TECHNISCHE UNIVERSITÄT ILMENAU
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik
Fachgebiet Telematik
Prof. Dr. Dietrich Reschke

Hauptseminar Telematik
Sommersemester 2003

Thema:
Das Grid. Einordnung einer Technologie für verteiltes Rechnen

Betreuer: Dipl.-Inf. Thorsten Strufe

vorgelegt von:
Thomas Sesselmann
Thomas.Sesselmann@stud.tu-ilmenau.de

Inhaltsverzeichnis

1 Der Power-Grid-Vergleich **4**

2 Definition **5**

3 Idee **7**

3.1 Virtuelle Organisation 8

3.2 API 8

3.3 SDK 8

4 Architektur **9**

4.1 Fabric Layer 10

4.2 Connectivity Layer 12

4.3 Resource Layer 13

4.4 Collective Layer 14

4.5 Application Layer 15

4.6 Beispiel 15

5 Klassifikation **16**

6 Fazit **20**

1 Der Power-Grid-Vergleich

Die Idee des Grids ist eine zusätzliche allgemein verfügbare Ressource zu schaffen, ähnlich dem Strom. Demnach soll es später möglich sein, seine Rechenleistung wie Strom aus einer Steckdose zu beziehen. Dann ist es egal, wo die Rechenleistung herkommt oder wie sie an die richtige Stelle gelangt, es ist nur noch notwendig eine einheitliche Schnittstelle zu besitzen. Am Ende des Monats wird dann zum Beispiel eine Rechnung über die gerechneten CPU-Cycles fällig.

Zur Zeit ist die Entwicklung im Bereich Grid aber soweit, wie damals 1910 bezüglich des Stroms. Jeder, der Strom benötigte, besaß seinen eigenen (Strom-)Generator. Mit der Entwicklung des Stromnetzes, auch Power-Grid genannt, wurde ein großer Fortschritt erzielt. Jeder konnte seinen Strom auch ohne eigenen Generator beziehen und auch die überschüssige Leistung an andere verkaufen. Aktuell besitzt jeder, der für eine Berechnung einen Computer braucht, einen eigenen Rechner oder Großrechner (je nach Aufgabe). Nun soll das Grid einen ähnlichen Fortschritt mit der Verteilung der Rechenleistung erzielen, wie damals das Stromnetz mit dem Strom.

2 Eine spezielle Definition

Foster und Kesselman geben folgende Definition in [2] vor: “A grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.”

- **“infrastructure”**:
  Das Grid bietet eine große Vielfalt an angebotenen Ressourcen (z.B. Computer, Daten, Sensoren, ...). Die notwendigen Verbindungen können nur mit einer guten Hardwareinfrastruktur gesichert werden. Eine gute Softwareinfrastruktur wird benötigt, um die Ergebnisse zu kontrollieren und anzeigen zu können.

- **“dependable”**:
  Damit ist die Zuverlässigkeit gemeint, also eine Garantie für die Funktionalität. Die Nutzer des Grids brauchen Zusicherungen, dass sie vorhersagbare, konstante und oft auch hohe Performanz verschiedener Komponenten erhalten (also die komplette Implementierung von QoS[1]). Programme ohne solche Zusicherungen werden nicht geschrieben oder benutzt. Die Performance ist von Applikation zu Applikation unterschiedlich wichtig, aber meistens sind Netzwerkbandbreite, Wartezeiten, Rechenleistung, Softwaredienste, Sicherheit und Vertrauenswürdigkeit dabei.

- **“consistent”**:
  Die Konsistenz bedeutet, dass überall die gleichen Dienste erreichbar sind, unabhängig vom Ort oder der Hardware, mit der sich verbunden wurde. Mit anderen Worten benötigen wir Standarddienste, die mit Standardschnittstellen erreichbar sind und mit Standardparametern arbeiten. Ohne solche Konsistenz wäre ein durchdringendes Grid nicht denkbar (siehe nächsten Punkt). Zur Entwicklung solcher Standards ist es notwendig die Heterogenität zu fördern, ohne die Performanz einzuschränken.

- **“pervasive”**:
  Eine durchdringender Zugriff erlaubt einer Menge von Diensten immer erreichbar zu haben, egal in welcher Umgebung wir uns befinden. Damit ist aber nicht gemeint, dass jeder zu jeder Zeit Zugriff auf die Gridressourcen besitzt, sondern es müssen erst ein Account und Anschlüsse existieren (in einem neuen Haus gibt es auch noch keinen Strom).

