



# MAGISTERARBEIT

Titel der Magisterarbeit

„Grid und Cloud Computing“

Verfasserin

Bahareh Shahi Barogh, Bakk.rer.soc.oec.

angestrebter akademischer Grad

Mag.rer.soc.oec.

Wien, April 2010

Studienkennzahl lt. Studienblatt:

A 066 926

Studienrichtung lt. Studienblatt:

Wirtschaftsinformatik

Betreuer:

Ao. Univ. Prof. DI Dr. Siegfried Benkner

### Angaben zur Person

Nachname(n) / Vorname(n) **Shahi Barogh / Bahareh**  
Adresse(n) Freytaggasse 21/23, 1210, Wien  
E-Mail b.shahi@gmail.com  
Staatsangehörigkeit Deutschland  
Geburtsdatum 21.03.1981  
Geschlecht weiblich

### Schulbildung

Bildungseinrichtung Universität Wien  
Zeitraum 01.Oktober 2001 – 10.Februar 2005 Bakkalaureatsstudium Wirtschaftsinformatik  
01.März 2005 – voraussichtlich 30.Juni 2010 Magisterstudium Wirtschaftsinformatik  
Schwerpunkte Software Quality Engineering im Bakkalaureatsstudium  
Grid Computing im Magisterstudium  
Bildungseinrichtung Bundesoberstufenrealgymnasium  
Zeitraum 01.September 2000 – 30.Juni 2001  
Schwerpunkte Mathematik  
Französisch

### Berufliche Tätigkeiten

Firma First Data Austria  
Zeitraum 01.August 2006 – 30.Juni 2009 (Karenzantritt)  
Berufstaetigkeit Key Account Manager  
Schulungen Zertifizierung Presentation Techniques  
Firma Österreichische Volksbanken AG  
Zeitraum 1.Oktober 2005- 30.Juni 2006  
Berufstaetigkeit Homepage-Erstellung  
Firma Blue IT  
Zeitraum 06.Juni 2005 – 30.September 2005  
Berufstaetigkeit Softwaretesterin

### Persönliche Fähigkeiten und Kompetenzen

Muttersprache(n) Farsi, Deutsch, Englisch, Französisch, Spanisch

## Kurzfassung

In dieser Arbeit werden die theoretischen Konzepte und mögliche praktische Anwendungsmöglichkeiten von Grid- und Cloud-Computing dargestellt.

Als erster Teil der Arbeit wird zuerst auf Grids eingegangen. Hierzu wird die Klassifikation von Grids gegliedert nach Distributed Supercomputing, Data-Intensive Computing, On-Demand Computing, High-Troughput Computing bzw. Collaborative Computing erläutert. Anschließend werden im Rahmen einer historischen Betrachtung die Technologien hinter Grid-Computing dargestellt. Hierbei wird sich zeigen, dass mit jeder neuen Generation eine weitergehende Standardisierung erfolgt ist. Innerhalb dieser Abschnitte wird auch die Architektur beschrieben, mit der Darstellung der verschiedenen Schichten, es wird auf das Globus Toolkit eingegangen und es wird beschrieben werden, wie unter dem SOA-Aspekt und Nutzung von WebServices immer stärker Industriestandards anstatt proprietärer Lösungen eingesetzt werden. Dieser Teil wird dem Problem der Sicherstellung eines Quality-of-Service (QoS) abgeschlossen.

Der zweite Teil behandelt Cloud-Computing. Hierbei werden in ähnlicher Weise wie im ersten Teil die theoretischen Konzepte für Clouds erläutert, so erfolgt eine Definition des Begriffs und es werden die Arten von Clouds beschrieben. Dabei wird in einem stärkeren Ausmaß auch auf die wirtschaftlichen Gründe für die Entstehung von Clouds eingegangen. Dies erfolgt sowohl in Hinsicht auf die Anbieter (Cloud-Provider), als auch auf die Anwender von Clouds. Zusätzlich werden auch die Probleme aufgezählt, die einer stärkeren kommerziellen Nutzung von Cloud-Computing im Weg stehen könnten. In eigenen Abschnitten wird auf den Zusammenhang zu Virtualisierung und dem „Everything as a Service“-Konzept eingegangen. In einem weiteren Kapitel dieses Teils werden die kommerziellen Cloud-Umgebungen von Amazon, Google, IBM und Microsoft vorgestellt, hierbei werden Kostenbeispiele gebracht. Anschließend werden mögliche Anwendungen beschrieben, der Teil schließt mit einem Experiment in einer Cloud.

## Abstract

In this work the theoretical concepts and potential practical applications of Grid and Cloud Computing are presented.

The first section of the work focuses on Grids. Here the classification of Grids into Distributed Supercomputing, Data-Intensive Computing, On-Demand Computing, High-Throughput Computing / Collaborative Computing is explained. The technologies behind grid computing are presented in a historical perspective. This will show that with each new generation a further standardization takes place. Within these sections the architecture is also described with the representation of the different layers, the Globus Toolkit is examined and it is described how, by using SOA aspects and Web services, increasingly more industry standards instead of proprietary solutions are being implemented. This part is concluded with an assessment of the problem of ensuring Quality of Service (QoS).

The second section deals with Cloud Computing. These are explained in a similar manner as in the first section - the theoretical concepts for Clouds are explained, thus defining the term, and similarly the types of Clouds are described. To a greater extent, the economic reasons for the formation of Clouds are discussed. This is done both from the view of the provider (Cloud Provider), as well as from that of the users of Clouds. In addition, the problems which might stand in the way of a greater commercial use of Cloud Computing are enumerated. Separate sections address the connection with Virtualization and the "Everything as a Service" concept. Further, the commercial Cloud environments of Amazon, Google, IBM and Microsoft are presented here, as well as examples of the costs associated with them. Finally, possible applications are described, and this section is concluded with an experiment within a Cloud.

