereits erwähnt, hat jedes der Research Councils eine eigene Anzahl an Projekten, die es mit der ihm zugedachten Summe unterstützt. Bedingt durch die unterschiedlichen Auswahlverfahren innerhalb der einzelnen Research Councils wurden die 20 aktuellen Projekte zu unterschiedlichen Zeiten gestartet. Unter den ersten Realisierungen waren 2 Projekte des schon erwähnten Forschungszentrums für Partikelphysik und Astronomie (PPARC – Particle Physic and Astronomy

[1] http://www.escience-grid.org.uk/
[2] http://www.ost.gov.uk/
[3] http://lhc-new-homepage.web.cern.ch/
[4] http://www.clrc.ac.uk/

Research Council)[5] sowie 6 Projekte des Forschungszentrums für technische und physikalische Wissenschaften (EPSRC – Engineering and Physical Sciences Research Council)[6], auf die wir hier kurz und bei Einigen später genauer eingehen werden.

2.3.1 RealityGrid

Dieses Projekt, welches durch ein Universitätskonsortium (inklusive Edinburgh, Loughborough, Manchester und Oxford) unter Führung von Prof. Peter Coveney geleitet wird und dessen industrielle Partner AVS, SGI und Fujitsu sind, beschäftigt sich mit der realistischen Modellierung komplexer, verdichteter Materie auf molekularer Ebene und der Erforschung neuer Materialien. Besonders kritisch für dieses Projekt ist die Integration von „high performance computing“ und Visualisierungseinrichtungen, wodurch eine künstliche Umgebung für die Modellierung von Problemen bereitgestellt wird. Die Ergebnisse solcher Modellierungen werden mit experimentellen Daten verglichen und ausgewertet.

2.3.2 Comb-e-Chem

Das Comb-e-Chem[7] – Projekt beschäftigt sich mit der Synthese von neuen Präparaten / Materialien mittels kombinatorischer Methoden. Es ist ebenfalls eine Kollaboration zwischen mehreren Universitäten (Southampton und Bristol), dessen wissenschaftlicher Leiter Dr. Jeremy Frey ist. Die industriellen Partner sind unter anderem Pfizer und IBM.

Kombinatorische Methoden eröffnen neue Möglichkeiten in einer Zeit, in der sich auch die Chemie mit großen Mengen an Wissen, und demzufolge mit einer großen Anzahl an Daten beschäftigt. In diesem Zusammenhang muss eine große Breite von Primärdaten akkumuliert und integriert werden, damit Relationsmodelle für maximale Effektivität erstellt werden können. Das Projekt hat die Entwicklung einer integrierten Plattform zum Ziel, die existierende Strukturen und Datenquellen mittels einer grid-basierten informations- und wissensverbreitenden Umwelt kombinieren.

Die erste Anforderung an diese Plattform ist die Unterstützung der Sammlung weiterer, neuer Daten, wobei sowohl Prozess- als auch Produktdaten basierend auf einer Integration von elektronischen Laboratorien und so genannten „e-logbook facilities“ in Betracht kommen. Der nächste Schritt ist das Einbinden von Methoden, so dass Daten „on-demand“ mittels einer grid-basierten Massen- und Simulationsmodellierung generiert werden.

Damit diese Umgebung auch der breiten Masse zur Verfügung gestellt werden kann, ist es natürlich notwendig, Schnittstellen zu entwickeln, die einen einheitlichen Blick auf die Ressourcen sowie einen transparenten Zugriff auf den Erhalt der Daten, die Online-Modellierung und das Design von Experimenten ermöglichen. Die dafür benötigte dienstbasierte Gridinfrastruktur wird sowohl auf die Geräte in den Laboratorien als auch auf die Datenbanken und Computerressourcen ausgedehnt.

[5] http://www.pparc.ac.uk/
[6] http://www.epsrc.ac.uk/
[7] http://www.combechem.org/

2.3.3 DAME : Distributed Aircraft Maintenance Environment

Bei diesem Projekt sind unter anderem die Universitäten von York, Oxford, Sheffield und Leeds unter der Leitung von Prof. Jim Austin beteiligt. In Zusammenarbeit mit Rolls Royce und Data Systems and Solutions zielt dieses Projekt auf die Entwicklung eines grid-basierten Diagnosesystems für Flugzeuge ab. Es wird im Kapitel DAME auf Seite 26 noch im Detail vorgestellt.

2.3.4 GEODISE: Grid Enabled Optimisation and DesIgn Search for Engineering

GEODISE ist eine Kollaboration der Universitäten von Southampton, Oxford und Manchester und der Industriepartner BAE Systems, Rolls Royce und Fluent. Ziel ist der grid-basierte Zugriff auf ein intelligentes Wissensarchiv, einer state-of-the-art Kollektion von Optimierungs- und Suchwerkzeugen, industrielle Stärkeanalysecodes und verteilten Berechnungs- und Datenressourcen. Dieses Projekt wird ebenfalls auf Seite 23 ausführlich erläutert.

2.3.5 Discovery Net

Dieses Projekt[8] hat im Vergleich zu anderen Projekten andere Ziele, da es sich auf den Punkt des “high throughput Computing” fokussiert. Im Kontext dieses Projektes sollen mit Hilfe einer entsprechenden Infrastruktur zeitkritische Meßdaten in Echtzeit berechnet, interpretiert, visualisiert und archiviert werden. Hierbei wird auch auf neue Technologien gesetzt, unter anderem auf Biochips bei der Biologie, high-throughput Screening-Technologien in der Biochemie und kombinatorischen Chemie sowie high-throughput Sensoren bei der Energie- und Umweltforschung. Eine ganze Reihe von Anwendungsszenarien wurden bei diesem Projekt berücksichtigt, zum Beispiel Daten zur Vorhersage von Erdbeben. Die Kollaboration besteht hier zwischen dem Imperial College London unter der Leitung von Dr. Yike Guo und den Industriepartnern Infosense Ltd, Deltadot Ltd und Rvca Inc.

2.3.6 myGrid

Hier steckt ein größeres Konsortium dahinter, bestehend aus GSK, AstraZeneca, IBM, und SUN in Zusammenarbeit mit den Universitäten von Manchester, Southampton, Nottingham, Newcastle und Sheffield, dem Europäischen Institut für Bioinformatik. Das Ziel besteht im Design und der Entwicklung einer Funktionalitätsschicht zur Unterstützung von Wissenschaftlern und Forschern, die komplexe, verteilte Ressourcen benutzen, wobei diese Schicht über der existierenden Gridinfrastruktur sitzt. Das myGrid-Projekt ist eines von nur sehr wenigen „OGSA-DAI early adopter“. Weiterführende Informationen über das Projekt folgen ab Seite 18.

[8] http://www.discovery-on-the.net/

2.3.7 GridPP

Die bisher vorgestellten Projekte kamen alle aus dem Forschungsbereich des ESPRC, GridPP[9] und AstroGrid sind jedoch Projekte des bereits erwähnten PPARC. Dabei ist GridPP eine Kollaboration von Partikelphysikern und Computerwissenschaftlern aus ganz Großbritannien und dem CERN-Institut. Das Ziel ist die Erstellung eines UK-Grids, sowie die Bereitstellung von Grid-Middleware und –hardwareinfrastruktur für die Large Hadron Collider (LHC)-Experimente am CERN-Institut. Diese Experimente sollen 2005 im vollen Umfang beginnen.

2.3.8 AstroGrid

Das AstroGrid-Projekt[10] war eines der ersten Programme innerhalb des UK e-Science. Es stellt sich den Bau einer Gridinfrastruktur zur Erstellung eines virtuellen Observatoriums („Virtual Observatory“) zum Ziel und ist ebenfalls einer der OGSA-DAI early adopter. Dabei ist es nur ein Teil eines globalen virtuellen Observatoriums und wird ab Seite 29 noch einmal gesondert erläutert.

2.4 Kernprogramm

Das Ziel des e-Science Kernprogramms ist es, die verschiedenen Anforderungen an die Grid-Middleware, die von den einzelnen Projekten kommen, zu identifizieren. In Zusammenarbeit mit Wissenschaftlern aller Fachrichtungen, Computerexperten und der Industrie besteht somit die Aufgabe, eine robuste Grid-Middleware zu entwickeln, die nicht nur die einzelnen Anwendungsgebiete untermauert, sondern auch für Industrie und Wirtschaft von Bedeutung sein soll. Das Kernprogramm besteht aus 6 Kernelementen, die hier näher beleuchtet werden sollen.

2.4.1 Implementation eines nationalen e-Science Grid Testbeds, basierend auf einem Netzwerk von e-Science Zentren

Momentan gibt es neun e-Science Zentren (siehe Abbildung 1).

Ein nationales e-Science Zentrum wurde in Zusammenarbeit der Universitäten Glasgow und Edinburgh in Edinburgh errichtet. Acht weitere regionale Zentren wurden in Belfast, Cardiff, Manchester, Newcastle, Oxford, Cambridge, London und in Southampton aufgebaut, wodurch ein Großteil von Großbritannien abgedeckt ist.

Diese Zentren haben 3 Aufgabengebiete:

- Allokieren von substantiellen Rechen- und Datenressourcen und deren Verbindung mit einem Standardsatz von Grid-Middleware, um die Basis für die Konstruktion des UK e-Science Grid Testbeds zu bilden.

[9] http://www.gridpp.ac.uk/
[10] http://www.astrogrid.org/

- Erstellung eines Portfolios von ähnlichen industriellen Grid-Middleware und -Werkzeugprojekten.
- Verbreiten von Informationen und Erfahrungen mit dem Grid innerhalb ihrer lokalen Region.

Abbildung 1: e-Science Standorte

Ebenfalls involviert in diesen Punkt sind die Laboratorien des CLRC in Rutherford und Daresbury, sowie das europäische Institut für Bioinformatik in Hinxton, wodurch eine gewaltige Menge an bioinformatischen Daten dem e-Science zur Verfügung stehen. In jedem dieser Zentren gibt es einen AccessGrid – Knoten, damit Interaktionen zwischen den einzelnen Zentren erheblich einfacher und innovativer gestaltet werden können.

2.4.2 Förderung der Entwicklung von generischer Grid-Middleware

Das e-Science Grid wird als Testbed in zweierlei Hinsicht gebraucht:

Die initiale Middleware hier ist dieselbe wie bei der NASA in ihrem IPG (siehe Seite 20). Bei der NASA allerdings könnte man das ganze als ein „Intra-Grid“ bezeichnen, da alle Laboratorien, die damit verbunden sind, Teil ein- und der selben Organisation mit einheitlichen Sicherheitsrichtlinien sind. Beim UK e-Science Grid werden aber verschiedene Universitäten mit sehr unterschiedlichen Policies, Firewalls usw. verbunden. Also ist dies ein sehr guter Test der grundlegenden Globus-Infrastruktur und dem digitalen zertifikatsbasiertem Sicherheitssystem.

Weiterhin stellen die e-Science Zentren eine heterogene Masse von Ressourcen für das Grid zur Verfügung, inklusive Supercomputer, Cluster und verschiedenen Datenspeicherungssystemen. Dieses Grid kann also auch als eine Testplattform für neue Grid-Middleware und als Ressource für neue industrielle Projekte dienen.

Das Kernprogramm steht zurzeit mit einer Reihe von IT-Firmen wie IBM, Sun, Oracle und Microsoft, sowie mit Globus, Condor und SRB in Verbindung, um die

zukünftige Entwicklung von Grid-Middleware zu diskutieren. In diesem Zusammenhang ist es erwähnenswert, dass sowohl IBM als auch Sun stark mit dem Globusteam zusammenarbeiten, um eine neue Generation von verbesserter und robusterer Grid-Middleware zu entwickeln.

Wie schon erwähnt, ist es eine der Hauptaufgaben des Kernprogramms, alle Anforderungen an die Grid-Infrastruktur von den einzelnen Projekten aufzufangen und auszuwerten. Dies schließt sowohl Datenspeicherungs- und Netzwerkanforderungen, als auch gewünschte Funktionalität an Grid-Middleware mit ein. Damit die einzelnen Projekte nicht ihre Energien damit verschwenden, getrennt und jedes für sich die einzelnen Technologien mehrfach zu untersuchen, gibt es eine Reihe von öffentlich zugänglichen Papern und Reports über den momentanen Status von Grid-Middleware. Zurzeit gibt es Reports über Globus, SRB/Datenbanken und .NET, wobei eine Analyse der Sun Grid Engine gerade in der Entwicklung ist.

Für die Forschung an Grid-Middleware wurden zwei Taskforces eingerichtet:

Die Grid DataBase Task Force (DBTF)[11] beschäftigt sich mit den Schnittstellen von Grid-Middleware und relationalen Datenbankmanagementsystemen. Dabei gibt es sowohl ein kurzfristiges Ziel – die Entwicklung einer Schnittstelle mit minimaler Funktionalität – sowie ein langfristiges Ziel – der Blick auf Forschungsgebiete hinter flachen Dateien und relationalen Daten.

Zusätzlich gibt es auch eine Grid Architecture Task Force (ATF)[12], welche sich um die generelle Richtung der Grid-Architektur kümmert. Die ersten Ideen deuten auf eine Implementation von Web Services ähnlichen „Grid Services“ hin. Dies führt zu dem Gedanken der Grid-Middleware als eine dienstorientierte Architektur, bei der Grid-Dienste von höherschichtigen Anwendungen konsumiert werden. In diesem Zusammenhang beschäftigt sich die ATF intensiv mit OGSA und OGSA-DAI.

Reporte beider Task Forces werden als „White Papers“ an das Global Grid Forum für weitere Diskussionen geleitet, wobei erwartet wird, dass in naher Zukunft lauffähige und einsetzbare „Open Grid Service“ Protokolle mit spezifizierten Schnittstellen zu Betriebssystemen und relationalen Datenbankmanagement-systemen entstehen werden. Dabei wird versucht, intensiv mit dem Globusteam zusammen zu arbeiten, um möglichst schnell zu ersten Ergebnissen zu gelangen.