[1] Quality of Service

- **“inexpensive”**:
  Die Gridinfrasturktur muss mit zumutbaren Kosten zugänglich sein, sonst wird sie nicht allgemein anerkannt werden.

- Ebenfalls ist es notwendig eine nahtlose Integration von Ressourcen zu gewährleisten.

3 Idee

Die Idee des Grids ist ein koordiniertes Ressource-Sharing und Problemlösen in dynamischen, multiinstitutionalen Virtuellen Organisationen[2]. Diese Virtuellen Organisationen sind sehr dynamisch und werden im Bedarfsfall für einen bestimmten Zweck ins Leben gerufen. Es soll ein direkter Zugriff auf alle erdenklichen Ressourcen möglich sein, wie zum Beispiel Rechner, Softwaredaten, Sensoren, Elektronenmikroskope und vieles mehr.

Eine einheitliche Schnittstelle wird mit Hilfe einer Middleware oder SDKs[3] (mit APIs[4]) erreicht.

Die Middleware liegt zwischen Betriebssystem und der Anwendung und bietet so der Anwendung eine einheitliche Schnittstelle, die von dem darunterliegenden Betriebssystem unabhängig ist.

Besondere Herausforderungen an das Grid sind folgende:

- Interoperabilität
- Hochverfügbarkeit
- hoher Durchsatz
- Sicherheit
- Vertraulichkeit
- Datenreplikation
- Jobscheduling
- Scavenging[5]
- einheitliche Schnittstelle zu den Ressourcen
- Heterogenität

[2]VO: siehe Abschnitt 3.1
[3]SDK: Software Development Kit (siehe Abschnitt 3.3)
[4]API: Application Programming Interface (siehe Abschnitt 3.2)
[5]Scavenging: Suche nach freien Ressourcen

3.1 Virtuelle Organisation

Eine Menge von Institutionen mit Zugriffsregeln bildet eine Virtuelle Organisation (VO). Innerhalb dieser VO definieren Sharing-Regeln, wer, wann, wie auf welche Ressource zugreifen kann bzw. darf. Also bedeutet eine VO ein "Sharing" von Ressourcen. Eine VO ist sehr dynamisch und wird im Bedarfsfall für einen bestimmten Zweck ins Leben gerufen. Das bedeutet, es kann jederzeit eine Sharing-Regel oder eine Institution hinzugenommen oder entfernt werden. Innerhalb einer VO existiert keine Zentrale, und niemand ist allwissend. Das bedeutet jeder ist auf jeden angewiesen, um ein gemeinsames Problem zu lösen. Da sich das Grid in bestehende Sicherheitssysteme integriert, kann jede Institution ihr eigenes Sicherheitssystem besitzen.

Zum Beispiel können Institutionen mit gleichen Interessen teure Ressourcen sparen, indem nicht jeder alle Ressourcen einzeln braucht, sondern jeder auf jede Ressource zugreifen kann (zum Beispiel: Das EU-Datagrid wird näher erklärt in Abschnitt 5).

3.2 API

Eine "Application Programming Interface" (API) definiert eine Standardschnittstelle, also eine Menge von Unterprogrammen oder Objekten mit Methodenaufrufen, um eine bestimmte Funktionalität zu ermöglichen. Eine API gibt ein standardisiertes Verhalten an, kann aber mehrere Implementierungen besitzen.

Um den Zusammenhang von APIs mit Protokollen zu verstehen: Eine API kann mehrere Protokolle benutzen und implementieren, aber ein Protokoll sagt nichts über die Implementierung einer API (für Protokollnachrichten) aus. Standard-APIs ermöglichen Portabilität, Standardprotokolle hingegen Interoperabilität.

3.3 SDK

Ein "Software Development Kit" (SDK) ist eine Menge von Code, die in einem Programm eingelinkt werden kann. Das Programm kann so die Funktionen des SDKs benutzen. Ein SDK ist typischerweise eine Implementierung einer API. Wenn eine API verschiedene Implementierungen zuläßt, gibt es meist auch mehrere (verschieden implementierte) SDKs. Einige SDKs benutzen Protokolle und geben dem Programm die Möglichkeiten die Dienste eines Protokolls in Anspruch zu nehmen, aber ein SDK muss nicht unbedingt ein Protokoll verwenden.