# Grid und Cloud Computing

## Inhaltsverzeichnis

<b>INHALTSVERZEICHNIS .....</b>	<b>V</b>
<b>ABBILDUNGSVERZEICHNIS .....</b>	<b>VII</b>
<b>1 EINLEITUNG .....</b>	<b>1</b>
<b>2 GRID COMPUTING .....</b>	<b>2</b>
<b>2.1 Klassifikation .....</b>	<b>5</b>
2.1.1 Distributed Supercomputing .....	6
2.1.2 Data-Intensive Computing .....	6
2.1.3 On-Demand Computing .....	7
2.1.4 High-Throughput Computing .....	7
2.1.5 Collaborative Computing .....	8
<b>2.2 Grid Technologien .....</b>	<b>8</b>
2.2.1 Die erste Generation des Grids .....	8
2.2.2 Die zweite Generation des Grids .....	15
2.2.3 Die dritte Generation des Grids .....	21
<b>2.3 Bestehende Grid Infrastrukturen .....</b>	<b>25</b>
2.3.1 EGEE .....	26
2.3.2 TeraGrid .....	28
2.3.3 VGE .....	30
<b>2.4 Grid Business Models - QoS und SLA .....</b>	<b>37</b>
<b>3 CLOUD COMPUTING .....</b>	<b>41</b>
<b>3.1 Aspekte .....</b>	<b>41</b>
3.1.1 Arten des Cloud Computing .....	43
<b>3.2 Business Context .....</b>	<b>44</b>
3.2.1 Gründe für Cloud Computing .....	44
3.2.2 Vorteile aus Sicht des Cloud Providers .....	45
3.2.3 Vorteile aus Sicht des Cloud Users .....	50
3.2.4 Mögliche Probleme und Lösungsansätze .....	56
<b>3.3 Virtualisierung .....</b>	<b>64</b>
3.3.1 Eigenschaften von virtuellen Maschinen .....	66
3.3.2 Anwendungsbeispiele von virtuellen Maschinen .....	68
3.3.3 „Everything as a Service“-Konzept .....	70

<b>4</b>	<b>KOMMERZIELLE CLOUD UMGEBUNGEN .....</b>	<b>72</b>
<b>4.1</b>	<b>Amazon EC2.....</b>	<b>73</b>
4.1.1	Vorgehensweise für die Nutzung von EC2 .....	76
4.1.2	Bepreisung der Cloud-Nutzung .....	79
<b>4.2</b>	<b>Google App Engine.....</b>	<b>82</b>
4.2.1	Bepreisung der App Engine .....	84
<b>4.3</b>	<b>IBM Blue Cloud .....</b>	<b>85</b>
<b>4.4</b>	<b>Microsoft Azure.....</b>	<b>86</b>
4.4.1	Bepreisung von Windows Azure .....	88
<b>5</b>	<b>ANWENDUNGEN VON CLOUD COMPUTING .....</b>	<b>89</b>
<b>6</b>	<b>EXPERIMENT GOOGLE APP ENGINE.....</b>	<b>94</b>
<b>6.1</b>	<b>Test: Hello World! .....</b>	<b>94</b>
6.1.1	Applikationsentwicklung.....	95
6.1.2	Testen der Applikation .....	97
6.1.3	Deployment der Applikation .....	98
<b>6.2</b>	<b>Produktion: Schachspiel .....</b>	<b>102</b>
6.2.1	Applikationsentwicklung.....	102
6.2.2	Testen der Applikation .....	107
6.2.3	Deployment der Applikation .....	108
<b>6.3</b>	<b>Resultat .....</b>	<b>110</b>
<b>7</b>	<b>ZUSAMMENFASSUNG .....</b>	<b>111</b>
<b>III</b>	<b>LITERATURVERZEICHNIS.....</b>	<b>113</b>

# Abbildungsverzeichnis

- Abbildung 2.1. Power grid
- Abbildung 2.2. Konzeptionelle Architektur des Grids
- Abbildung 2.3. Layer eines Grids
- Abbildung 2.4. Globus Toolkit Version 4 Architektur
- Abbildung 2.5. Virtuelle Organisationen
- Abbildung 2.6. OGSA Architektur
- Abbildung 2.7. VGE System-Architektur
- Abbildung 2.8. VGE Service Infrastruktur
- Abbildung 2.9. VGE Client Infrastruktur
- Abbildung 2.10. VGE Generic Application Service
- Abbildung 3.1. Kapazität für Spitzenzeiten
- Abbildung 3.2. Rechensystem ohne Virtualisierung
- Abbildung 3.3. Rechensystem mit Virtualisierung
- Abbildung 4.1. Simple Storage Service / Elastic Compute Cloud
- Abbildung 4.2. Azure Services Platform
- Abbildung 5.1. Hype Cycle 2009
- Abbildung 6.1. Screenshot: Hello World!
- Abbildung 6.2. Screenshot: Create an Application
- Abbildung 6.3. Screenshot: Login-Maske Schach-Applikation
- Abbildung 6.4. Screenshot: Startseite Schach-Applikation
- Abbildung 6.5. Screenshot: Blitz Lobby

„Ich habe mich bemüht, sämtliche Inhaber der Bildrechte ausfindig zu machen und ihre Zustimmung zur Verwendung der Bilder in dieser Arbeit eingeholt. Sollte dennoch eine Urheberrechtsverletzung bekannt werden, ersuche ich um Meldung bei mir.“

# 1 Einleitung

In dieser Arbeit soll gezeigt werden, wie sich „Cloud Computing“ heute von Endanwendern nutzen lässt. Hierzu werden zuerst die notwendigen Technologien gezeigt und die Verbreitung des Wissens über die Nutzung von Grid und Virtualisierung beschrieben.