2.4.3 Interdisciplinary Research Collaboration (IRC) Grid Projekte

Das EPSRC hat insgesamt 4 Projekte gestartet, bei denen es vorrangig um die langfristige Zusammenarbeit von Wissenschaftlern und Forschern aus mehreren Disziplinen geht:

- Das Equator Projekt aus Nottingham beschäftigt sich mit technologischer Innovation im physischen und digitalen Leben

[11] http://www.cs.man.ac.uk/grid-db/dbtf.html
[12] http://www.nesc.ac.uk/teams/atf.html

- Beim Advanced Knowledge Technologies Projekt (AKT)[13] dreht es sich um das Management des Lebenszyklus von Wissen.
- Dem DIRC Projekt[14], bei dem es um die Zuverlässigkeit von Computer-basierten Systemen geht.
- Und schließlich in Zusammenarbeit mit dem Medical Research Council (MRC)[15] das MIAS Projekt[16], das sich um die Übersetzung von Daten in Form von medizinischen Bildern und Signalen in brauchbare klinische Informationen für Mediziner kümmert.

2.4.4 Erstellung einer Support-Struktur für e-Science Projekte

Damit die einzelnen Projekte nicht allein dastehen, wurde ein Grid Support Centre errichtet, welches vom Kernprogramm und dem PPARC zusammen betrieben wird. In diesem Zentrum gibt es ein UK „Grid Starter Kit“, welches das Globus Toolkit, Condor und die SRB Middleware enthält. Es ist neben dem Support für bestehende und zukünftige Projekte auch für die Evaluation und Zusammenstellung zukünftiger Grid Starter Kits verantwortlich.

2.4.5 Unterstützung für internationale Aktivitäten

Es ist für das e-Science Programm sehr wichtig, dass alle Teilnehmer aktiv miteinander und vor allem mit der internationalen Grid-Community kommunizieren und zusammenarbeiten. Aus diesem Grund wurde ein „GridNet“ Netzwerkprojekt eingerichtet, welches ein substantielles Reisebudget besitzt, um sicherzustellen dass Experten des UK e-Science auf allen wichtigen Konferenzen zum Thema Grid vertreten sind.

Weiterhin wird versucht, bedeutungsvolle Verbindungen zu internationalen Grid-Projekten (wie zum Beispiel dem EU DataGrid[17], oder den US iVDGL[18] Projekten) zu schaffen.

2.4.6 Unterstützung der UK e-Science Netzwerkanforderungen

Das UK e-Science stützt sich auf das Universitätsnetzwerk SuperJanet4 um die erforderliche Bandbreite für seine Projekte zu erhalten. Seit Mitte 2002 wurde dieses auf einen 20 Gbps Backbone erweitert:

[13] http://www.aktors.org/
[14] http://www.dirc.org.uk/
[15] http://www.mrc.ac.uk/
[16] http://www-ipg.umds.ac.uk/mias/main.htm
[17] http://eu-datagrid.web.cern.ch
[18] http://www.ivdgl.org/

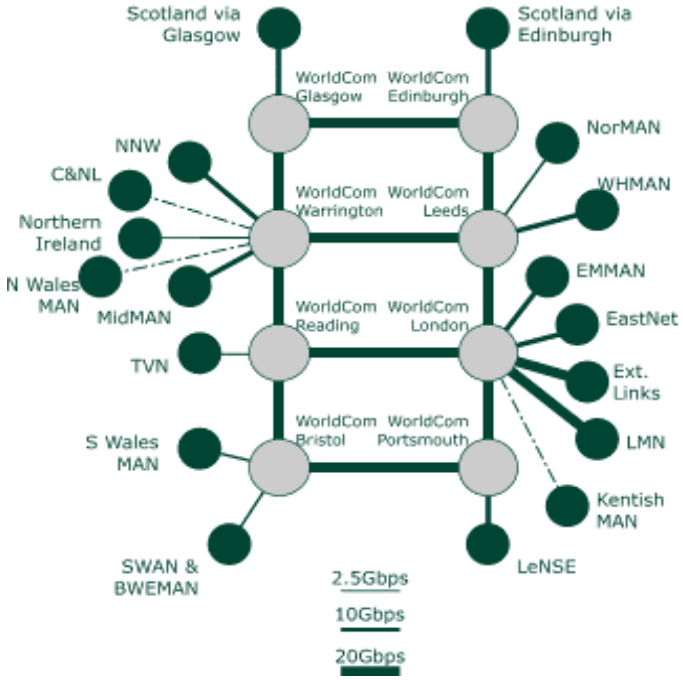


Abbildung 2: Backbone des e-Science Netzwerkes

Damit auch wirklich alle Anforderungen der einzelnen Projekte konzentriert verarbeitet werden können, wurde ein Grid Network Team (GNT) geschafften, dessen kurzfristiges Ziel die Identifizierung von Flaschenhälsen und deren Beseitigung ist, und dessen langfristige Aufmerksamkeit Quality of Service Merkmalen gilt.

3 Biowissenschaftliche Anwendungen

3.1 BIRN

Biomedical Informatics Research Network (BIRN)[19] ist eine Initiative des National Center for Research Resources (NCRR), die sich zum Ziel gesetzt hat, eine Testumgebung für biomedizinische Wissenschaftler zu schaffen. Dadurch wird den Wissenschaftlern der Zugriff auf Daten ermöglicht, die über viele Institute verteilt sind. Die Testumgebung beinhaltet sowohl die Hardware als auch die Entwicklung der Software für Datenbanken und Rechenzentren und natürlich auch die üblichen Problemstellungen, wie Benutzerauthentifizierung, Datenintegrität und Sicherheit.

Der Schwerpunkt der Testumgebung liegt dabei in der Forschung, insbesondere dem Neuroimaging. Dabei wird versucht, das bereits vorhandene Wissen der erfahrenen Community im Bereich der Informationstechnologie, zu nutzen. Ein großer Vorteil der Testumgebung soll die schnelle Installationsmöglichkeit in anderen Entwicklungszentren sein, die nicht unbedingt mit Neuroimaging arbeiten, d.h. aber auch, dass die Soft-/Hardware neben der Skalierbarkeit wieder verwendbar und erweiterbar sein muss.

BIRN setzt stark auf den Ressourcen des “Next Generation Internet“ der National Science Foundation[20] auf. Das erste Ziel war zusammen mit den General Clinical Research Centers (GCRCs) und den Biomedical Technology Research Resources (P41s) die Installation der notwendigen Infrastruktur für die Neuroimaging Projekte.

Die Projektziele lassen sich grob in 5 Bereiche einteilen:

- Der Aufbau eines stabilen High Performance Netzwerkes zusammen mit P41 und GCRC bei Verwendung der Internet2 -Technik
- Der Aufbau einer verteilten und verbundenen Datensammlung für Projekte in den Testumgebungen
- Verteilte heterogene gridbasierte Rechen-Ressourcen für projektspezifische Datensicherung und -analyse
- Die Möglichkeit einer Datensuche in den verschiedenen Datenspeichern bzw. Datenbanken für Nervenzellenaufnahmen
- Entwicklung einer stabilen Soft- und Hardwareinfrastruktur, die es den Centern erlaubt, ihre Arbeit so zu koordinieren, dass an größeren Forschungsprojekten gearbeitet werden kann, als es in einer einzelnen Forschungseinrichtung möglich wäre

[19] http://www.nbirn.net/
[20] http://www.nsf.gov/

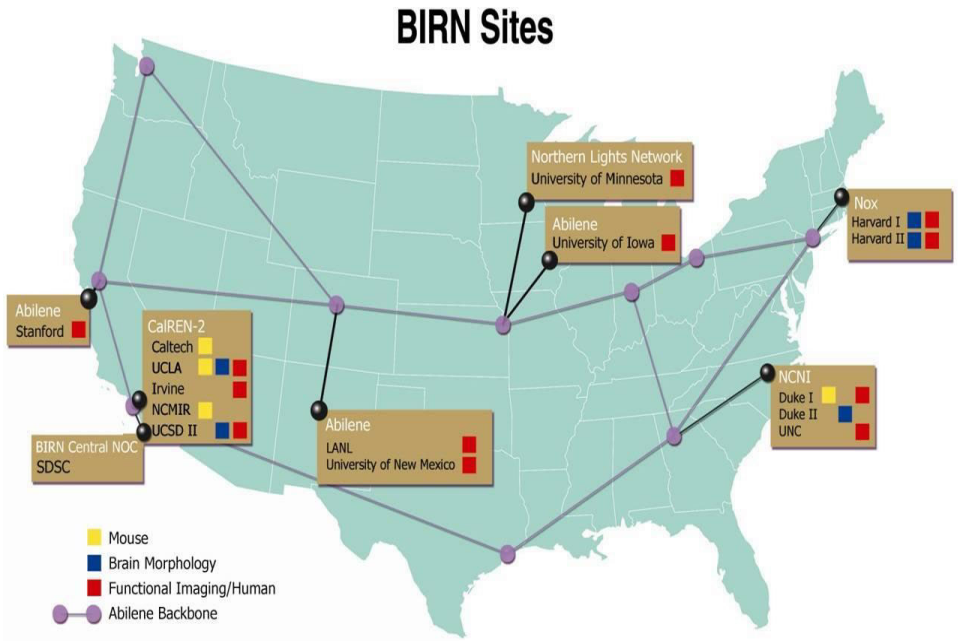


Abbildung 3: Das Abilene Hochgeschwindigkeitsnetzwerk des Internet2 für die Datenübertragung zwischen den BIRN Projektcentern

Um eine einheitliche Experimentenstruktur aufzubauen, werden bestimmte Anforderungen an das System gestellt:

- Leichte Einsatzmöglichkeit
- Schnelle Anpassungen an Soft- und Hardwarekonfigurationen
- Instrumente für die Netzwerkspeicherarchitektur
- System muss mit der Anzahl der Center skalieren
- Softwarekonsistenz muss in der gesamten Organisationsstruktur erhalten bleiben

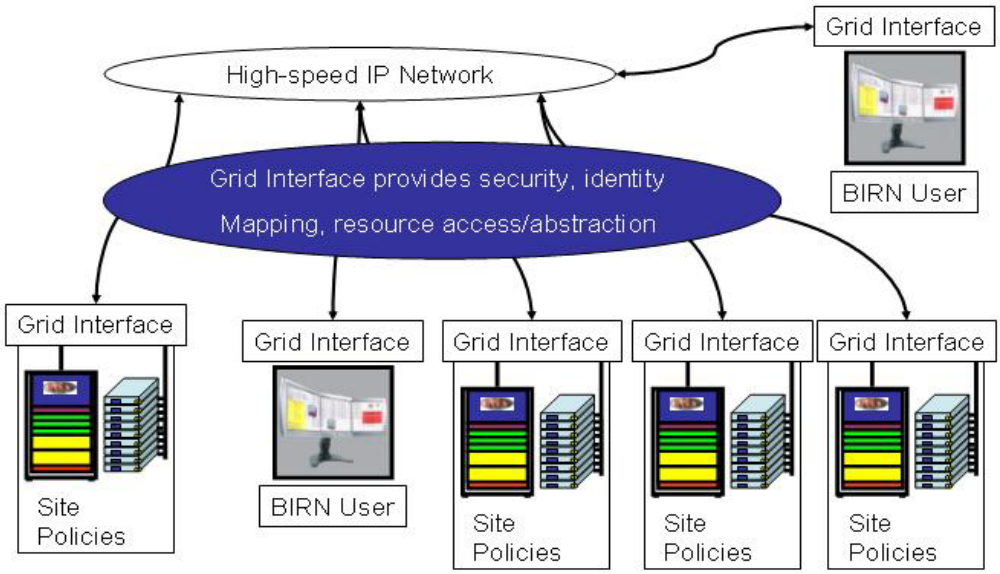


Abbildung 4: Replikation und Symmetrie für die Skalierbarkeit

Daher werden an die Infrastruktur harte Anforderungen gestellt. Die BIRN Wissenschaftler wollen Daten, Datenbanken und Ressourcen teilen, daher sind einige Dinge zu beachten:

- Die Ressourcen sind geographisch verteilt
- Der minimal notwendige Datendurchsatz ist sehr hoch
- Verschlüsselung und Sicherheit sind essentiell
- Die Infrastruktur sollte ihre Komplexität, wenn immer möglich, vor dem Anwender verbergen

Somit liegen die Schwerpunkte der Arbeit in den Bereichen der Infrastruktur und Interoperabilität, wo die Datenmenge heute schon im Terabyte Bereich liegt (bei 10 Centern), dem hohen Datendurchsatz sowie der Spezifizierung und Einsatz von Soft- und Hardware.

Jedes BIRN-Center ist mit der gleichen Hardware ausgestattet und benötigt als Minimum folgende Komponenten:

- Gigabit/10/100 Network Switch – Cisco4006
- Network Statistics System
- Gigabit Ethernet Network Probe
- Network Attached Storage – Gigabit Ethernet
  - 1.0 bis 4.0 TB
- Grid POP (Compaq/HP DL380G2)
  - SRB, Globus
  - Dual Processor Linux mit 1 GB Speicher
- DL 380G2 General purpose
  - Spezialisierte Verschlüsselung
- UPS für Rack

Mögliche Einsatzgebiete sind zum Beispiel die Untersuchungen von Veränderungen sowohl des menschlichen Gehirns als auch des Gehirns von Mäusen.