4 Architektur

Angelehnt an [1] existiert folgendes Architekturkonzept:
Die Architektur des Grids ist in Layers aufgeteilt, ähnlich der Internet-Protokoll-Architekur. Jede Komponente jeder Schicht besitzt Merkmale und Fähigkeiten, die auf Fähigkeiten von tieferen Schichten aufbaut und den oberen Schichten anbietet. Bei der Grid-Protokoll-Architektur ist es allerdings im Gegensatz zur Internet-Protokoll-Archtektur möglich nicht nur auf die Fähigkeiten der nächst tieferen Schicht zuzugreifen, sondern auch direkt auf tiefere Schichten [6]. Zu sehen ist das Schichtenmodell in Abbildung 1.

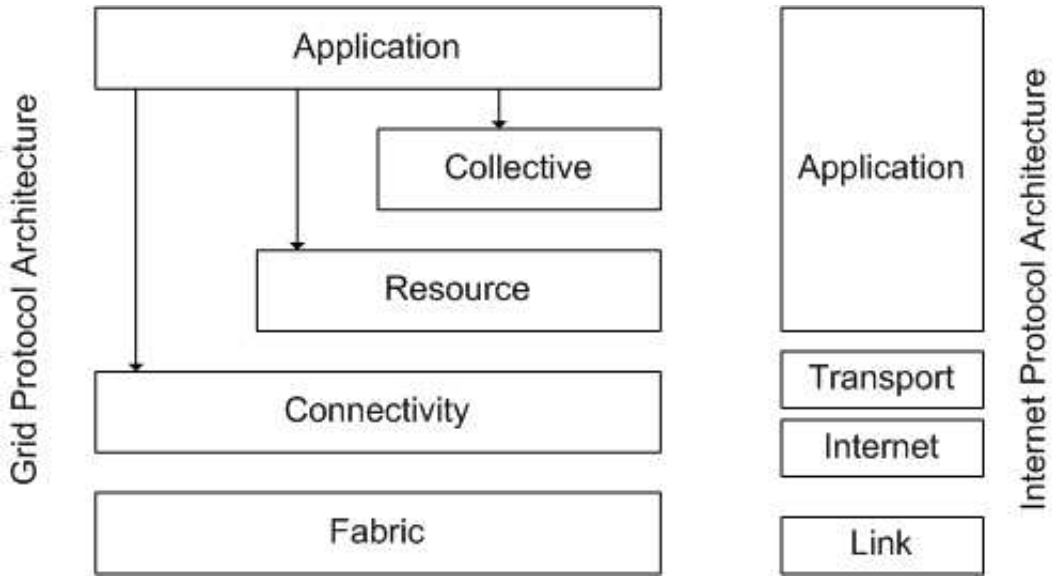


Abbildung 1: Grid-Protokoll-Architektur

Auch findet wie bei den Internetprotokollen das Sanduhrmodell Anwendung (siehe Abbildung 2). Im Hals der Sanduhr befindet sich ein kleiner überschaubarer Teil von Kernprotokollen. Auf diese Kernprotokolle können höherwertige Handlungsweisen abgebildet werden (oberer Teil der Sanduhr). Der Hals der Sanduhr benutzt die darunterliegenden Protokolle. Da der obere und untere Teil der Protokollschichten mit diesen überschaubaren Kernprotokollen verbunden wird, ist es ganz einfach möglich die oberen und unteren Schichten beliebig zu erweitern (solange der Hals beibelassen wird). Beim Internet ist das Kernprotokoll "IP" und bei dem Grid sind es die "Resource"- und "Connectivity"-Protokolle.

[6] Eine Ausnahme bildet nur das Fabric-Layer, das nur vom Connectivity-Layer angesprochen werden darf.

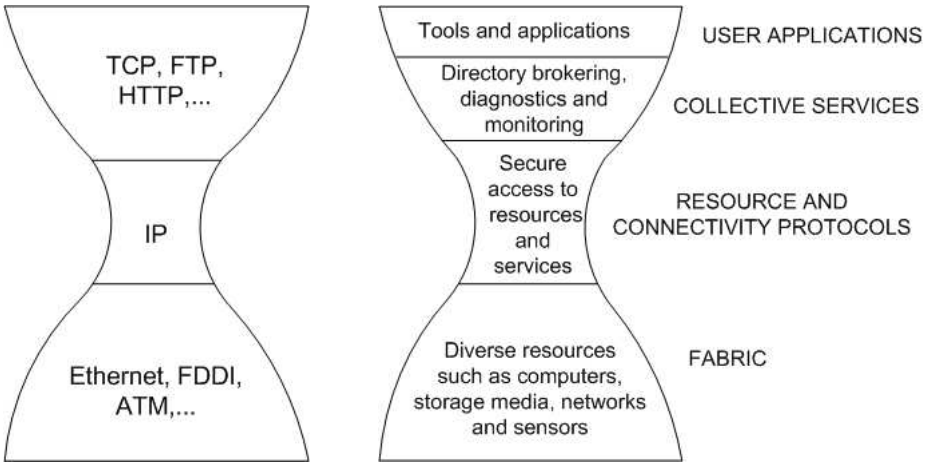
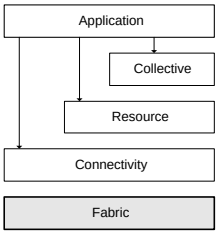