Im nächsten Kapitel wird zuerst das Konzept des Grid Computings, unter Beachtung der geschichtlichen Entwicklungen, präsentiert.

Im dritten Kapitel werden die Aspekte des Cloud Computings vorgestellt und gezeigt, auf welche Weise die Virtualisierung von physikalischen Ressourcen eine Voraussetzung für das Cloud Computing darstellt.

Im vierten und fünften Kapitel werden dann einige kommerzielle Cloud Umgebungen vorgestellt, auch um zu zeigen, welche unterschiedliche Ziele die Anbieter dieser Clouds mit deren Bereitstellung verfolgen. Danach werden die Clouds und deren Nutzungsmöglichkeiten aus Sicht der Anwender beschrieben und wie diese Clouds bereits heute nutzen.

Das sechste Kapitel schließt mit einer Demonstration, wie eine Applikation in einer Cloud laufen gelassen wird. Hiermit soll gezeigt werden, wie einfach die Nutzung einer Cloud sein kann – unter der Voraussetzungen, dass einige Kernfunktionalitäten, speziell das Deployment der Applikation, vom Anbieter bereitgestellt werden.

## 2 Grid Computing

Die Anwendung von Computern in Firmen ist inzwischen schon alltäglich, welche Konsequenzen Grids haben können, muss sich noch erweisen.

Die große Vision hinter der Gestaltung von Grid Computing lässt sich als Analogie zur Stromversorgung darstellen. Benutzern soll high performance Rechenleistung so transparent und einfach wie möglich zur Verfügung gestellt werden, so wie dies für Energie mittels der Steckdose erfolgt (power grid [48] (siehe Abbildung 2.1.)). Dadurch ist es für den Strombenutzer im Grunde egal, wie die Produktion des Stroms selbst erfolgt. Auch im Bezug auf die Bereitstellung dieses jederzeit verfügbaren Stroms ist eine Analogie möglich. So wie auch dort die Produktion über Großkraftwerke, genauso wie über viele kleine verteilte Kraftwerke erfolgen kann, gibt es beim Grid Computing die Möglichkeit von Superrechnern im Vergleich zu kleinen haushaltsüblichen PCs [1].

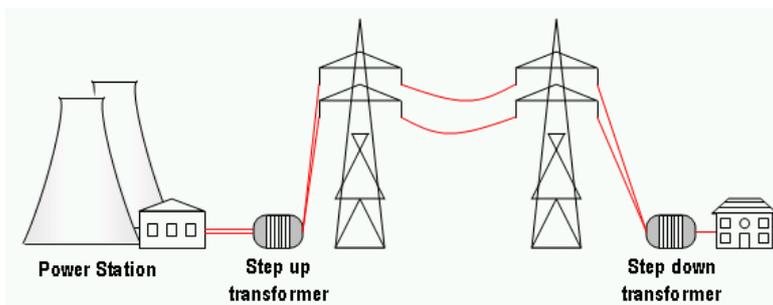


Abbildung 2.1.: Power grid.

Quelle: [48]

Die Umsetzung dieser grundlegenden Idee erfordert die Betrachtung und Lösung technischer, gesellschaftspolitischer und sozialer Aspekte und Probleme. Auf jedem dieser Gebiete wurden und werden viele Forschungen und Entwicklungen angestrengt.

Die erste Definition stammt von Ian Foster und Carl Kesselmann [2]:

*A computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.*

In dieser Definition wird mehr auf die Bereitstellung der Rechenleistung Wert gelegt: wesentlich ist der „zuverlässige, gleichmäßige, überall vorhandene und günstige Zugriff“, so wie dies auch beim Strom möglich ist. Die Produktion hingegen wird eher beiläufig als „hochentwickelte Rechenkapazitäten“ beschrieben wird, womit man dies auch eher als Verwendung von Superrechner sehen kann.

Nach der Entstehung des Grids weiten Ian Foster Carl Kesselmann die Definition aus [5]:

*The sharing that we are concerned with is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled, with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a virtual organization.*

Der wesentliche Unterschied zur ursprünglichen Definition ist der kollaborative Aspekt, der es erlaubt komplexe Aufgaben durch die Zusammenarbeit von vielen Rechnern zu lösen und nicht über einen einzelnen Hochleistungsrechner. Dieses somit im zuvor beschriebenen Vergleich mit der Stromversorgung über Kleinkraftwerke erfolgen würde.

Somit ergibt sich aber auch eine notwendigerweise umfangreichere Definition für das Teilen der Ressourcen. Während man in der ersten Definition von Ian Foster und Carl Kesselmann [2] noch keinen Wert darauf legt (bei Verwendung eines Superrechners als Produktionsanlage ergäbe sich eine n:1-Beziehung). Im Gegensatz dazu wird bei Verwendung von mehr Rechnern auf Produktionsseite (somit einer n:n-Beziehung) die Zuordnung von nachgefragten und angebotenen Ressourcen komplexer (z.B. Fragestellen, wie wer darf wann auf welchen Rechner zugreifen, wie werden Aufgaben verteilt, werden sie auf einen oder mehrere PCs verteilt). Dadurch spielt nun das „Sharing“ in der neuen Definition eine viel größere Rolle als ursprünglich.

Weiters umfasst die neue Definition nicht nur den Datenaustausch und den Rechenleistungszugriff, sondern auch den direkten Zugriff auf Computer, Software, Daten und andere Ressourcen.