3.2 MCell

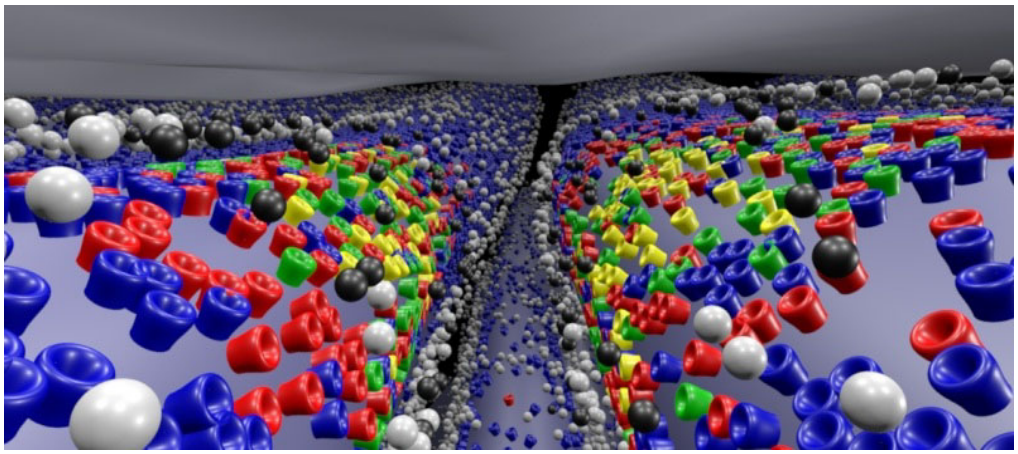


Abbildung 5: MCell Miniatur-Simulation einer Abzweigung einer neuromuskulären Synapse

MCell[21] ist ein gemeinsames Projekt des Terry Sejnowski Labors des Salk-Institutes und des Miriam Salpeter Labors der Cornell Universität. MCell ist „A General Monte Carlo Simulator of Cellular Microphysiology", der mit Hilfe der Monte Carlo Diffusion und von chemischen Reaktionsalgorithmen in Berechnungen in 3D durchführt, um die komplexen biochemischen Interaktionen der Moleküle innerhalb und außerhalb von Zellen zu simulieren. Die Mikrophysiologie ist dabei die Studie der physiologischen Erscheinungen in der mikroskopischen Ebene lebender Zellen. Wie in Abbildung 6 zu sehen ist, läuft die MCell Simulation auf der biologischen Skala über der „molecular dynamic" Ebene und unterhalb der „ganzen" Zellen Ebene ab.

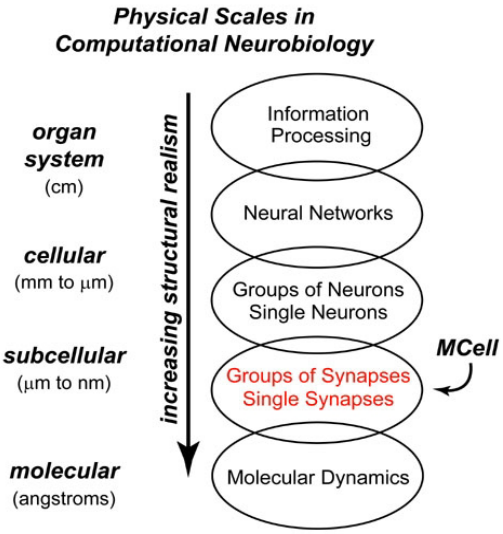


Abbildung 6: Physikalische Ebene der MCell Simulation

Typsicherweise treten bei MCell Simulationen Ereignisse wie die Freisetzung von Ligandmolekülen (im Falle der Mikrophysiologie sind Liganden Neurotransmittermoleküle) in einer Lösung von einer Struktur (z.B. Bläschen), einer de novo Erstellung oder Zerstörung von Ligandmolekülen (z.B. Synthese, Hydrolyse oder Redoxreaktionen), Ligandendiffusion innerhalb einer beliebig ausgewählten Fläche (z.B. pre- und postsynaptische Membrane) sowie chemische Reaktionen zwischen Ligand- und Effektormolekülen (z.B. Rezeptoren oder Enzyme) auf.

Für die Anwendungsbereiche können 3D Bilder erzeugt werden, wobei in diesen polygonale Oberflächen benutzt werden, um die Zellmembrane darzustellen und die in manchen Flächen enthalten Effektoren. Darin wird die Trajektorie der Liganden simuliert, wobei der Interaktion zwischen den

[21] http://www.mcell.cnl.salk.edu

Liganden und den Effektoren besonderes Interesse gilt. Wenn nämlich Liganden und Effektoren aufeinander einwirken, besteht die Möglichkeit, dass sich ein Ligand an einen Effektor bindet und so Informationen aus der Zelle heraus überträgt. Dieser Wechselwirkung gilt das Hauptinteresse der MCell Benutzer.

Zurzeit konzentriert sich MCell vor allem auf die biologische Signaltransduktion, nämlich die Mikrophysiologie der synaptischen Transmission. Es werden aber auch andere Bereiche wie die statische Chemie, die Diffusionstheorie, die Single-Channel Simulationen, Datenanalyse, Geräuschanalyse sowie Markov Prozesse betrachtet. MCell-Simulationen erlauben den einzigartigen Einblick in das Verhalten und die Schwankungen von echten Systemen bei einer endlichen Interaktion der Moleküle in einer begrenzten komplexen Umgebung.

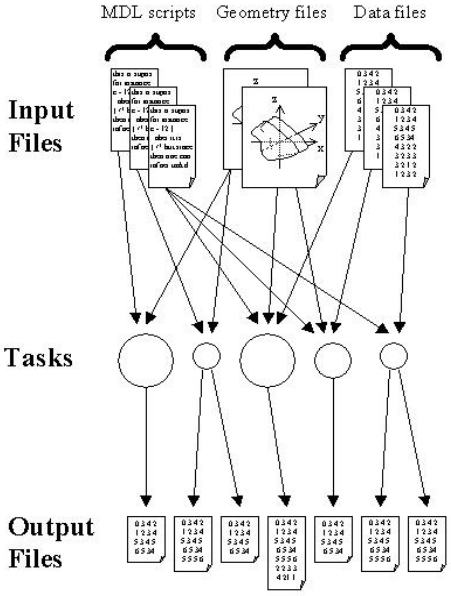


Abbildung 7: Ablauf der „Parameter Sweep Application“

Die Entwicklung von MCell Anwendungen hat zwei substantielle Bereiche:

Der erste, ist die Unterstützung eines Modells für die bedeutende Gruppe der biowissenschaftlichen Anwendungen, für die neue disziplinarische Ergebnisse für noch größere Probleme erreicht werden können. Die zweite, ist die Unterstützung MCells für die Gruppe der „Parameter Sweep“ Anwendungen, eine sehr zukunftsträchtige Anwendung des Grids.

In der aktuellen MCell Implementation sind die Liganden, Effektoren, Reaktionsmechanismen, 3D-Flächen und andere für die Simulation notwendige Komponenten mittels einer einfachen Programmiersprache bzw. der Model Description Language (MDL), einer direkt für die Biologen entwickelten Sprache, spezifiziert. Bei der Ausführung von MCell werden die verschiedenen MDL „Input Files“ geparst und daraus Simulationsobjekte erstellt, die Länge der Simulation hängt dann entsprechend von der gewählten Iterationsdauer ab, wobei jede Iteration einem Monte Carlo Schritt entspricht.

Um die Simulation realistisch zu machen, werden bei der Monte Carlo Funktion Zufallszahlen benutzt.

Bei jeder einzelnen Ausführung hängt die Diffusion der Ligandenmoleküle von der Beweglichkeit der Moleküle und dem 3D Konzentrationsgradienten ab. Der herkömmliche Weg zur Berechnung ist die Diskretisierung des Raums und Nutzung eines eindimensionalen Gleichungssystems (PDE).

Aufgrund der Implementationsart der Monte Carlo Simulation in MCell ist es nötig, dass verschiedene Instanzen derselben Berechnung mit verschiedenen Zufallszahlen durchgeführt werden. Wenn genügend Ergebnisse vorhanden sind, wird aus der Berechnung des Durchschnitts aller Ergebnisse dieses als

repräsentatives Ergebnis der Simulation gewertet. Die Software wurde so entwickelt, dass sie portabel ist und somit die verschiedenen Instanzen auf ganz verschiedenen Rechnern durchgeführt werden können. Da die Simulationsinstanzen alle unabhängig voneinander sind, können alle Berechnungen parallel ausgeführt werden.

Jede Instanz der Simulation (bzw. Aufgabenstellung) führt die Monte Carlo Simulation für eine bestimmte vordefinierte Anzahl aus. Die Eingabedaten bestehen einerseits aus vom Benutzer geschriebenen MDL Skripten und Dateien, die die Elemente der Simulation beschreiben (z.B. räumliche Geometriebeschreibungdateien, Dateien mit den Stellen der Effektoren, die Verteilung der Chemikalien). Zurzeit benötigt ein typischer Durchlauf 5MB Eingabedaten pro Aufgabenstellung für insgesamt 1000 Aufgabenstellungen. Jede Aufgabenstellung produziert Ausgabedateien, die für 2D ungefähr 1MB bzw. in 3D ungefähr 10MB Speicher benötigen. Pro Aufgabenstellung sind ca. 100MB RAM notwendig und benötigen in den heutzutage schnellsten Rechnern ca. 10 Minuten Rechenzeit. Die Ausgabedateien werden vom Benutzer gesammelt und können anschließend auf verschiedenen Wegen nachbearbeitet werden je nach Anwendungszweck (z.B. Datenuntersuchungen und -analysen, Visualisierungen, Animationen usw.).

Wenn jeder einzelne Benutzer jeden Job per Hand verteilen und auswerten muss, wird dies sehr aufwendig und ineffizient, besonders da man daran interessiert ist hunderte von Jobs parallel zu berechnen. Dafür sind auch mehr Rechner nötig als einzelne MPPs, die den Instituten zur Verfügung stehen. Daher ist eine der Herausforderungen, die Berechnungen auf mehrere Maschinen zu verteilen, wobei diese nicht Teil desselben Netzwerkdateisystems sind.

Es ist daher nicht nur aufgrund der Aufgabenanzahl, sondern auch anhand des Rechenaufwands und der Datenanforderungen für jede Aufgabe absehbar, dass bald Simulationen laufen, die mehrere Tage Rechenzeit benötigen. So sind zum Beispiel Aufgabenstellungen für 100000 Tasks mit 50 MB Eingabedaten in Planung, die mehr RAM und Rechenzeit benötigen, als einem Durchschnittsbenutzer zur Verfügung stehen. Solche Berechnungen würden die Modellierung der Zellen auf molekularer Ebene ermöglichen, was immer noch eine Herausforderung für die Forscher darstellt. Zurzeit ist es für die Entwickler solcher Berechnungen schwer, ausreichende und häufige Rechenzeit auf den MPPs zugewiesen zu bekommen. Daher ist diese Anwendung durch die Struktur des Codes und den Hardwareanforderungen ideal für Grids geeignet.

Zurzeit wird ein Prototyp entwickelt, der MCell als eine “parameter sweep” Anwendung strukturiert, d.h. es können viele Instanzen der Anwendung mit verschiedenen Parametern ausgeführt werden und anschließend auch wieder eingesammelt werden sowie eventuell weiterverarbeitet. Solche Berechnungen sind also ideal für das Grid, wo jede Instanz unabhängig von den anderen ist und so auf einem beliebigen Rechner ausgeführt werden kann. Allerdings vergrößert sich durch die gemeinsame Nutzung die Komplexität der Entwicklung der performanzorientierten Implementierung. In der Abbildung 7 werden die Speicherabhängigkeiten aufgezeigt. Die Implementation benötigt außerdem Mechanismen zur Ressourcenverwaltung, Fehlererkennung, Fehlerkorrektur und zum Scheduling, welche für solche Aufgaben zwingend notwendig werden. Ebenso

muss eine angemessene Softwareinfrastruktur, die den MCell Benutzern die vernetzten Rechner und Speicher zugänglich macht, geschaffen werden.

3.3 myGrid

myGrid[22] ist ein vom Engineering and Physical Sciences Research Council (EPSRC)[23] gegründetes Projekt als Teil des UK e-Science Programms (siehe Seite 4). Beteiligt daran sind die Universitäten von Manchester, Newcastle, Nottingham, Sheffield und Southampton. Weiterhin sind zusätzlich IT Innovations (Southampton) und das European Bioinformatics Institute (EBI)[24] (Hinxton, Cambridge), sowie mehrere kommerzielle Partner darin involviert.

Das Ziel ist die Entwicklung der notwendigen Middlewareinfrastruktur. Diese agiert oberhalb der existierenden Web Services und Infrastruktur um den Wissenschaftlern Zugang zu den verteilten Ressourcen zu gewähren. Der erste Anwendungsbereich ist die Bioinformatik. So soll myGrid als erstes den „e-Biologist's workbench" freigeben. Generell soll es aber den verschiedensten Wissenschaften den Zugang zum Grid ermöglichen.

myGrid wird den Wissenschaftlern eine Vielzahl an zusätzlichen Diensten bereitstellen. Der Inhalt dieser Dienste ist die Abdeckung der grundsätzlichen Aufgabenbereiche, wie die Reproduzierbarkeit von Analysen, Veröffentlichung von Ergebnissen und der automatischen Meldung bei Änderungen von Daten. So stellt myGrid quasi ein virtuelles Laborbuch dar, welches die eigenen Experimente verfolgt und diese in gewissem Maße sichert, so dass diese verteilt und anderswo ausgeführt werden können. Es ermöglicht die gleichen Standards in Bezug auf die Aufnahmeprotokolle und Ergebnisse, als würde man ein typisches Experiment im eigenen Labor durchführen.