Abbildung 2: Das Sanduhrmodell des Internets und des Grids

4.1 Fabric Layer



Das Fabric-Layer stellt die Ressourcen für den gemeinsamen Zugriff von Grid-Protokollen bereit. Ressourcen können vielfältiger Natur sein, wie zum Beispiel Rechenleistung, Datenspeicherung, Kataloge, Netzwerke, Sensoren usw. Die Fabric-Komponenten implementieren die lokal vorkommenden und ressourcenspezifischen Operationen und bieten den höheren Layern eine einheitliche Schnittstelle für verteilte Operationen. Höhere Layer können dann beispielsweise einfach mit open, read, write, usw. auf die Ressourcen zugreifen. Die genaueren und ressourcenspezifischen Operationen übernimmt das Fabric-Layer. Bessere Fabric-Layer-Funktionalität ermöglicht mehrere fortgeschrittene verteilte Operationen zur gleichen Zeit. Zum Beispiel erlauben solche fortgeschrittenen Operationen höherwertigen Diensten, Ressourcen mittels anderen Strategien zuzuweisen und zu verteilen, die sonst nicht möglich wären (co-scheduling). Aber solche verbesserten Reservierungen erhöhen den Aufwand der Integration neuer Ressourcen in das Grid und deshalb existieren auch recht wenige.

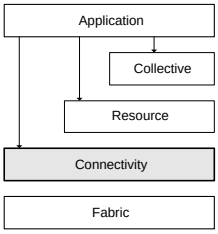
Als Minimum von Fabric-Layer-Funktionen sollte ein Abfragemechanismus für die Struktur, den Status und die Fähigkeiten der Ressource und ein Ressourcenmanagementmechanismus mit QoS implementiert werden.

Einige ressourcenspezifische Fähigkeiten:

- Rechenressourcen (computational resources)
  - Mechanismus zum Starten von Anwendungen
  - Überwachung und Kontrolle von Prozessen
  - Verwaltung von Ressourcen
  - erweiterter Reservierungsmechanismus
  - Erkennung von Hardware- und Software-Eigenschaften
  - Erkennung von wichtigen Statusinformationen (z.B. Last und Pufferinhalt)
- Speicherressourcen (storage resources)
  - Ziehen und Schieben von Dateien
  - Third-Party- und hoch-performante Datentransfers
  - Lesen und Schreiben von Dateiteilen
  - Ausführen von entfernten Dateiteilen
  - Verwaltung von Ressourcen für Datentransfers (Speicherplatz, Festplattenbandbreite, Netzwerkbandbreiten, CPU-Last)
  - erweiterter Reservierungsmechanismus
  - Erkennung von Hardware- und Software-Eigenschaften
  - Erkennung von wichtigen Statusinformationen (z.B. freier Speicherplatz, Bandbreitenausnutzung)
- Netzwerkressourcen
  - Mechanismus für die Kontrolle der Ressourcenbelegung für Netzwerktransfers (Priorisierung, Reservierung)
  - Feststellen von Netzwerkeigenschaften und Last
- Code-Repositories
  Diese spezielle Form von Speicherressourcen benötigt einen Mechanismus für die Verwaltung von versionsbehafteten Quellen und Code (z.B. CVS).

- Kataloge
  Diese spezielle Form von Speicherressourcen braucht einen Mechanismus für die Implementierung von Kataloganfragen und Updateoperationen (z.B. eine relationale Datenbank).

4.2 Connectivity Layer



Das Connectivity-Layer definiert Kern-Kommunikations- und Authentisierungsprotokolle, die für gridspezifische Netzwerktransaktionen benötigt werden. Kommunikationsprotokolle ermöglichen den Datenaustausch zwischen verschiedenen Fabric-Layer-Ressourcen. Die Authentisierungsprotokolle bauen auf den Kommunikationsdiensten auf, um Verschlüsselungsmechanismen für die Identitätsbestimmung von Nutzern und Geräten (Ressourcen) zu ermöglichen. Zur Kommunikation gehören hier Transport, Routing und Adressierung. Da hier zur Zeit keine besonderen Protokolle benötigt werden, nimmt man die schon vorhandenen Protokolle vom TCP/IP-Stack (z.B. TCP, UDP, IP, ICMP, DNS, OSPF usw.). Später kann es sein, dass das Grid andere spezielle Protokolle benötigt und benutzt.