Um zu vermeiden, dass Systeme fälschlicherweise als Grid-Systeme bezeichnet werden, hat Ian Foster eine Checkliste für die Eigenschaften eines Grids vorgeschlagen [6]:

1. es werden Ressourcen koordiniert und integriert, die sich nicht unter zentraler Kontrolle befinden
2. es werden offene, standardisierte, allgemeine Protokolle und Interfaces verwendet
3. es werden nicht-triviale Dienste bereitgestellt

Aus einer anderen Blickrichtung definieren Berman et. al. [7]:

*Grid infrastructure will provide us with the ability to dynamically link together resources as an ensemble to support the execution of large-scale, resource-intensive, and distributed applications.*

Der Schwerpunkt liegt hier mehr auf der Ausführung von ressourcen-intensiven und sehr großen verteilten Anwendungen. Aus der Geschichte des Grid Computing geht hervor, dass einige Aufgaben nicht einmal durch Hochleistungsrechnern lösbar waren.

Ein nennenswertes Beispiel für erfolgreiches verteiltes Arbeiten auf *Standard-PCs* ist das SETI@home Projekt [31]. Bei diesem Projekt werden für die Suche nach außerirdischer Intelligenz Radiosignale aus dem Weltall aufgenommen und die Daten nicht wie bei anderen SETI-Projekten über Superrechnern nach Mustern analysiert, sondern es wurde Software entwickelt, damit dies auf Standard-PCs möglich ist.

Ziel des Grid Computing ist es, Benutzern jederzeit den Zugriff auf die Ressourcen die sie benötigen bereit zu stellen. Grid Computing unterscheidet zwischen zwei unterschiedlichen aber verwandten Zielen: Fernzugriff auf einzelne IT-Ressourcen bereitzustellen und Zusammenfassung von Prozessorkapazitäten (d.h. CPU-Leistung mehrerer PCs).

## **2.1 Klassifikation**

Aufgrund der unterschiedlichen Anwendungsgebiete für Grid Computing lässt sich dieses in mehrere Kategorien (Klassen) unterteilen.

Ian Foster und Carl Kesselmann identifizieren folgende fünf grundlegende Klassen von Grid-Anwendungen [9]:

### **2.1.1 Distributed Supercomputing**

Distributed Supercomputing versucht Ressourcen für die Lösung eines Problems zu verwenden, welches auf einem einzelnen System nicht gelöst werden könnte. Hierbei können als Lösung entweder nur die Rechner innerhalb eines Unternehmens oder auch mehrere Supercomputer zusammen verwendet werden. So wie dies zum Beispiel bei der Simulation von Klimamodellen, also von komplexen, physikalischen Prozessen in hoher zeitlicher und räumlicher Auflösung notwendig ist.

### **2.1.2 Data-Intensive Computing**

Das Grid wird benutzt, um neue Informationen aus bereits vorhandenen Daten zu erzeugen. Hier wird auf weltweit verteilte Datenbanken zugegriffen. Die Erzeugung neuer Informationen kann in manchen Fällen sehr rechenaufwendig sein, in anderen Fällen können Sensoren sehr einfach aufgebaut sein, durch eine Beobachtung in sehr kurzen Intervallen werden aber viele Daten gesammelt, deren Analyse dann rechenintensiv wird. In beiden Fällen kann eine Übertragung großer Datenmengen notwendig werden.

So entstehen bei der Durchführung von physikalischen Experimenten wie z.B. am CERN mit dem LHC (Large Hadron Collider) oder bei medizinischen Experimenten, wie der Suche nach neuen Medikamenten in der Krebsforschung riesige Datenmengen, die nicht mehr durch einzelne Computer – auch wenn diese Hochleistungsrechner sind – gespeichert und auch analysiert werden können oder es auch nicht möglich wäre, dies zu zentralisieren. Bei letzterem Aspekt kann man zum Beispiel an verteilte Sensorensysteme denken, wo es mehr Sinn machen könnte die Daten vor Ort zu speichern und zu analysieren und nur die Ergebnisse über eine Datenanbindung zu versenden.

### **2.1.3 On-Demand Computing**

Beim On-Demand Computing werden kurzfristig benötigte Rechen-Ressourcen auf verschiedenste Weisen zur Verfügung gestellt. Dies könnten auch auf Vorrat gekaufte Hardware-Ressourcen sein, bei denen mit dem Verkäufer ein entsprechendes Lizenzmodell ausgehandelt wurde oder die Ressourcen werden über einen Grid genützt. Besonders bekannt ist hier der Slogan „E-Business on Demand“, unter dem IBM seine Dienstleistungen angeboten hat.

Man bezeichnet diese Form auch als „Utility Computing“ und es gibt verschiedene Kostenmodelle, damit Firmen die Ressourcen nachfragen können.

### **2.1.4 High-Throughput Computing**

Das Grid wird benutzt um ein Problem, das aus sehr vielen lose gekoppelten oder voneinander unabhängigen Teilproblemen besteht, zu lösen. Dadurch können mehrere Rechner gleichzeitig arbeiten ohne darauf angewiesen zu sein, auf das Ergebnis anderer Rechner zu warten. So wird sichergestellt, dass der Durchsatz durch den Einsatz vieler unabhängiger Ressourcen erhöht wird. Beispiele hierfür sind Parameterstudien, wie für die Suche nach neuen Medikamenten (so kann die Wechselwirkung von Proteinen für jedes unabhängig getestet werden, wie dies auch im Forschungsprojekten des World Community Grid, z.B. gegen Krebs gemacht wird [49]) oder bei Monte-Carlo-Simulationen in der Finanzwelt, wo ein Algorithmus mit unterschiedlichen Parametern und Zufallsereignissen ausgeführt wird und erst die Ergebnisse in einer Wahrscheinlichkeitsverteilung zusammengefasst werden. Die einzelnen Durchläufe sind voneinander unabhängig, daher kommt es zu keinem Datenaustausch zwischen den Client.