Die folgenden Komponenten stellen die spezifischen Ziele des myGrid Projektes dar:

- Standort und Zugriff auf Ressourcen wie Datenbanken und Analysewerkzeuge
- Zusammengesetzte Arbeitsabläufe, virtuelle Experimente
- Personalisierung
    - Persönliches Archiv
    - Virtuelles Laborbuch
    - Konfiguration
    - Mitbenutzung
    - Notizen
- Nachrichtendienste
    - Eventhandler
- Benutzerschnittstellen
- Verzeichnis der Servicedienste
- Entdeckung von Diensten
    - Ontologie
    - Metadaten

[22] http://www.mygrid.info/
[23] http://www.epsrc.ac.uk/
[24] http://www.ebi.ac.uk/

- Sicherheit
- Herkunftsmanagement
  - Aufzeichnung der Prozesse
  - Attributierung
  - Aktualisierung der Dienste

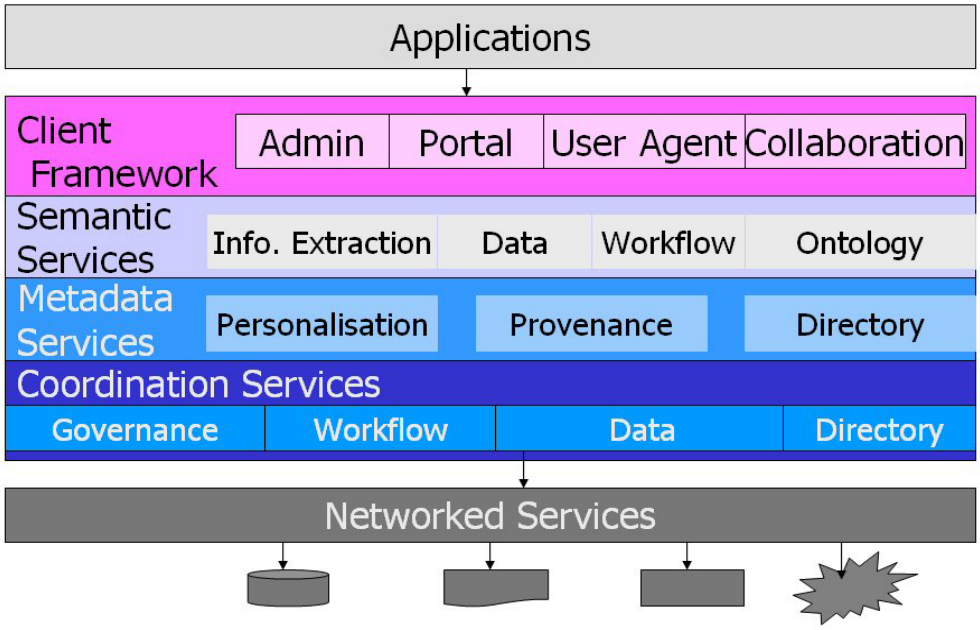


Abbildung 8: myGrid Stack

myGrid unterstützt eine allgemeine Ausführungsumgebung, die die Ausführung von parallelen „bag-of-task“ Anwendungen ermöglicht. Diese Anwendungen können auf allen Rechnern ausgeführt werden, auf die der Benutzer Zugriff hat und werden auch „Benutzergrid“ genannt. Sie bestehen aus unabhängigen Aufgaben, die in beliebiger Reihenfolge, aber nicht simultan ausgeführt werden können. Da solche Grids sehr groß sein können, ist die Rechenzeit potentiell unbegrenzt für die „bag-of-task“ Anwendungen.

Das Erstellen einer myGrid Anwendung soll sehr einfach vonstatten gehen. Der Benutzer beschreibt mit sequentiellen Anweisungen die unabhängigen Aufgabenbereiche und spezifiziert für jeden die Ein- und Ausgabedateien. Um die Aspekte des Grid Computing, wie spezielle Infrastrukturen oder Scheduling, muss sich der Benutzer nicht kümmern. Durch ein paar Abstraktionen wird es dem Benutzer ermöglicht, das Grid auf bequeme Art und Weise zur Fertigung zu nutzen.

Auf den einzelnen Maschinen des Benutzergrids muss keine zusätzliche Software installiert werden, alle Komponenten, die zusätzlich zur Ausführung benötigt werden, werden automatisch installiert. Der Benutzer muss lediglich das Open Grid unterstützen, was bedeutet, dass er zu den beteiligten Maschinen Daten transferieren sowie einen Remotezugriff herstellen können muss.

4 Ingenieursbasierte Anwendungen

4.1 NASA IPG

NASA's Information Power Grid (IPG)[25] ist ein Hochleistungsrechen- und Datengrid, das geographisch verteilte Computer, Datenbanken und Instrumente verbindet.

Die NASA hat in der Entwicklung der Grids und den Zugriffsmöglichkeiten zwischen Grids Grundsatzarbeit geleistet. Am Anfang wollten Organisationen Rechenzeit, Daten und Werkzeuge teilen, auf der Basis „Wir geben dir Zugriff auf unsere Computer und Instrumente, wenn du uns Zugriff auf deine gewährst“. Die dazu notwendige Middleware wurde eingerichtet, verbessert und erweitert. Werte wie Sicherheit und Kompatibilität kamen hinzu, so dass heute der Zugriff auf Ressourcen größtenteils auf Zertifikaten basiert. Dadurch ergaben sich enorme Vorteile wie mehr Leistung, Ressourcen und Flexibilität. Heute sind Aufgaben auf vielen Rechnern auf einmal ausführbar, wobei man sich die passenden aussuchen kann, so, dass berechenbare Aufgaben immer größer werden.

Gegründet vom Computing, Information, and Communications Technology (CICT)[26] Programm des NASA Ames Research Center[27] ist das IPG ein gemeinsames Werk von:

- NASA Ames Research Center (ARC)
- NASA Glenn Research Center (GRC)[28]
- NASA Langley Research Center (LaRC)[29]
- Partnerships for Advanced Computational Infrastructure (PACI)[30]
    - NSF PACI Programm am National Center for Supercomputing Applications (NCSA)[31]
    - NSF PACI Programm im San Diego Supercomputing Center (SDSC)[32]
- Argonne National Laboratory (ANL)[33]
- Information Sciences Institute (USC/ISI)[34]

[25] http://www.ipg.nasa.gov/
[26] http://www.cict.nasa.gov/
[27] http://www.arc.nasa.gov/
[28] http://www.grc.nasa.gov/
[29] http://www.larc.nasa.gov/
[30] http://www.paci.org/
[31] http://www.ncsa.edu/About/Alliance/
[32] http://www.npaci.edu/
[33] http://www.anl.gov/
[34] http://www.isi.edu/

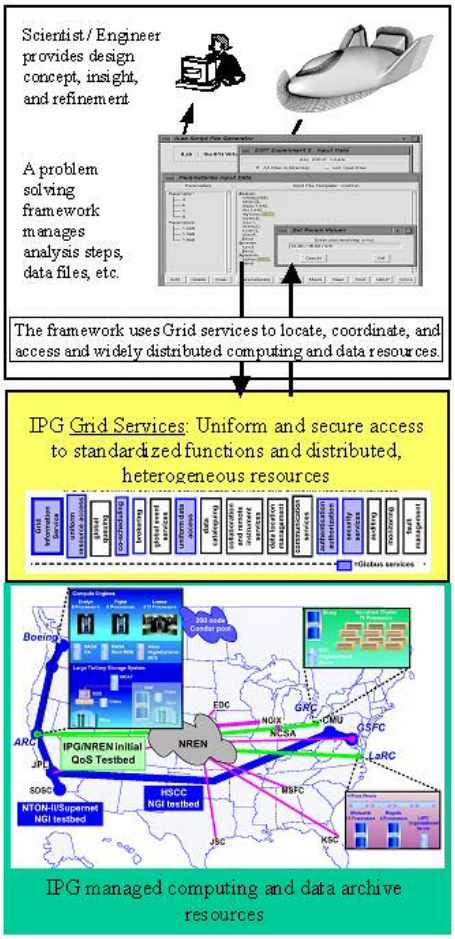


Abbildung 9: IPG Konzept, einheitlicher Zugriff auf verteilte heterogene Ressourcen

Das Ziel des IPG ist die Art des Rechnens in den wissenschaftlichen und technischen Aktivitäten bei der NASA zu revolutionieren. Dieses Ergebnis wird durch fundamentale Veränderungen und Verbesserungen beim Zugriff auf Rechensysteme, Datenarchive, Instrumente und Werkzeuge erreicht. Der Fortschritt besteht in der Art der Nutzung, da Dienste in die Arbeitsumgebungen der Benutzer eingebunden werden, so dass diese einen einheitlichen und leistungsfähigen Zugang zu den Ressourcen haben, egal wo diese liegen. Dies gilt für transiente Ressourcen wie Computersysteme und Datencaches, die z.B. für Simulationen oder Datenanalysen für ein einzelnes Experiment benötigt werden, genauso, wie für persistente Ressourcen wie Datenbanken, Archive und Instrumente.

Eines der gesetzten Ziele im Rahmen des Projektes ist die Fähigkeit der Dienstimplementierungen, die für die Problemlösung benötigten Ressourcen lokalisieren und schedulen zu können. So unterstützen diese zum Beispiel auf Anfrage das Zusammenfügen von Anwendungen, sowie das Sammeln und Verarbeiten von Daten von Onlineinstrumenten in Echtzeit. Dadurch ist das Steuern von Experimenten ebenso möglich, wie das Vornehmen von Änderungen innerhalb eines Experimentes in Abhängigkeit der aktuellen Ergebnisse. Weiterhin kann man verschiedene Experimentumgebungen zusammenschalten.

Es müssen also Dienste in der Middleware geschaffen werden, welche die organisatorisch und geographisch verteilten Entwicklungsumgebungen unterstützen. Die Basisdienste werden neben der Lokalisierung und dem Scheduling der Ressourcen auch einen einheitlichen und sicheren Zugang zu heterogenen Ressourcen sowie einen globalen Namens- und Abgrenzungsmechanismus zur Unterstützung von Entwicklung und Management der virtuellen Organisationen bzw. Gemeinschaften bieten. Weiterhin werden Monitoring, Auditing und Fault Recovery unterstützt.

An der Spitze der Basisdienste werden generische und spezifische Workflowmanagementsysteme laufen, die die benutzerdefinierten Protokolle für die verschiedenen Szenarien wie zum Beispiel für Simulationen, Datenanalysen, globale Datenkatalogisierung und Replikationssysteme managen. Diese Dienste sollen unmittelbar von den Wissenschaftlern und Ingenieuren verwendet werden.

Letztendlich werden alle Dienste über das Netz bzw. die Desktopschnittstelle (das „problem solving framework“ in Abbildung 9) zugänglich gemacht, um eine

einsatzfähige Umgebung zu schaffen, in denen die Protokolle gestaltet, kontrolliert, modifiziert, mit anderen Aspekten des Frameworks integriert und auch mit anderen Teilnehmern sicher geteilt werden können.

Die Fortschritte, die dabei sowohl von der IPG als auch der Grid Community erreicht werden, werden durch das Global Grid Forum[35] koordiniert. IPG spielt durch seine interne R&D Arbeit und praktische Einsätze eine Hauptrolle und arbeitet auch mit verschiedenen Universitäten und dem NSF Supercomputers Center (NASA-NSF MOU) zusammen. Zurzeit hat das IPG eine große Prototypentwicklungsumgebung, die auch das ARC, GRC, LaRC und bald JPL mit einbezieht.

Abbildung 10 zeigt eine Anwendungsmöglichkeit für die NASA. So können zum Beispiel einzelne Aspekte eines Flugzeuges, wie die Flügel, Triebwerke und das Fahrwerk einzeln modelliert und untersucht werden. Jeder Teil kann von Ingenieuren, die auf der ganzen Welt verteilt sein können, bearbeitet werden und dann mit Hilfe des Grids in die Gesamtstruktur eingebunden werden, woraus sich der Vorteil der parallelen Arbeitsmöglichkeit vieler Ingenieure ergibt.

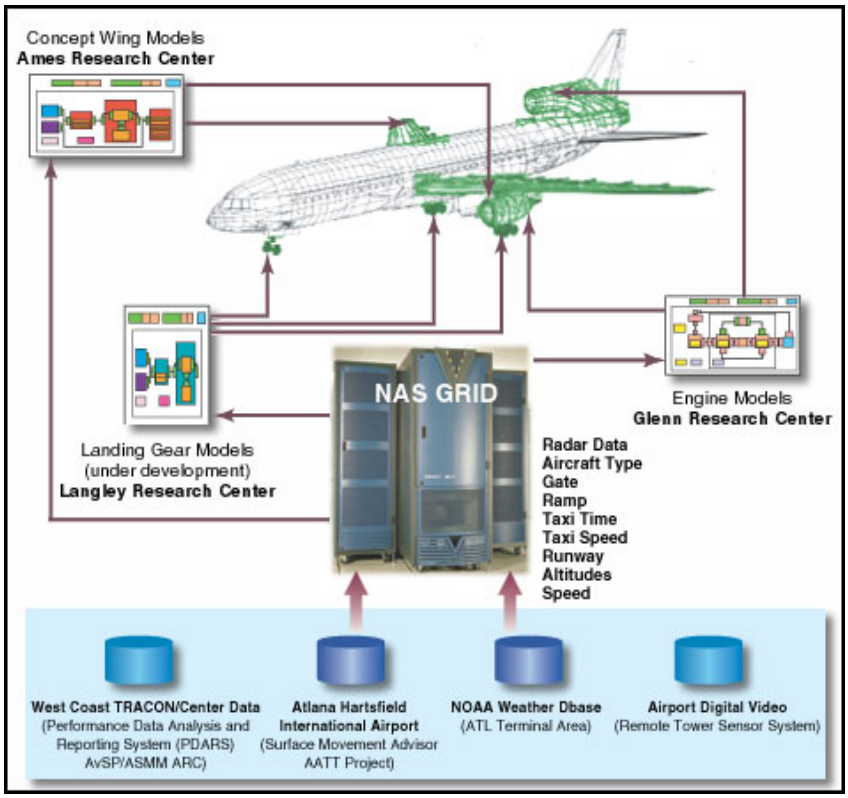


Abbildung 10: NAS Grid

Die Ziele im Jahr 2000 waren der Aufbau einer prototypischen Grid Umgebung, die eine heterogene, verteilte Rechen-, Daten- und Werkzeugumgebung bereitstellt, die den einheitlichen Zugriff auf Ressourcen unterstützt. Ein besonderes Ziel war die Umsetzung eines baseline Betriebssystems, das aus folgendem besteht:

[35] http://www.gridforum.org/

- Rechenressourcen: 600 CPU Knoten in einem halben Duzend SGI Origin 2000, mehrere Workstationcluster in Ames, Glen und Langley, sowie Pläne für die Zusammenarbeit mit Goddard und JPL und 270 Workstations in einem Condor Pool
- Kommunikation: Ein Highspeed WAN Testnetz für die teilnehmenden Center
- Speicherressourcen: 30-100 TB zur Archivierung der Daten in einem einheitlichen und sicherem, für alle zugänglichem, System
- Software: Globus[36], unterstützt gemeinsame Dienste sowie die Programmierung und Ausführung von Programmen, MPI, CORBA, einem Jobmanager und dem Condor Managementsystem
- Personaleinsatz: Stabile und arbeitsunterstützende Umgebung
- Anwendungen: Mehrere Benchmarks laufen im IPG (Parameter Studien, Simulationen, CFD)
- Gridübergreifende Anwendungen (z.B. zwischen dem IPG und dem NCSA Grid)

Diese Umgebung ist in Abbildung 9 zu sehen, welche dann auch benutzt wird, um zu zeigen, dass das Ziel erreicht wurde.