Authentisierungsprotokolle für Virtuelle Organisationen sollten folgende Eigenschaften besitzen:

- **Single-Sign-On:**
  Ein Nutzer soll sich nur einmal einloggen müssen, um Zugang auf alle Ressourcen zu haben.

- **Delegation:**
  Ein Nutzer sollte in der Lage sein, ein Programm mit seinen Rechten laufen zu lassen. Dieses Programm kann dann mit den Rechten des Nutzers Ressourcen belegen, für die der Nutzer authorisiert ist. Außerdem kann das Programm auch Rechte oder Teilrechte an ein weiteres Unterprogramm weitergeben usw.

- **Integration in verschiedene lokale Sicherheitslösungen:**
  Da jeder Provider meist ein anderes Sicherheitssystem benutzt[7], sollte das Grid keine speziellen Voraussetzungen haben, sondern sich leicht in bestehende Sicher-

[7] z.B. Kerberos, Unix, ...

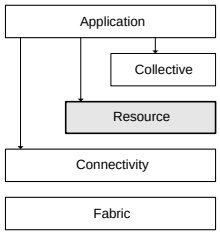
heitslösungen integrieren lassen. Das Grid bildet dann seine Sicherheitsregeln auf die lokalen ab.

- **Nutzerbasierte Vertrauensbeziehungen:**
  Ein Nutzer kann Ressourcen von verschiedenen Anbietern nutzen und sogar zusammenarbeiten lassen, ohne dass die Anbieter spezielle Sicherheitsregeln untereinander definiert haben.

Grid-Systeme sollten auch flexible Möglichkeiten zum Schutz der Kommunikation bieten (z.B. Grad des Schutzes definieren).

4.3 Resource Layer

Das Resource-Layer baut auf den Kommunikations- und Authentisierungsprotokollen des Connectivity-Layers auf und definiert Protokolle, APIs und SDKs für sichere Übertragung, Initiierung, Monitoring, Kontrolle, Accounting und Abrechnung für gemeinsame Operationen auf individuellen Ressourcen. Resource-Layer-Implementierungen von diesen Protokollen verwenden Fabric-Layer-Funktionen um die lokalen Ressourcen zu benutzen und zu kontrollieren.



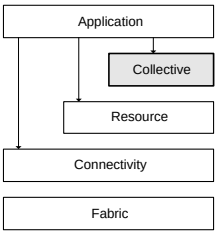
Es existieren zwei Arten von Resource-Layer-Protokollen:

- **Informationsprotokolle** werden verwendet um den aktuellen Status einer Ressource zu erhalten (Konfiguration, Last, Policy, Kosten, usw.).

- **Managementprotokolle** werden angewendet um mit den Ressourcen zu arbeiten, zum Beispiel Prozessgenerierung, Datenzugriff, Überwachung und Kontrolle der Ressourcen.

Da das Resource- und das Connectivity-Layer zusammen den Hals der Sanduhr bilden, sollte die Protokollanzahl dieser beiden Layer relativ klein und konstant bleiben. Sie müssen so gewählt werden, dass sie die grundlegenden Mechanismen vom verteilten Zugriff auf verschiedene Ressourcen erfassen, ohne die höheren Protokollebenen zu stark einzuschränken. Die Funktionen vom Fabric-Layer sind auch die Funktionen, die vom Resource-Layer benötigt werden, aber mit der Notwendigkeit von genau einer ("exactly once") Semantik für viele Operationen mit zuverlässigen Fehlerberichten.

4.4 Collective Layer



Das Resource-Layer definiert die Interaktionen zwischen einzelnen Ressourcen. Das Collective-Layer dagegen enthält Protokolle und Dienste (und APIs und SDKs), die nicht mit anderen Ressourcen verbunden sind, aber von globaler Natur und Verbindungen über Ressourcensammlungen sind. Die Collective-Komponenten können auf den Diensten und Protokollen von Resource- und Connectivity-Layer aufbauen und eine große Vielzahl von verteilten Handlungen definieren.