### **2.1.5 Collaborative Computing**

Beim Collaborative Computing liegt der Fokus auf der Kommunikation von Mensch-zu-Mensch, also wie es die deutsche Übersetzung von „collaborative“ ausdrückt, in der gemeinschaftlichen Nutzung von Computerressourcen. Somit können Wissenschaftler, die verstreut über die Erde leben, gemeinsam an einem Forschungsprojekt arbeiten („virtuelles Labor“) und ihre Teilarbeiten in einem Archiv zur Verfügung stellen.

## **2.2 Grid Technologien**

Die ersten Entwürfe für die Verteilung rechenintensiver Aufgaben hat es in den 1960er gegeben. Um mehr Rechenleistung zur Lösung wissenschaftlicher Probleme verwenden zu können, wurden in den 1980er Jahren neue Verfahren und Techniken analysiert. Parallelrechner (d.h. Rechner mit mehreren CPUs) gelangten an die Grenzen ihrer Leistungsfähigkeit und Applikationen forderten immer mehr Rechenleistung und Speicherkapazität. Zu dieser Zeit hatte man die Vorstellung, die eng gekoppelten Parallelrechner zu entkoppeln und verteiltes Supercomputing zu ermöglichen. Die Basis des Grid Computings, nämlich Koordination und Verteilung von Ressourcen, wurden in den 1990er gelegt. In den letzten 25 Jahren hat sich Grid Computing aus dem verteilten und parallelen Computing entwickelt.

### **2.2.1 Die erste Generation des Grids**

Anfang der 90er wurden die ersten Grid Umgebungen geschaffen. Die ersten Grids dienten zum Zusammenschluss mehrere Super-Computing Zentren und wurden als Meta-Computing Infrastrukturen bezeichnet. Unterschiedlichste Projekte mit verschiedenen Hauptmerkmalen und Ausrichtungen wurden zu dieser Idee entwickelt.

Jedoch arbeiteten sie alle an ähnlichen Problemen [10]:

Ressourcen Management, das Arbeiten mit nicht-lokalen Daten, Kommunikation zwischen den Systemen, Effizienz und Effektivität des Grids.

In diesem Zusammenhang sind die zwei wichtigsten Projekte (nach [10]) FAFNER und I-WAY zu nennen.

FAFNER [11] hatte das Ziel, eine relativ einfache, inhärente parallele Anwendung zu verteilen (Faktorisierungsproblem). Ihm kam es hauptsächlich auf die Verwendung sehr vieler Ressourcen an, um die benötigte Rechenkapazität kostengünstig verwenden zu können (d.h. dabei wurden viele low-end Systeme verbunden). Für die Applikation wurde kein so großes Wert auf eine schnelle Verbindung zwischen den einzelnen Systemen gelegt, da diese nicht voneinander abhängig waren. Somit ein Beispiel für high-throughput Computing. FAFNER war der Vorgänger vieler heutigen Systeme wie SETI@home [12] oder Distributed.Net [13].

I-WAY (Information Wide Area Year) [14] hatte anders als FAFNER das Ziel, mehrere Super-Computer über schnelle Verbindungen zu verbinden (distributed super-computing). Ein neuartiges Visualisierungssystem (CAVE [15]) sollte noch zusätzlich entwickelt werden. Die Grundlegenden Ideen für das I-WAY Projekt waren: Supercomputing, Zugriff auf verteilte Ressourcen, Virtuelle Realität und Video. I-WAY hat weitgehend die Entwicklung des Globus-Toolkits unterstützt. (Details siehe Kapitel 2.2.2.1)

### 2.2.1.1 Konzeptionelle Architektur

Nach Foster et. al. kann die Architektur des Grids – analog zu distributed systems – in Schichten dargestellt werden ([22] siehe Abbildung 2.2.). Hierbei kann dies analog zum Internetprotokoll dargestellt werden. Die Schichten der Grid Protocol Architecture entsprechen dabei TCP/IP in folgender Weise:

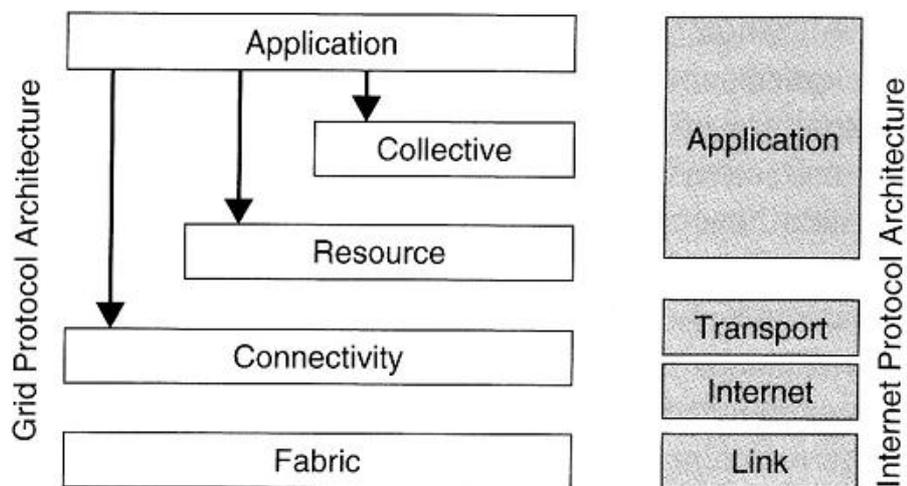


Abbildung 2.2.: Konzeptionelle Architektur des Grids im Vergleich zum TCP/IP Stack.