4.2 GEODISE

Intelligente Suchwerkzeuge werden bald einen wichtigen Bestandteil bei der Entwicklung im Ingenieurswesen bilden. Diese Werkzeuge helfen dem Benutzer beim Herstellungs-, Ausführungs- und Nachbearbeitungsprozess sowie bei der Suche von Entwürfen und Optimierungsmöglichkeiten. Dieser Ansatz benötigt allerdings im großem Umfang verteilte Simulationen, die mit verbunden Werkzeugen arbeiten, die die Entwürfe in Abhängigkeit der daran angeschlossen Datenbanken beschreiben und gestalten. Die Werkzeuge sind oftmals an physikalisch weit entfernten Orten aufgebaut und unter Kontrolle verschiedenster Verantwortlichkeitsbereiche. Während die Bewertung eines einzelnen Entwurfes Daten im GB-Bereich erfordert, wäre eine Verbesserung des Entwurfsprozesses wesentlich effektiver, wenn dieser die verteilten Daten, die im TB Bereich liegen, nutzen könnte.

Das Ziel von Grid Enabled Optimisation and Design Search for Engineering (GEODISE)[37] ist die Bereitstellung eines auf Gridtechnologie basierenden nahtlosen Zugangs zu intelligenten Wissensspeichern, aktuellsten Optimierungs- und Suchwerkzeugen und verteilten Rechen- und Datenressourcen zu schaffen.

Die Hauptziele sind dabei:

- Bereitstellung eines Demonstrators für die Werkzeugoptimierung für die Berechnung von Strömungsdynamiken (CFD)
- Nutzung der Gridtechnologie zur flexiblen Kopplung von Rechen- und Datenressourcen
- Die Verbesserung des Entwicklungsprozesses durch die Aneignung von Wissensmanagement und Ontologiemethoden

[36] http://www.globus.org/
[37] http://www.geodise.org/

- Die Adoption von industriellen Projektmanagementmethodiken für laufende Projekte

Geodise beabsichtigt die Bereitstellung eines Grid-basierten generellen Integrationsframeworks für rechen- und datenintensive Optimierungsaufgaben bei Aufrechterhaltung der individuellen Aufgabenbereiche.

Um flexible und auch kostengünstige Hochleistungsberechnungen durchführen zu können, wird das Projekt Grid- und Clustertechnologie (Globus, Condor, OGSA) nutzen. Geodise wird mit Hilfe einer Matlabumgebung und einem Portal, das den Fernzugriff auf das CFD und die Optimierungswerkzeuge ermöglicht, umgesetzt.

Folgende Schlüsseltechnologien sollen dabei eingesetzt werden:

- Open Grid Services Architecture und Web Services
- Entwurfsoptimierungstechniken
- Berechnung von Strömungsdynamiken (CFD)
- Steuerung von Arbeitsabläufen und deren Ausführung
- Bildung und Unterstützung von Ontologiediensten
- Wissensmanagement
- Integrierte Rechen- und Datendienste
- Matlab Werkzeug

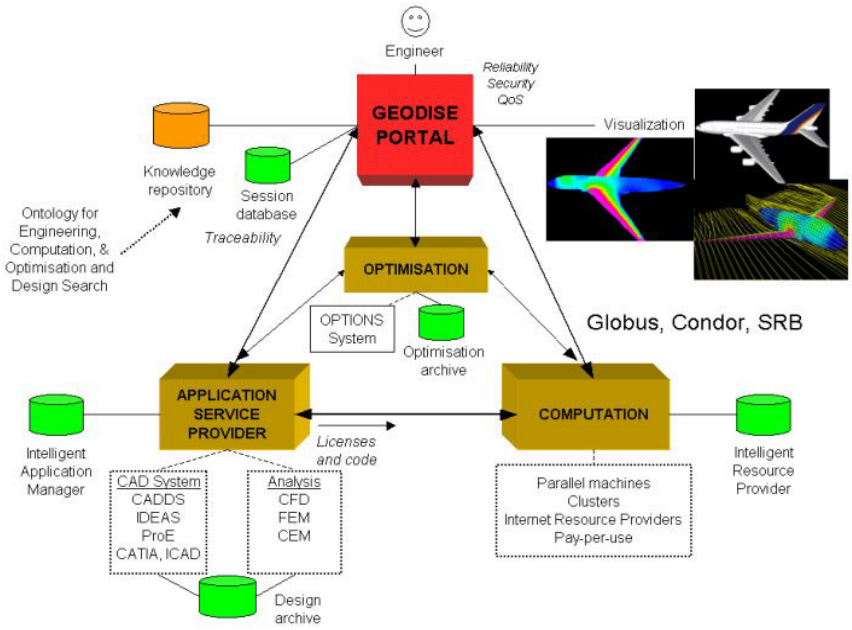


Abbildung 11: GEODISE Ingenieursplanung

Der erste industrielle Einsatz des Projektes wird die Berechnung von Strömungsdynamiken sein. Das große Ziel ist es nun, das CFD und das Entwurfswissen, welches bis vor kurzem nur sehr wenigen Spezialisten zur Verfügung stand, einer größeren Gruppe von weniger erfahrenen Entwicklern in einer Umgebung zur Problemlösung (PSE) verfügbar zu machen. Ausgehend von einem Startentwurf mittels eines konventionellen CAD-Programms soll das Geodise PSE dem Entwickler ermöglichen seinen Entwurf zu optimieren. Die CFD Optionen und

Optimierungsmodelle werden dann über das Portal mit Hilfe anderer aus der Datenbank gewählt.

Aber CFD ist nur eine Möglichkeit für den Einsatz solcher Werkzeuge. Die Benutzer des Projektes sehen für Entwürfe in den Bereichen der Architektur für strukturelle oder photonische Geräte großes Potential. Weiterhin entstehen viele Möglichkeiten in der weiteren Optimierung, wenn verschiedene Analysemethoden kombiniert werden.

Bei der Entwicklung von Geodise sind sowohl die Universitäten von Southampton, Oxford and Manchester als auch große Industriepartner beteiligt. Im Gebiet der Luftfahrt sind dies BAE Systems[38], Rolls-Royce[39] und Fluent[40], einer der weltweit führenden Entwickler für CFD Software, sowie auch eine Gruppe unterstützender Firmen wie Intel (Hardware), Microsoft (Software), Compusys[41] (Systemsintegration), Epistemics[42] (Wissensforschung) und Condor[43] (Grid-Middleware).

[38] http://www.baesystems.com/
[39] http://www.rolls-royce.com/
[40] http://www.fluent.com/
[41] http://www.compusys.co.uk/
[42] http://www.epistemics.co.uk/
[43] http://www.cs.wisc.edu/condor/

5 Datenorientierte Grid-Applikationen

5.1 DAME

Das DAME (Distributed Aircraft Maintenance Environment)[44] Projekt hat zwei Ziele: Die Erstellung eines computerbasierten Fehlerdiagnose- und -prognosesystems und die Integration von diesem mit einem „vorhersagbaren Wartungssystem“ im Kontext von Flugzeugmotoren. Dabei bedeutet „vorhersagbare Wartung“, dass es ein genügend großes Zeitintervall zwischen der Beobachtung von unnormalen Verhalten und dem aktuellen Auftreten von Fehlern gibt. In genau diesem Zeitfenster setzt DAME Griddienste ein, um folgendes zu machen:

- Erstellen einer Diagnose (Warum ist das Verhalten so?)
- Entwicklung einer Prognose (Was wird demzufolge passieren?)
- Planung von rehabilitierenden Maßnahmen (Was kann man dagegen machen und wann?)

Eine zentrale Anforderung dabei ist auch die Erstellung eines Proof of Concepts für die industriellen Partner dieses Projekts (Rolls Royce, usw). DAME basiert auf einer „open web services“ Architektur, die auf dem Globus Gridmodell aufsetzt (siehe Abbildung 12), wobei in diesem Zusammenhang gerade geklärt wird, inwieweit dieses Webservice Modell wirklich in dem gegebenen Kontext anwendbar ist.

Zur Verwirklichung dieser Ziele wurden zwei neue Techniken entwickelt:

- QUOTE, welches Techniken, basierend auf neuronalen Netzen, einsetzt, um Anwendungen in Echtzeit zu überwachen, und dieses als Webservice-Anwendung zur Verfügung stellt.
- AURA (Advanced Uncertain Reasoning Architecture for Pattern Matching), wobei es sich um ein System handelt, das speziell für Pattern Matching Aufgaben entwickelt wurde, und im Rahmen von DAME eine Umsetzung für Globus (AURA-G) erhielt, damit auch verteiltes Rechnen über verschiedene Motoren an verschiedenen Stellen ermöglicht wird. Dabei ist AURA OGSA-konform.

[44] http://www.iri.leeds.ac.uk/Projects/IAProjects/karim1.htm, http://www.cs.york.ac.uk/dame/

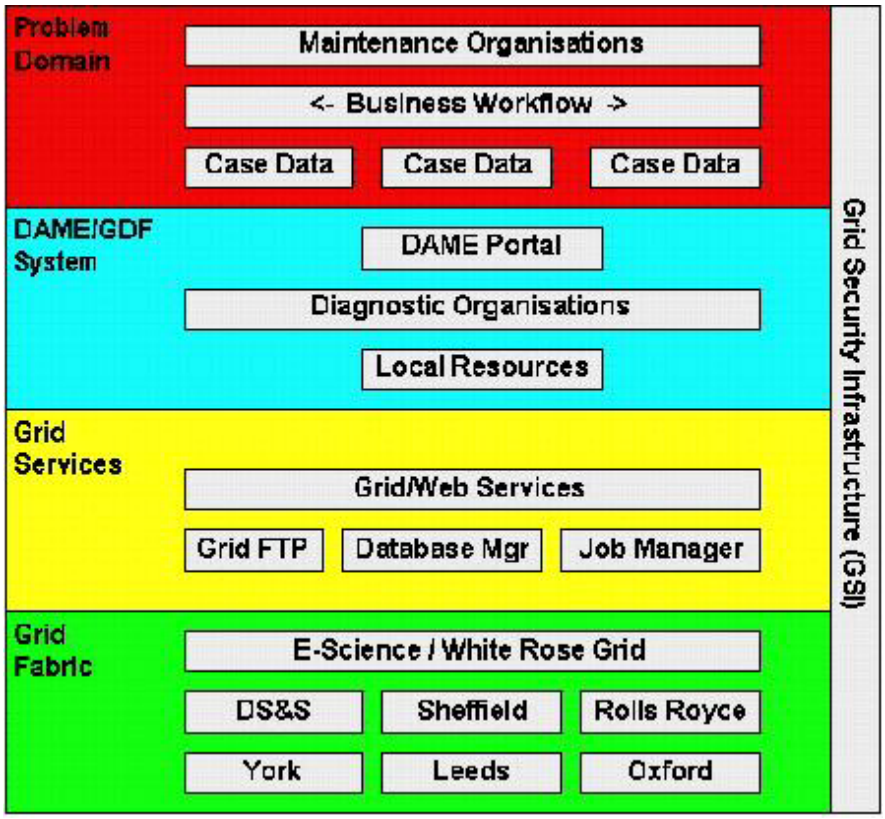


Abbildung 12: DAME Architektur

Der Ablauf ist folgendermaßen:

Die Motordaten werden von QUOTE aufgenommen und gefiltert und dann an eine Diagnosestation weitergeleitet. Weiterhin werden die Daten an ein Data Warehouse gesendet, welches an AURA-G angeschlossen ist. Falls nun ein abnormales Verhalten in der Diagnosestation erkannt wird, werden Anfragen an AURA-G geschickt, um nähere Auskünfte über die Probleme zu erhalten.

Dabei vergleicht AURA-G die vorliegenden Daten und versucht ähnliche Verhaltensmuster in der Vergangenheit wieder zu finden. Aus dieser Analyse werden dann die Diagnosen und Prognosen für das aktuelle Problem erstellt, sowie ein Plan aufgestellt, wie das Problem am besten zu lösen ist.

Dieser Plan wird wieder an die Diagnosestation gesendet, die ihn je nach Fall direkt mit dem betreffenden Flugzeug, bzw. betreffenden Wartungsstationen und Flughäfen koordiniert, um ihn umzusetzen.