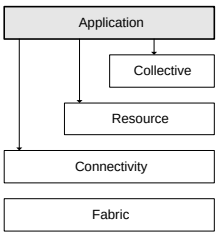
Zum Beispiel:

- **Verzeichnisdienste** erlauben VO-Teilnehmern die Existenz und die Eigenschaften von VO-Ressourcen zu verteilen. So können Nutzer die Ressourcen nach Namen oder Attributen suchen.
- **Co-allocation-**, **Scheduling-** und **Brokering-Dienste** erlauben VO-Teilnehmern eine oder mehrere Ressourcen für einen bestimmten Zweck zu reservieren.
- **Monitoring-** und **Diagnose-Dienste** dienen der Erkennung von Fehlern, Ausfällen, Überlast oder Angriffen.
- **Datenreplizierungsdienste** erlauben die VO-Datenressourcen zu verwalten, um den Zugriff zu maximieren.
- **Software-Discovery-Services** suchen die beste für die Softwareimplementierung und die Softwareausführung vorhandene Plattform, je nach den vorhandenen Problemparametern.
- **Gemeinschaftauthorisationsserver** bieten den Gemeinschaftsmitgliedern Zugriff auf die Gemeinschaftressourcen.
- **Gemeinschaftsbuchungs-** und **Gemeinschaftszahlungsdienste** sind für die Abrechnung der genutzten Ressourcen bzw. Dienste der Gemeinschaftmitglieder verantwortlich.

Wie diese Beispiele zeigen, gibt es eine große Vielfalt für solche Protokolle und Dienste. Diese Collective-Funktionen können als beständige Dienste mit den dazugehörigen Protokollen oder als SDKs implementiert werden.

4.5 Application Layer

Das letzte Layer in der Grid-Architektur enthält die Nutzerapplikationen, die in einer VO-Umgebung laufen. Auf jedem Layer existieren "wohldefinierte" Protokolle, die den Zugang für viele nützliche Dienste bieten (z.B. Ressourcenverwaltung, Datenzugriff, Ressourcenverteilung usw.). Für diese hat das Application-Layer, mit Ausnahme des Fabric-Layers, direkten Zugriff.



4.6 Beispiel

Ein beliebtes Beispiel ist im Bereich Gridcomputing immer Ray Tracing[8], da viel Rechenleistung benötigt wird und sich das Problem auch gut zerlegen läßt (Ausschnitte des zu berechnenden Bildes von anderen Rechnern berechnen lassen). Für diese Probleme werden vor allem Datenspeicher, Rechenleistung und Netzwerke benötigt, die vom Fabric-Layer verwaltet werden. Das Connectivity-Layer sorgt für die ordnungsgemäße Authentisierung und Verteilung der Daten über das Netzwerk an die Rechner. Der direkte Zugang zu den Ressourcen (Datenspeicher, Rechner usw.) erfolgt aber mit den Diensten des Resource-Layers. Weiterhin gibt das Resource-Layer auch Informationen über den Zustand und der Leistung der einzelnen Ressourcen (z.B. freier Plattenplatz, CPU-Last usw.). Die Dienste des Collective-Layers sorgen dann für die Ressourcen-Findung, die Gruppen-Authorisierung und Systemüberwachung. Auf der Anwendungsebene (Application-Layer) müssen außerdem ein Jobmanagement, Checkpointing und Fortschrittsanzeigen implementiert werden.

[8]Bildberechnung mit genauem (Licht-)Strahlenverlauf

5 Klassifikation

Die Anwendungen des Grids werden in fünf Klassen aufgeteilt:

- **verteiltes Großrechnen (Supercomputing)**
  Hierzu zählen alle Probleme, die sehr groß sind, viel Rechenleistung und Speicher benötigen.

  Beispiele:

  - Distributed Interactive Simulation (DIS) ist eine Anwendung zum Training und zur Planung beim Militär. Realistische Szenarios benötigen über 100000 komplexe Einträge, ein aktueller Supercomputer kann zur Zeit aber nur 20000 Einträge meistern.
  - Berechnung von komplexen physikalischen Prozessen benötigen eine hohe räumliche und zeitliche Auflösung. Um mit feinen Detailsstufen neue wissenschaftliche Erkenntnisse zu gewinnen, werden aber verbundene Supercomputer benötigt. Erfolgreiche Anwendungen sind zum Beispiel im Bereich Klima-Modellbildung und Astronomie zu finden.