Quelle: [22, Figure 6.2]

So wie es beim Internet-Protokoll eine Unterteilung zwischen der Applikation und den Verbindungsmechanismen gibt, ist diese Unterscheidung auch in der Grid Protokoll Architektur möglich. Eine weitere direkte Übereinstimmung ist nicht in gleicher Form gegeben, so müssen sowohl im Grid als auch im Internet die Daten aus der Applikationsebene übertragen wird, dies erfolgt über die Verbindungsebenen der beiden Protokolle, es ist hierbei aber der Connectivity-Layer des Grid mehr zu vergleichen mit dem Application-Layer des Internetprotokoll, d.h. die Übertragung von Daten im Grid stellt eine mögliche Anwendung für das Internet-Protokoll dar. Somit kann das Internet diesen Teil der Grid Funktionalitäten übernehmen.

### **2.2.1.1.1 Fabric layer**

Die Hauptressourcen die im Grid gemeinsam verwendet werden sollen, z.B. Rechner, Netzwerk Speicher und Ressourcen, etc. sind im *fabric layer* zu finden. Es ist nicht notwendig dass eine Ressource eine physikalische Einheit ist, es kann auch ein logisches Gebilde sein, dass selbst auf anderen Ressourcen aufbaut. Die Operationen die durch die Ressourcen im *fabric layer* auf höheren Ebenen zur Verfügung gestellt werden, müssen festgelegte Grundanforderungen erfüllen. Die Komplexität des *fabric layer* steigt, wenn mehr Funktionalitäten bereitgestellt werden und dadurch komplexere Funktionen auf höheren Ebenen des *fabric layer* angeboten werden. Je komplexer der *fabric layer* ist, desto höher sind dementsprechend auch die Kosten für die Bereitstellung des Layers.

Die Erfahrung hat gezeigt, dass sich im Grid kaum Ressourcen gleichzeitig verwenden lassen, jedoch lassen sie sich reservieren und können bei Verfügbarkeit genutzt werden. Die Ressourcen müssen zumindest Funktionalität zur Auffindung (*enquiry*) (Abfrage der Eigenschaft) und zum Ressource Management bieten, damit sie intelligent genutzt werden können [22].

### **2.2.1.1.2 Connectivity layer**

Durch den *connectivity layer* werden Protokolle für die grundlegende Kommunikation und Authentifikation für Grid Ressourcen bereitgestellt. Durch die Kommunikationsprotokolle ist es möglich, Daten zwischen Ressourcen auszutauschen. Authentifikationsprotokolle ermöglichen die Identität von Ressourcen und Benutzern zu verifizieren.

Da sich auch in internen Netzwerken TCP/IP bereits als Standard für derartige Aufgaben etabliert hat, wird dieses auch für den *connectivity layer* der heute eingesetzten Grids verwendet.

Dies bezieht sich sowohl auf die low-level Protokolle als auch die Anwendungsprotokolle. Ebenso natürlich für die sicherheitsbezogenen Aspekte, wo somit auch standardisierte Protokolle eingesetzt werden können (derzeit vor allem SSL/TSL und X.509 Zertifikate) und keine eigenen Lösungen nur für Grids entwickelt werden müssen.

### **2.2.1.1.3 Exkurs: Sicherheit im Connectivity-Layer und in der Anwendungsebene**

#### **Single sign-on**

Der Benutzer authentifiziert sich nur einmal, und die Registrierung soll für alle Ressourcen im Grid gültig sein.

#### **Delegation**

Damit ist die Weitergabe von Security-Credentials beim Ausführen der Programme, die durch einen Benutzer angeordnet wurden, gemeint. Dadurch können die existierenden Rechte auch an andere Programme abgegeben werden, die somit die gleichen Rechte wie das eigentlich gestartete Hauptprogramm haben.

#### **Integration mit existierenden Lösungen**

Zur lokalen Authentifikation werden zurzeit vor allem Unix-basierte Lösungen und Kerberos (Active Directory in Windows) verwendet. Grid Lösungen sollen sich in diesen Lösungen integrieren, alle Grid Ressourcen sollen genutzt werden können sobald sich der Benutzer lokal authentifizieren.

#### **Anwender basiertes Vertrauen**

Gegenseitige Vertrauensmitteilung der einzelnen Systeme untereinander, d.h. ein Benutzer der Rechte auf zwei verschiedene Systeme hat, sollte Programme auf beiden Systemen miteinander ausführen können.

#### **2.2.1.1.4 Resource layer**

Der *resource layer* hat den Kommunikations- und Authentifikationsprotokolle des *connectivity layers* als Grundlage. Diese Ebene beschreibt Protokolle für sichere Initialisierung, Verhandlungen und Überwachung, Zahlung und Abrechnung für individuelle Ressourcen [22]. Es können zwei Formen von Protokollen differenziert werden:

##### **Informationsprotokoll**

Mit diesem Protokoll kann man Informationen über die Ressourcen abfragen.

##### **Managementprotokoll**

Dieses Protokoll dient zur Ausführung der Dienste und deren Überwachung. Zusätzlich dienen diese Protokolle für Verhandlungen über die Ressource. Dabei kann über die Reservierungen von Ressourcen verhandelt werden (sowie über Anforderungen oder Quality of Service).

Für jeden Service im Grid sind *connectivity* und *resource layer* wesentlich und sollten die Funktionalität des *fabric layers* umfassen.

#### **2.2.1.1.5 Collective layer**

Für die Interaktion von mehreren Services oder auch für Gruppen von Ressourcen dient der *collective layer*. Der *resource layer* fungiert jedoch nur zur Interaktion mit einem Service.