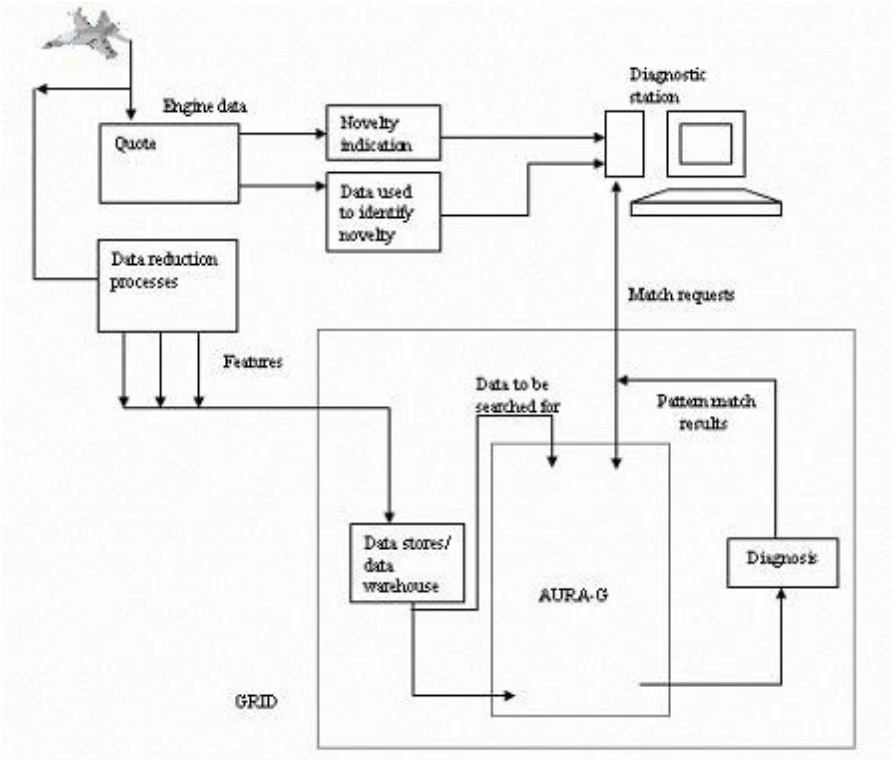


Abbildung 13: DAME AURA-G

Die Anforderungen, die dabei entstehen, sehen folgendermaßen aus:

- Es muss eine sehr große Menge an Daten von entfernten Quellen (Flugzeug, Wartungszentren, Date Warehouses) übertragen werden.
- Es wird ein Zugriff auf hochspezialisierte Daten- und Diagnosesoftware gefordert.
- Es müssen extrem große Datenmengen durchsucht werden.
- Verschiedene Institutionen aus aller Welt müssen reibungslos miteinander ihre komplizierten Interaktionen abhalten können.
- Die erstellten Diagnosen und Prognosen müssen qualifiziert beweisbar sein.
- Und zuletzt dürfen die kritischen Aspekte für eine kommerzielle Verbreitung (Echtzeit, Kosten, Quality of Service,…) nicht außer acht gelassen werden.

Allerdings ist hier, genauso wie an allen anderen Orten dieses Projekts, die Dokumentation sehr spärlich (z.B. findet man keinerlei Angaben darüber, wie viel Daten wirklich übertragen werden müssen) und die Homepage von diesem Projekt wurde zuletzt vor mehr als einem Jahr geupdatet. Daraus lässt sich die Frage ableiten, ob es dieses Projekt wirklich noch gibt, oder ob es nicht inzwischen mehr oder weniger gescheitert ist. Was man jedoch sagen kann, ist, dass AURA-G einen schnellen Zugriff auf große, komplexe Daten liefert und neben dem kompletten Diagnose-Framework von DAME auch durchaus einzeln in anderen Kontexten eingesetzt werden kann.

6 Physikalische Grid-Applikationen

Im Zusammenhang mit der Physik gibt es eine Menge Anwendungsgebiete für Grids. Vor allem die Teilchenphysik ist ein Gebiet, in dem immer schneller immer größere Datenmengen erzeugt und verarbeitet werden müssen. Als Beispiel sei hier das schon erwähnte Large Hadron Collider (LHC)[45] Projekt im CERN Institut genannt, in dem das Higgs-boson erforscht und am Problem der dunklen Materie geforscht wird. Hier arbeiten mehr als 100 Institutionen mit über 1000 Physikern aus Europa, USA und Japan zusammen. Wenn die Experimente 2005 dann beginnen, werden für jedes Experiment mehr als 10GB pro Sekunde an Daten erzeugt, die für weitere Analysen alle ausgewertet und verteilt werden müssen. Um dies zu bewältigen werden Grids und Gridprojekte aus aller Welt zusammenarbeiten, unter anderem das EU DataGrid, UK e-Science GridPP oder aus den USA das GriPhyN[46]. Da die dafür benötigte Infrastruktur sich allerdings noch im Aufbau befindet, wollen wir hier nicht näher darauf eingehen, sondern uns um einen anderen Bereich der physikalischen Anwendungen kümmern, nämlich der Astronomie. Genauer gesagt werden wir das Astrophysical Virtual Observatory der EU (EU AVO)[47], das National Virtual Observatory der USA (NVO)[48] und das bereits erwähnten AstroGrid des UK e-Science (siehe Seite 4) näher beleuchten.

6.1 EU AVO / NVO / UK AstroGrid

Gemeinsames Ziel dieser Projekte ist es, einen einheitlichen Zugang zu einem verbundenen, aber gleichzeitig weit verteilten Speicher von astronomischen Daten, die von Radiowellenaufnahmen bis hin zu Röntgenanalysen reichen, zu schaffen. Bisher war es so, dass astronomische Daten mit vielen verschiedenen Teleskopen in allen denkbaren Wellenlängen aufgenommen wurden und dementsprechend auch vollkommen unabhängig voneinander gespeichert und aufbewahrt wurden. Dies hatte zur Folge, dass es so gut wie unmöglich war, an alle Daten heranzukommen, um zu sehen, inwieweit sich andere Forscher schon mit bestimmten Aspekten der Astronomie beschäftigt haben. Und selbst wenn man auf alles zugreifen könnte, die momentan täglich erzeugte Datenmenge der Teleskope und Observatorien braucht zur Verarbeitung und Vereinheitlichung ohne die Nutzung von Grids mehr als 60 Tage.

Aus diesem Grund sind jetzt mehrere Bestrebungen im Gange. Zum einen sollen die verschiedenen, schon existierenden Grid-Infrastrukturen benutzt werden, um die anfallenden Daten aufzubereiten und zu verteilen. Dafür werden unter anderem das schon erwähnte UK e-Science und das TeraGrid (>13 Teraflops) aus den USA zu Rate gezogen, wodurch die benötigte Zeit auf weniger als 5 Tage reduziert werden könnte.

[45] http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/
[46] http://www.griphyn.org/
[47] http://www.euro-vo.org/
[48] http://www.us-vo.org/

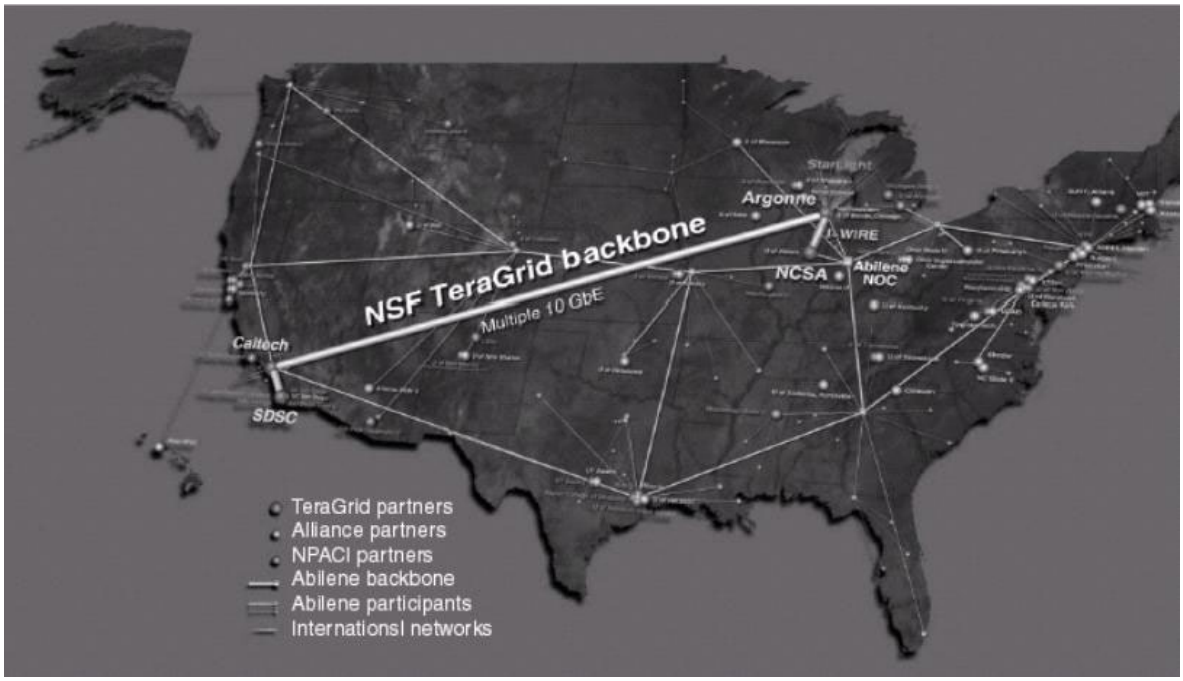


Abbildung 14: NSF TeraGrid Backbone

Zum anderen wird an der Errichtung so genannter virtueller Observatorien (VO – Virtual Observatory) gearbeitet. Als Beispiel sei das National Virtual Observatory (NVO) der USA genannt. In so einem virtuellen Observatorium werden alle erreichbaren astronomischen Daten sortiert, katalogisiert und archiviert. Diese Daten werden dann im Zusammenhang mit speziellen Werkzeugen zur Analyse und Suche komplett öffentlich zugänglich gemacht und die Daten werden in die Public Domain gegeben.

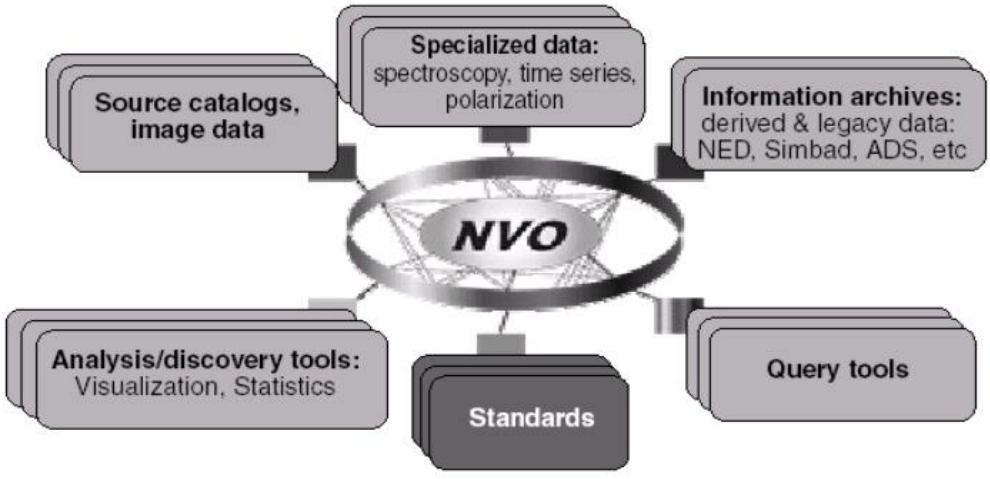


Abbildung 15: Architektur des NVO

Da es aber keinen Sinn machen würde, wenn es mehrere solcher VOs getrennt voneinander geben würde, haben sich einige der bedeutendsten dieser Projekte zusammengeschlossen um ein gemeinsames internationales virtuelles Observatorium zu errichten, so dass man einen einheitlichen Zugang zu allen Daten

erhält. Der Nutzen und die Arbeitsweise eines solchen VOs sei hier an zwei Beispielen kurz demonstriert, die auch als Prototyp auf der NVO Website abrufbar sind.

Gamma-Ray Burst Follow-Up Service Science Prototype

Eine schnelle Sammlung von Bilder-, Katalog- und Observationsdaten aller Wellenlänge im nachhinein eines interessanten Ereignisses ist essentiell in diesem Bereich der Wissenschaft. Dieser Dienst ist genau dafür ausgelegt, und kann auch dazu benutzt werden, um schnell alle verfügbaren Daten für jeden Bereich des Himmels zu erhalten. Momentan werden Daten verschiedener Teleskope (zurzeit 13), inklusive Bilder, Objektkataloge und Observationslisten hier gesammelt und in einem einheitlichen und leicht verständlichen Informationssatz verfügbar gemacht. Es ist dadurch möglich viele verschiedene Visualisierungswerkzeuge auf diese Daten anzuwenden.