- **Rechnen mit hohem Durchsatz**
  Es ist nötig eine hohe Anzahl von lose gekoppelten oder unabhängigen Prozessen möglichst schnell zu berechnen. Dies wird mit Hilfe der Rechenleistung von gerade nicht benutzten Rechnern erreicht (meist Workstations im idle-Modus).

  Beispiele:

  - In der heißen Phase der Chip-Entwicklung des AMD K6 und K7 wurden tausend Rechner dafür verwendet, um den notwendigen hohen Durchsatz zu erreichen. Diese Rechner waren Desktop-PCs der Ingenieure von AMD, verteilt auf einigen Lokationen, und wurden nur verwendet, wenn sie nicht anderweitig benutzt wurden.
  - Das Condor-System von der University of Wisconsin-Madison wird benutzt um Rechenpools von hunderten Desktoprechnern von Universitäten und Laboren auf der ganzen Welt zu verwalten. Diese Ressourcen werden für molekulare Simulationen von Flüssigkeitskristallen, Studien über bodendurchdringenden Radar und Design von Dieselmotoren verwendet.

  - Mehrere Bemühungen gehen in das Lösen von schweren kryptographischen Problemen mit vielen tausenden Computern.

- **Rechnen bei Bedarf (On-Demand Computing)**
  Von den Anwendungsprogrammen werden die Ressourcen immer nur relativ kurz benötigt. Daher ist es auch nicht kosteneffektiv, die entsprechenden Ressourcen selbst lokal zu besitzen. Im Gegensatz zur absoluten Leistung beim verteilten Großrechnen ist hier ein gutes Kosten-Leistungsverhältnis wichtig.

  Beispiele:

  - Eine computererweiterte MRI-Maschine und ein Rastertunnelmikroskop, entwickelt im "National Center for Supercomputing", benutzen Anwendungen zur Supercomputernutzung, um Echtzeit-Bildberechnung zu betreiben.
  - Ein System der "Aerospace Corporation" für Datenberechnung von meterologischen Satelliten benutzt dynamisch zugewiesene Supercomputer, um die Ausgabe eines Wolkenerkennungsmechanismus an Meterologen in quasi Echtzeit zu übermitteln.

- **sehr datenintensive Berechnungen**
  Hier steht die Entstehung von neuen Informationen aus großen Mengen von Daten im Vordergrund. Da diese Daten meist in geografisch verteilten Bibliotheken und Datenbanken enthalten sind, ist auch die Kommunikation sehr hoch.

  Beispiele:

  - In Zukunft werden Hochenergiephysikexperimente Terabytes an Daten am Tag erzeugen (ein Petabyte im Jahr). Die komplexen Anfragen zum Erkennen der "interessanten" Ereignisse benötigen den Zugriff auf große Teile dieser Daten. Die wissenschaftlichen Labore, die auf die Daten zugreifen wollen, sind weit verteilt, genauso wie die Daten sehr verteilt sein werden.
  - Die digitale Himmelserfassung speichert einige Terabytes astronomischer Fotodaten in vielen verteilten Datenbanken.
  - Bei der Wettervorhersage werden einige Gigabytes Satellitendaten ausgewertet.

  - Das EU-DataGrid wurde von der Europäischen Union gegründet, um den Zugriff auf geografisch verteilte Rechenleistungen und Datenspeichern für verschiedene Institutionen zu ermöglichen. Dadurch werden die Ressourcen geschaffen, die das hohen Datenaufkommen der wissenschaftlichen Experimenten verarbeiten zu können. DataGrid-Applikationen finden hauptsächlich in den Bereichen Hochenergiephysik, medizinische und biologische Bildverarbeitung sowie Erdbeobachtungen Anwendung. Dort werden große Datenmengen erzeugt (Bilder, Gencodes, Analysen von Ozonwerten usw.) und müssen verarbeitet und gespeichert werden.

- **gemeinsames Lösen einer Aufgabe (Berechnung)**
  Bei gemeinschaftlichen Forschungen und Entwicklungen ist es sehr wichtig, die Wechselwirkungen zwischen den Forschern zu ermöglichen und zu verbessern. Meist existieren auch gemeinsame (Daten-)Ressourcen wie Datenarchive und Simulationen.

  Beispiel:

  - Das BoilerMaker-System wurde in den "Argonne National Laboratory" entwickelt und erlaubt vielen Nutzern an der Entwicklung eines Schadstoffausstoss-Kontrollsystems für industrielle Verbrennungsöfen mitzuarbeiten. Die Nutzer interagieren miteinander und mit der Simulation des Verbrennungsofens.
  - Das CAVE5D-System ermöglicht entfernte und gemeinschaftliche Erforschung von großen geologischen Daten und generiert Modelle (z.B. ein gekoppeltes physisches und biologisches Modell der Chesapeake Bay).
  - Das NICE-System von der "University of Illinois at Chicago" erlaubt Kindern sich bei der Bildung und Erhaltung von realistischen virtuellen Welten für Unterhaltungen und Ausbildungen zu beteiligen.