Auch wenn die Ressource keine zusätzliche Funktionalität anbietet, kann der *collective layer* eine Reihe von Funktionen darbieten.

Einige Beispiele für solchen Funktionen wären [22]:

- Brokering und Scheduling Dienste, um zahlreiche Ressourcen zur selben Zeit zu reservieren.
- Diagnostik- und Überwachungsfunktionen, um die Überlastung, Fehler, etc. zu identifizieren.
- Replikationsdienste, um Zuverlässigkeit und Performance zu steigern.
- Verzeichnisse, um Ressourcen und deren Eigenschaften zu ermitteln. Auf diese Weise ist es möglich nach Ressourcen mit bestimmten Eigenschaften zu suchen.

Dienste, die in jedem Grid notwendig sind sowie Dienste, die nur innerhalb einer virtuellen Organisation relevant sind, können vom *collective layer* bereitgestellt werden. Die Implementation findet als selbstständiger Dienst oder als Applikationsbereich im Grid statt.

#### **2.2.1.1.6 Application layer**

Die Applikationen in der obersten Schicht der Architektur bestehen aus einzelnen Services bzw. rufen die einzelnen Services auf. APIs werden von jeder Schicht bereitgestellt, diese können wiederum von übergeordneten Schichten und der Applikation aufgerufen werden.

In Abbildung 2.3. werden die APIs eines Grids mit den jeweiligen Schichten gemeinsam mit den Protokollen und ihren Interaktionen dargestellt.

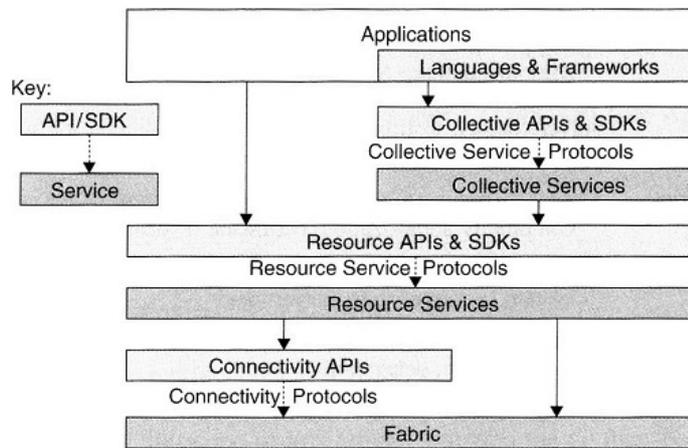


Abbildung 2.3.: Layer eines Grids mit den jeweiligen APIs und Protokollen.

Quelle: [22, Figure 6.4]

## 2.2.2 Die zweite Generation des Grids

Anders als in der ersten Generation des Grids, hat es die zweite Generation des Grids als Ziel nicht nur mehr einzelne Supercomputer zu verbinden sondern eine Infrastruktur zu bieten, um verteilte Applikationen weltweit verteilen und ausführen zu können. Jedoch mussten zunächst folgende drei Anforderungen erforscht werden [10]: Heterogenität, Skalierbarkeit und Anpassungsfähigkeit.

Heterogenität setzt sich mit dem Aspekt der unterschiedlichen Ressourcen auseinander, die nicht zentral administriert werden können.

Skalierbarkeit befasst sich mit der Anzahl der Ressourcen (einige wenige bis mehrere Millionen) und der damit verbundenen Komplexität.

Anpassungsfähigkeit ist hinsichtlich der dynamischen Struktur des Grids und der Verwaltung der Ressourcen erforderlich.

Zusätzlich wurden Middleware-Systeme entwickelt, um die Problembereiche Heterogenität, Skalierbarkeit und Anpassungsfähigkeit zu lösen. Um die Komplexität bei der Vernetzung der Systeme zu vermindern und die Heterogenität vor Applikationen und Benutzern zu verstecken, setzen die Middleware-Systeme vermehrt auf Standards. Somit wird eine homogene Umgebung kreiert, welche den auf der Infrastruktur aufbauenden Applikationen eine standardisierte Schnittstelle bereitstellt.

### **Kerntechnologien der zweiten Generation**

Die wesentlichsten Projekte, die sich mit Infrastruktur, den wichtigsten Services, Zusammenarbeit, etc. befassen sind Globus und Legion. Globus wurde um hohen Maße vom I-WAY Projekt unterstützt. (Details siehe Kapitel 2.2.2.1).

Legion [16] wird als ein objekt-basiertes 'Metasystem' [10] bezeichnet. Es bietet eine Infrastruktur an, um verteilte, heterogene, high-performance Rechner nach verschiedenen Kriterien zu klassifizieren, damit eine Unterscheidung für den Zugriff möglich wird. Für alle Systembenutzern soll eine äquivalente Infrastruktur sichtbar sein, unabhängig von physikalischen Systemen. Dabei werden alle Systemkomponenten als Objekte angeboten und bieten wiederum objektorientierte Eigenschaften an.

#### **2.2.2.1 Globus Toolkit**

Das Fokus des Globus Projekts [24] lag darin, eine Rahmenbedingung für die Infrastruktur zu bilden, um komplexe Dienste im Grid zu ermöglichen. Globus wurde auf Basis des I-WAY Projekts [14] in der zweiten Generation des Grid entwickelt.

Das Projekt hatte als Ziel die Entwicklung von low-level Komponenten und Tools, um damit die Realisierung von komplexen Grid-Services zu ermöglichen, so wie auch von Techniken, um die Ausführung der darauf laufenden Services zu überwachen [24].