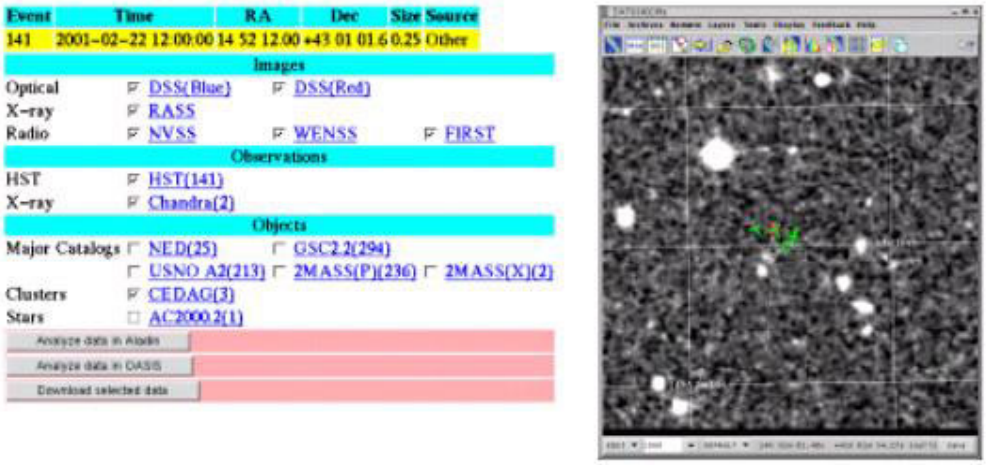


Abbildung 16: Webportal für die Suche der benötigten Informationen

Der Ablauf einer Suche könnte folgendermaßen aussehen:

- Ein Satellit observiert einen Gammastrahlenblitz und sendet die errechneten Daten zurück zur Erde.
- Innerhalb von Sekunden sucht das VO Archivbilder / -kataloginformationen aus allen bekannten Katalogen für diese bestimmte Region des Himmels, die in allen Ecken der Welt verstreut liegen.
- Danach wird eine Relationsbeziehung der verschiedenen Quellen generiert und eine komplette Liste aller bekannten Objekte dieser Region erstellt.
- Jetzt werden die normalen (wissenschaftlich uninteressanten) variablen Sterne und Himmelskörper identifiziert.
- Schließlich bereitet das VO eine Website mit Links zu den verfügbaren Informationen über die Objekte vor und stellt sie auf der Website zur Verfügung und setzt die Astronomen über die Gelegenheit, Folgeuntersuchungen mit ihren Teleskopen in anderen, bisher nicht katalogisierten Wellenlängen diesem Gebiet zu unternehmen, in Kenntnis.

- Ebenfalls interessant ist dies im Zusammenhang mit wissenschaftlichen Forschungsarbeiten, da die Forscher, quasi sofort nachdem sich etwas verändert, auf dem neuesten Stand sind.

Galaxy Morphology Science Prototype: A Case Study in Grid Computing

Bei diesem Prototyp kann man die Oberflächenhelligkeit und andere morphologische Werte einer Galaxie über einen längeren Zeitraum betrachtet heraussuchen. Durch die Verwendung von Grids ist es hier zum erstem Mal möglich, mehrere Datensätze mit einer Berechnung in Echtzeit zu verbinden, wobei sich der Benutzer sein Visualisierungsportal (z.B. Aladin, OASIS, DS9) selbst aussuchen kann. Im Folgenden soll nun der Ablauf einer Anfrage mit Hilfe einiger Bilder und beschreibendem Text näher erläutert werden. Dabei stehen die Zeichen des NVO ( ) auf der Landkarte jeweils für die beteiligten Rechner, wobei die Webportal-Server an der Ostküste und in Seattle stehen:

Abbildung 17: Anfrageauswertung bei den NVO (1)

Zuerst startet der Benutzer seinen Webbrowser und sucht sich den Cluster aus, über den die Analyse durchgeführt werden soll. Die Mitglieder dieses Clusters werden vom Portal herausgesucht und deren Position wird in einer internen Liste gespeichert. Danach fragt das Portal die „Simple Image Access“ Server bei Chandra, SkyView und dem DSS um Bilder der Mitgliedergalaxien des gewählten Clusters zu bekommen.

Abbildung 18: Anfrageauswertung bei den NVO (2)

Nachdem alle Bilder gesammelt wurden, startet der Benutzer die Analyse. Dabei werden zuerst die photometrischen Parameter (entweder aus dem NED oder dem CNOC Katalog, je nach Typ des Clusters) herunter geladen.

Abbildung 19: Anfrageauswertung bei den NVO (3)

Abbildung 20: Anfrageauswertung bei den NVO (4)

Der Server nimmt nun alle gesammelten Informationen und fügt sie zu einem Masterkatalog der Bilder und Photometriedaten zusammen. Dieser Katalog wird nun an den Grid-Dienst geschickt, der die morphologischen Parameter für alle Mitglieder des Clusters in diesem Katalog berechnet.

Schließlich werden die Resultate dieser Berechnungen auf dem Server gespeichert und der Benutzer kann sie sich herunterladen, um sie mit dem Analysewerkzeug seiner Wahl anzusehen und zu verwerten.

Abbildung 21: Anfrageauswertung bei den NVO (5)

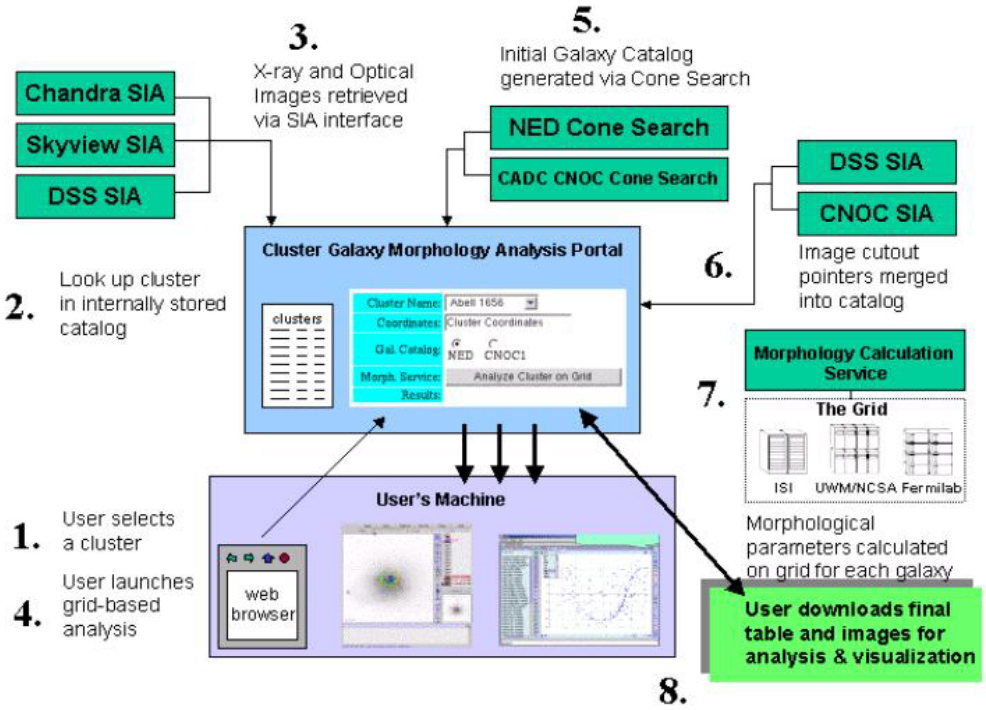


Abbildung 22: Ablauf einer Anfrage beim NVO

Wie man gesehen hat, wird in beiden Fällen viel Wert darauf gelegt, dass Daten aus unterschiedlichen Quellen, die weit verteilt sind, in möglichst kurzer Zeit gesammelt und analysiert werden, wozu jeweils immer Grid-ähnliche Infrastrukturen benutzt werden. Die Astronomie der Zukunft, bei der es Teleskope geben wird, bei denen täglich mehrere Terabyte bis sogar Petabytes an Daten anfallen, ist somit ein weiteres Gebiet der Wissenschaft und Forschung, welches Grids ins Auge gefasst hat.

7 Kommerzielle Anwendungen

Wenden wir uns nun kommerziellen Grid-Anwendungen zu. Man kann auch hier in verschiedene Kategorien unterteilen. So werden zum einen On-Demand Dienste angeboten und zum anderen das Grid an sich. Des Weiteren gibt es verschiedene Ansätze, die das Grid benutzen, um mehr Rechenleistung zu erreichen und somit auch mehr Geld verdienen zu können. Dazu zählen zum Beispiel Online-Multiplayer-Games, wo der Kunde dafür bezahlt, am Spiel teilnehmen zu können und mit Hunderten anderen Spielern zu spielen. Das Grid steht hier eher im Hintergrund und ist nur Mittel zum Zweck. Es wäre zum Teil nicht anders möglich, die Vielzahl an Spielern zu koordinieren. Diese Projekte sollen hier aber nicht näher betrachtet werden.

7.1 On-Demand Services

On-Demand Dienste verfolgen einen etwas anderen Ansatz. Auch hier wird das Grid nicht vom Dienstnutzer, sondern vom Dienstanbieter verwaltet. (Im Folgenden sollen die Begriffe Nutzer und Anbieter analog gebraucht werden.) Es soll nun untersucht werden, inwiefern sich für die beiden Seiten Vor- und Nachteile ergeben und welchen Einfluss ein On-Demand Konzept haben könnte.

Zunächst sei dazu erklärt, was On-Demand Dienst eigentlich bedeutet. Stark vereinfacht werden Dienste (Rechenleistung, Speicherkapazität, Nachrichtendienste usw.) sowie deren Verwaltung durch ein externes Unternehmen, den Dienstanbieter, erbracht. Dabei soll der Nutzer bei Bedarf auf den entsprechenden Dienst zugreifen können. Im Idealfall ist das jederzeit für jeden Nutzer möglich. Der Nutzer muss für die Diensterbringung aufkommen. Das heißt, wenn der Nutzer z. B. beim Computing-On-Demand CPU-Zeit kauft, so muss der Anbieter diese Zeit zur Verfügung stellen, wenn der Nutzer sie braucht und der Nutzer entsprechend dafür bezahlen.

Aus Sicht des Anbieters entsteht so ein völlig neues Marktsegment, in dem es auf den ersten Blick sehr einfach zu sein scheint, viel Geld zu verdienen. Die Unternehmen, welche On-Demand Dienste anbieten wollen, sind i.A. Hersteller von Hard- und Software wie z. B. IBM und HP. Sie gehen an dieser Stelle dazu über, das System nicht mehr als solches zu verkaufen, sondern nur zeitweise. Derselbe Rechner kann dann an mehrere Nutzer verkauft werden, indem er beim Dienstanbieter steht und die Nutzer darauf zugreifen können. Dabei ist nicht nur die Rechenleistung zu beachten, es gibt auch andere Dienste, die eine Rolle spielen. Aus Sicht des Dienstanbieters ist diese Situation vorteilhaft, da die Nutzer nicht ein Mal das komplette System kaufen, sondern oft einen Teil des Systems für einen bestimmten Zeitraum.

Das Konzept ist jedoch nicht ganz so einfach. Zuerst müssen Nutzer gefunden werden, die den entsprechenden Dienst auch benutzen möchten. Das stellt sich momentan als sehr großes Problem dar. Zum einen wollen die potentiellen Nutzer häufig eine Lösung, die sowohl Hardware als auch Software und Support beinhaltet, zu anderen sind dieselben Nutzer im Allgemeinen der Meinung, dass dieses System unter der eigenen Regie zum Einsatz kommen sollte.

Aus der Sicht der Dienstanbieter stellt sich die Situation für den Dienstnutzer dennoch als positiv dar. Er kann einen Teil der Ressourcen, die er für die Verwaltung seiner eigenen Computersysteme benötigt, einsparen, indem er einen On-Demand Anbieter für die entsprechenden Dienste bezahlt. Somit braucht der Nutzer nicht mehr für die komplette Hard- und Software zu bezahlen, sondern nur für den Dienst als solches. Das hat den Vorteil, dass Kosten nur dann entstehen, wenn ein Dienst erbracht wird und nicht dann, wenn der Rechner angeschafft wird, der später den Dienst erbringen soll. Auf diese Weise kann der Nutzer auf zweierlei Arten Kosten senken. Einerseits braucht er keinen Rechner anschaffen, den er nicht auslasten kann; schließlich kann man ja auf den Rechnern des Dienstanbieters rechnen lassen. Andererseits werden die Kosten für die Administration des (komplexen) Computersystems gespart, da diese auf den Dienstanbieter mitsamt dem Computersystem verlagert wurde. Die Lage stellt sich also auch aus der Sicht der Dienstnutzer positiv dar.

Trotzdem sind die momentan fehlenden Dienstnutzer ein ernstzunehmendes Problem. Es gibt einfach keine oder nur sehr wenige. Das liegt hauptsächlich daran, dass kein Unternehmen der Welt interne Betriebsgeheimnisse verraten will. Mit der Nutzung eines On-Demand Dienstes kann es aber sein, dass auf dem Weg durch das Internet (dieser muss eingeschlagen werden, damit das Konzept überhaupt Sinn macht) Informationen gelesen werden. Wenn solche Informationen der Konkurrenz zur Verfügung gestellt werden, können sie den Konkurs für das betreffende Unternehmen bedeuten, falls sie bestimmte Betriebsgeheimnisse beinhalten. Unternehmen tendieren deshalb eher dazu intern ein eigenes Grid zu betreiben.

7.2 Sun Grid Engine

Sun Grid Engine[49] bietet genau diese Lösung. Es wird hier kein Dienst angeboten, sondern Grid-Software verkauft. Sun Grid Engine gibt es in zwei verschiedenen Versionen. Frei zum Download verfügbar ist SGE (Sun Grid Engine), kaufen kann man SGEEE (Sun Grid Engine Enterprise Edition). Sun unterteilt Grids in drei Klassen: Cluster, Enterprise und Global Grid.

Cluster Grid ist die kleinste Form des Grids. Es besteht innerhalb einer administrativen Domäne und wird als eine Ressource betrachtet, die verschiedene Jobs ausführen kann. Enterprise Grid ist die nächstgrößere Stufe. Hier können administrative Grenzen überschritten werden, aber das Grid bleibt innerhalb eines Unternehmens. Hier können die Ressourcen auch örtlich getrennt liegen. Global Grid, als einen Zusammenschluss mehrerer Enterprise Grids, bezeichnet den Fall, dass verschiedene Unternehmen miteinander arbeiten und zusammen ein Grid nutzen. Sun Grid Engine ist dabei eher für kleine Grids (Cluster), die Enterprise Edition für große Grids (Enterprise- oder Global-Grid).