Vorläufer der Grid Entwicklung ist zum Beispiel SETI@Home. SETI ist die Abkürzung für "Search for Extraterrestrial Intelligence". Das Projekt fängt Strahlungen aus dem All auf und analysiert sie auf Spuren außerirdischen Lebens. Da dabei eine große Menge an zu berechnenden Daten anfällt, die ein Supercomputer alleine nicht bewältigen würde, bindet SETI@Home Rechner in der ganzen Welt mit ein. Der Client wird auf einen Heimrechner installiert und empfängt immer ein Paket (zum Durchsuchen) nach dem anderen. Der Client berechnet dieses Paket, wenn der Heimrechner gerade nicht

benutzt wird.

Es wird deutlich, dass die Idee des Grids verwendet wird, jedoch nicht die Technologie, sondern eine spezielle Software.

6 Fazit

Das Grid ist gut geeignet für speicher- und rechenintensive Aufgaben, die das Limit eines einzelnen Rechners (oder Großrechners) erreichen oder überschreiten. Zum Beispiel technische und wissenschaftliche Projekte mit großen Datenmengen und/oder komplexen Berechnungen finden Einsatz im Gridcomputing. Die sonst ungenutzten Ressourcen werden vom Grid für seine Aufgaben verwendet.

Die Ressourcen können sehr vielfältig sein, sie reichen vom normalen PC über Computerfarmen, speziellen Sensoren und Datenspeichern bis hin zum Supercomputer. Es wird also keine spezielle Hardware für das Grid benötigt. Aber bei der Verwendung von besonderer Hardware (wie zum Beispiel ein SIMD[9] Rechner) wird diese wirtschaftlicher ausgelastet, da sie nicht nur von einer Abteilung genutzt wird. Gridcomputing eignet sich gut für Institutionen, die geografisch verteilt sind und ein gemeinsames Projekt bearbeiten. Dort ist es dann nötig und wirtschaftlicher bestimmte Ressourcen gemeinsam zu nutzen (z.B. eine große Datenbank).

Für den Normalbenutzer ist das Grid noch nicht verwendbar, da es viel Aufwand und Spezialwissen erfordert. Ein weiterer Faktor ist, dass das Internet noch nicht ausreichend schnell genug ist. Denn die Übertragung der zu berechnenden Daten ins Internet zu den Grid-Rechnern verglichen mit der üblichen Berechnung (zu Hause) lohnt sich noch nicht.

[9]SIMD: Single Instruction Multiple Data

Abbildungsverzeichnis

1 Grid-Protokoll-Architektur 9
2 Das Sanduhrmodell des Internets und des Grids 10

Literatur

[1] Ian Foster, Carl Kesselman, Steven Tuecke: The Anatomy Of The Grid: Enabling Scalable Virtual Organizations (2001)
URL: www.mcs.anl.gov/globus/research/papers/anatomy.pdf

[2] Ian Foster, Carl Kesselman: Computational Grids (1985)
URL: dsl.cs.uchicago.edu/Courses/CS347/../../papers/gridbook_chapter2.pdf

[3] Data Grid: What is a Virtual Organization?
URL: http://web.datagrid.cnr.it/LearnMore/LearnMore3.jsp

[4] What is DataGrid?
URL: http://web.datagrid.cnr.it/LearnMore/LearnMore1.jsp

[5] The challenges of the Grid
URL: http://web.datagrid.cnr.it/LearnMore/LearnMore5.jsp

[6] What are Grids?
URL: http://web.datagrid.cnr.it/LearnMore/LearnMore4.jsp

[7] DataGrid Applications
URL: http://web.datagrid.cnr.it/LearnMore/LearnMore2.jsp

[8] Claudia Leopold: Parallel And Distributed Computing. Wiley-Interscience Publication, 2001

[9] IBM: The Era Of Grid Computing: A New Standard For Successful IT Strategies (2003)
URL: www.ibm.com/grid/pdf/it_exec_brief.pdf

[10] IBM: The Era of Grid Computing: Enabling New Possibilities For Your Business (2003)
URL: www.ibm.com/grid/pdf/business_exec_brief.pdf