Sun Grid Engine ist eine Globus-basierte Anwendung des Grids. Es wird nicht das Grid als Dienst verkauft wie bei On-Demand Diensten, sondern Grid-Technologie an sich. Es gibt dazu einige Fallstudien, die hier beispielhaft betrachtet werden sollen.

[49] www.sun.com/software/gridware

Ford Motor Company verwendet Sun Grid Engine für die Entwicklung neuer Motoren. Dabei wird vor allem vom High-Perfomance-Computing, welches durch Sun Grid Engine unterstützt wird, Gebrauch gemacht.

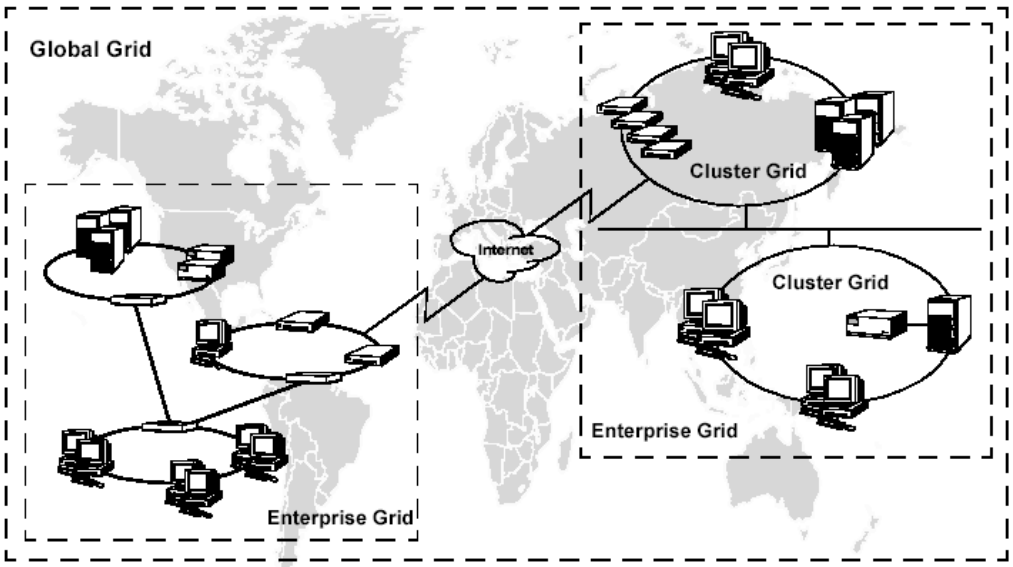


Abbildung 23: Grid-Klassifizierung nach Sun

Das Problem für Ford bestand darin, einerseits die Kosten zu senken und andererseits die Rechenleistung zu erhöhen, also die Standards wenigstens beizubehalten. Das Department „Engine & Transport“ sollte zum Beispiel die Kapazitäten erhöhen, aber dabei die bereitstehenden Ressourcen verwenden. Somit mussten diese Ressourcen besser genutzt werden. Da Ford schon seit längerer Zeit mit Sun zusammenarbeitet, schlug Sun eine Grid Computing Architektur vor. Es sollten so die CPU-Zyklen besser ausgenutzt werden: Während der Arbeitszeit sollten interaktive Tasks auf den Prozessoren laufen, in der Nacht und an Wochenenden rechenintensive Prozesse. Somit konnte das bestehende Potential besser ausgenutzt werden.

Ein anderes Fallbeispiel ist Motorola. Hier wird Sun Grid Engine eingesetzt, um das Chip-Design zu beschleunigen. Das Problem stellt sich sehr ähnlich zu dem dar, das Ford hatte: die vorhandene Hardware sollte effizienter ausgenutzt werden. Die eigenen Ressourcen sollten an der Leistungsgrenze betrieben werden und zusätzlich die Ressourcen anderer Abteilungen. Als Lösung diente dabei Sun Grid Engine, welche die Ressourcen in einen virtuellen Ressourcen-Pool bündelte. Somit konnten die Ingeneure über die notwendigen Ressourcen verfügen. Um diesen Prozess zu beschleunigen, wurde für die Ingeneure ein Training angeboten, bei dem ihnen beigebracht wurde, wie und wozu sie Sun Grid Engine benutzen konnten. Auf diese Weise wurden die Ressourcen von Beginn an optimal genutzt. Dieses Szenario hatte für Motorola einige Vorteile. Man konnte ohne zusätzliche Kosten die vorhandenen Ressourcen besser ausnutzen, das heißt, es gab weniger ungenutzte CPU-Zyklen bei den bereits vorhandenen Maschinen - somit sind in einer Zeitspanne mehr Tests durchführbar als vorher. Ein weiterer Vorteil liegt in der Fehlertoleranz. Wenn früher ein Computer abgestürzt ist, musste der Prozess, der gerade bearbeitet wurde, neu gestartet werden. Dadurch ging Rechenzeit verloren, da ein Teil des Jobs zweimal

ausgeführt werden musste. Dank Sun Grid Engine ist es aber möglich (mittels Checkpointing), den Job in einem solchen Fall auf eine andere Maschine zu übertragen, so dass weniger Rechenzeit vergeudet wurde. Auf diese Weise war die Abteilung in der Lage, die Deadline einzuhalten und den Chip auf den Markt zu bringen. Gleichzeitig zeigten sich andere Teile des Unternehmens an der neuen Technologie interessiert, so dass sich ein Vorteil für das gesamte Unternehmen ergab.

Ein drittes Beispiel ist RiboTargets, ein britisches Pharmazie-Unternehmen. Hier wird Grid-Technologie eingesetzt, um neue Antibiotika zu finden, die helfen sollen HIV, Hepatitis C u. a. zu bekämpfen. Da es eine sehr große Menge an möglichen Kandidaten gibt, werden diese von einer Software vorberechnet und anschließend diejenigen, welche am meisten versprechen, im Labor näher untersucht. Auf diese Weise kann man das Problem deutlich effizienter angehen. Leider ist dieses Problem sehr rechenintensiv, so dass auch hier eine Grid-Lösung benutzt wurde. Auf diese Weise wurde in der Vergangenheit ca. ½ Terabyte an Daten produziert, die jetzt ausgewertet werden müssen.

Zu kommerziellen Grid-Anwendungen kann man abschließend sagen, dass die Nachfrage bzgl. der Software-Lösung durchaus besteht. Andererseits sind die Unternehmen daran interessiert, ihre technologischen Innovationen geheim zu halten. Somit ist die Nachfrage bei On-Demand Diensten eher gering einzuschätzen. Vielleicht kann sich Messaging On Demand als Dienst etablieren, da hier möglicherweise der Austausch von sicherheitskritischen Informationen nicht stattfindet.

8 ClimatePrediction.net

Abschließend soll noch eine Grid-Anwendung betrachtet werden, die im Zusammenhang mit dem täglichen Leben eine Rolle spielen kann. Es geht dabei um Wettervorhersagen. Die hier betrachtete Software ist allerdings nicht direkt auf das Grid zugeschnitten, sondern kommt eher aus dem Bereich weitverteiltes Rechnen. Prinzipiell ist die Wettervorhersage jedoch wegen ihrer Komplexität durchaus eine Grid-Anwendung, da es bisher nicht möglich ist, das Wetter in Echtzeit vorherzusagen. Besser wäre es sicherlich, wenn man schneller ist, als sich das „originale“ Wetter entwickeln kann.

Das, hier betrachtete, ClimatePrediction.net[50] Experiment, soll für Klimamodelle, bestimmte Parameter näher eingrenzen. Somit sollen Wettervorhersagen genauer möglich sein. Auf dieser Basis soll dann das Klima des nächsten Jahrhunderts vorhergesagt werden. Dabei soll zunächst überprüft werden, inwieweit das verteilte Rechnen funktioniert (Experiment 0). Anschließend wird auf dieser Basis ermittelt, wie sich der Teil des Klimamodells verhält, der die Atmosphäre betrachtet. Dazu soll ein vereinfachtes Ozeanmodell benutzt werden. Auf diese Weise kann man Veränderungen, welche die Atmosphäre betreffen, schneller bemerken. Das heißt, die Atmosphäre verändert sich schneller, wenn man ihre Parameter verändert. Im Experiment sollen die Werte für die Treibhausgase betrachtet werden (Experiment 1). In einem weiteren Experiment sollen dann verschiedene Repräsentationen der zweiten Hälfte des 20. Jahrhunderts berechnet werden. Dazu sollen im Jahr 1950 unter gleichen Startbedingungen verschiedene Testläufe gestartet werden. Nach einigen Jahren werden Parameter verändert (für jeden Testlauf etwas anders) und anschließend wird mit diesen Parametern als Startbedingungen ein neuer Testlauf zu einer späteren Zeit gestartet. Führt man dieses Experiment bis zum Ende des 20. Jahrhunderts aus, so erhält man verschiedene Repräsentationen, die alle auf der gleichen Grundlage (den Startbedingungen) erzeugt wurden. Auf diese Weise kann man untersuchen, wie sich das Klima für sechs Monate bzw. mehrere Jahre vorhersagen lässt (Experiment 2). Im Hauptexperiment soll dann auf der Grundlage der anderen 3 Experimente die Vorhersage für die nächsten 100 Jahre gemacht werden.

Auch wenn das hier beschriebene Projekt nicht in erster Linie etwas mit Grid Computing zu tun hat (es ist ja eine weitverteilte Anwendung), kann es trotzdem eine gute Grundlage bieten. Die hier bestimmten Parametergrenzen für das verwendete Klimamodell könnten später in einer Grid-Lösung zur genaueren Wettervorhersage benutzt werden. Dadurch würde vielleicht auch ein Geschwindigkeitsvorteil entstehen, da nicht mehr so viele verschiedene Möglichkeiten existieren. Ob dieser Fall jedoch eintreten wird, ist zurzeit fragwürdig.

[50] www.climateprediction.net

9 Zusammenfassung

Zusammenfassend kann man sagen, dass auch Grid-Anwendungen ihre Grenzen haben. Diese sind zum einen durch bestimmte Gesetzmäßigkeiten (Amdahls Law, etc.), zum anderen durch Kommunikation zwischen den einzelnen Knoten des Grids bedingt.

Amdahls Law besagt, dass man durch massives Parallelrechnen nicht unbedingt einen Geschwindigkeitsvorteil haben muss. Man betrachte hierzu folgendes Beispiel. Ein Programm bestehe aus 100 Fragmenten. Von diesen Fragmenten können 20 nur sequentiell verarbeitet werden, die restlichen 80 können parallel verarbeitet werden. Nimmt man weiter an, dass eine Maschine mit 80 Prozessoren zu Verfügung steht, so kann man das Problem in optimaler Zeit lösen. Verdoppelt man die Prozessorzahl so bleibt die Verarbeitungszeit konstant, da maximal 80 Prozessoren parallel genutzt werden können.

Zum anderen ist es so, dass durch paralleles Rechnen der Kommunikationsaufwand erhöht wird. Das lässt sich dadurch erklären, dass die einzelnen Fragmente des Programms auf verschiedenen Prozessoren koordiniert verteilt werden müssen. Anschließend muss das Ergebnis aber auch wieder richtig zusammengefügt werden. Dazu ist ein gewisser Kommunikationsaufwand nötig. Somit kann es u. U. vorkommen, dass ein Prozess, der lokal auf einem Prozessor verarbeitet wird, schneller zum Ende kommt als ein äquivalenter Prozess, der mehrere Prozessoren und i. A. auch mehrere Computer benutzt. Dieser Fall wird insbesondere dann eintreten, wenn die Latenzzeit des verwendeten Netzwerkes hoch ist. Im Allgemeinen ist jedoch der parallele Ansatz oft schneller als der sequentielle.

Ein anderes Problem der Grid-Anwendungen ist es, dass z. T. die Grid-Technologie noch nicht ausgereift ist. Das Problem liegt vor allem darin, Programme zu schreiben, die auf einer bestimmten API aufsetzen, die jedoch selbst noch in der Entwicklung sind. Somit verändert sich die Schnittstelle, gegen die der Anwendungsprogrammierer programmiert und das Entwickeln der Software wird dadurch zur Qual. Leider resultiert daraus noch ein anderes Problem: Die Geldgeber der Grid-Technologie könnten sich fragen, warum sie in eine Technologie investieren, die keiner braucht. Schließlich gibt es ja auch keine Anwendungen dafür. Man sieht hier, dass sowohl das Grid als auch seine Anwendungen miteinander wachsen. Obwohl hier wohl noch viel Arbeit zu erledigen ist, gibt es aber trotzdem schon vielversprechende Grundlagen wie z. B. das e-Science-Projekt. Allerdings kann sich zum heutigen Zeitpunkt kein Projekt, das Grid-Technologie entwickelt, selbst tragen, sie alle sind davon abhängig, dass sie einen Sponsor haben. Das beste Beispiel ist hier wohl EU-Data-Grid, bei dem die Förderung Ende des Jahres auslaufen wird. Das Projekt kann nur dann weiterbestehen, wenn weitere Fördermittel genehmigt werden oder sich ein anderer Sponsor findet.

10 Quellen

Anmerkung: Die Verweise auf die primäre Homepage der Projekte sind jeweils als Fußnoten in den einzelnen Kapiteln angegeben und hier nicht noch einmal extra aufgeführt.

[1] Grid Computing: Making the Global Infrastructure a Reality Anthony J. G. Hey, Geoffrey Fox, Fran Berman, www.grid2002.org
[2] Butterfly Grid for Multiplayer Games, http://www.butterfly.net/
[3] Everquest Multiplayer Gaming Environment, www.everquest.com
[4] On-Demand-Services: www.theITportal.com, www.outsourcing-law.com, www.hp.com

ISBN 3-937786-28-7
ISSN 1613-5652