Das Globus Toolkit [38] umfasste in den ersten Versionen eigenständige Tools bzw. Module, die erkennbare Eigenschaften für Grid-Services bereitstellen. Folgende sechs Module wurden laut Ian Foster et. al. [24] als essentiell identifiziert:

**Authentication interface** erlaubt die Identifikation der User und Ressourcen. Dies können verschiedenste weitergehende Sicherheitsmechanismen sein und je nach Bedarf einen schwächeren, aber performanteren oder einen stärkeren Schutz bieten.

**Unified resource information service** soll das Abfragen der Informationen über den Zustand und die Struktur einer Ressource ermöglichen.

**Process creation** ermöglicht die Ausführung von Prozessen, nach dem Auffinden der Ressourcen. Dementsprechend soll es möglich sein Ausführungsumgebungen mit den ausführbaren Dateien zu generieren und die Parameter zu übergeben. Dabei soll der Prozess weiterhin überwachbar sein und beendet werden können.

**Communications** legt essentielle Kommunikationseigenschaften bereit. Hierbei werden unterschiedliche Kommunikationstypen (wie remote procedure call, stream-based, message passing, multicast und shared memory [24]) zugelassen.

**Data access** ermöglicht den Zugriff auf entfernte Datenbanken, Dateien, Daten, etc. Darüber hinaus können für den Zugriff verteilte Datenbanken, CORBA oder dergleichen verwendet werden.

**Resource location and allocation** ermöglicht eine Ressource im Grid zu finden, sie zu reservieren und die Beschreibung der Anforderung an sie. Um mit den Ressourcen im Grid arbeiten zu können müssen sie zunächst dynamisch gefunden und reserviert werden.

Die neueste Version 4 des Globus Toolkits [39] ist im Gegensatz zu den Vorgängern als Web Service realisiert worden. Für die meisten Globus Toolkit Dienste werden Web Services und Web Service Clients verwendet, diese wiederum benutzen SOAP für die Kommunikation. Jedoch bauen noch einige legacy Dienste (wie z.B. GridFTP) auf andere Protokolle (wie in Abbildung 2.4. dargestellt [39]. Daher folgt Globus nur zum Teil der Open Grid Service Architecture (OGSA)-Spezifikation. Diese wird in Kapitel 2.2.3.1 näher beschrieben.

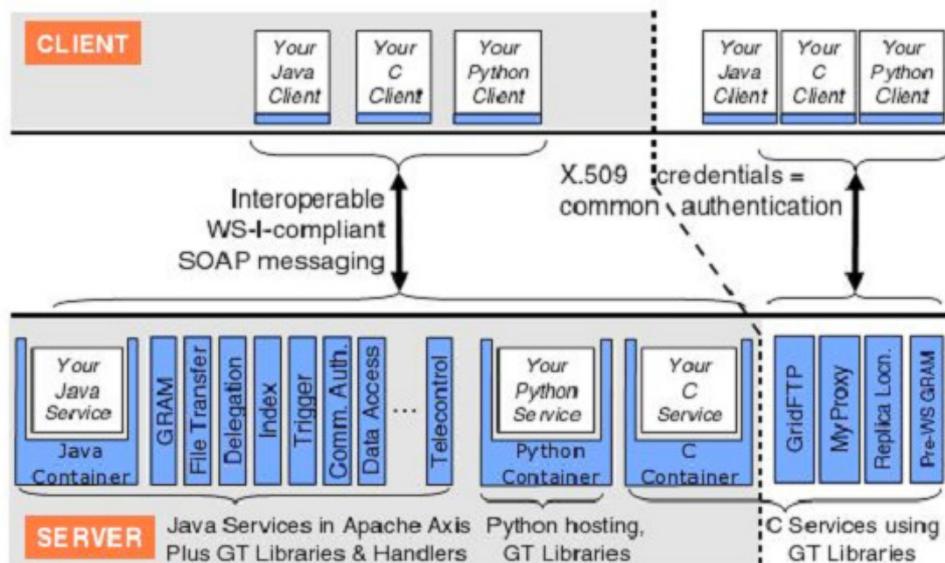


Abbildung 2.4.: Globus Toolkit Version 4 Architektur.

Quelle: [39, Figure 1]

## **Peer-to-Peer Systeme**

Peer-to-Peer (P2P [23]) Connection bezeichnet eine Rechner-Rechner-Verbindung, eine Verbindung unter Gleichen. In P2P Systemen sind alle Computer gleichberechtigt, es gibt keinen zentralen Server mehr. Das Problem der Skalierbarkeit wird anhand von P2P Systemen gelöst. Die Systeme der ersten Generation bezeichnet man als zentralisierte und reine P2P-Systeme, sie dienen vor allem als einfache Möglichkeit des Datenaustauschs (z.B. Napster, Gnutella).

Dezentrale Systeme bezeichnet man als Systeme der zweiten Generation (z.B. RetroShare) und als dritte Generation werden Systeme beschrieben, die Dateien über nicht-direkte Verbindungen weiterreichen [23].

Somit können mittels neuer P2P Systeme freie Speicherplätze und Rechenzeiten im Netz besser sicher gestellt werden. Beim Finden vorhandener Ressourcen und vor allem bei der Sicherheit sind bisher Probleme in P2P-Systemen aufgetreten.

Durch die Verwendung einer P2P-Grid-Architektur können Probleme bei der Zuteilung von Ressourcen behandelt werden, die entstehen würden wenn diese stattdessen über einen zentralen Rechner erfolgen müsste. Denkbar sind hier z.B. Fragestellungen, wie die Handhabung, wenn ein PC im Grid auf Rechenergebnisse eines anderen Rechners warten muss.

## **Ressource Broker und Scheduler**

Um die Verwendung von Ressourcen zu ermöglichen, gibt es Ressource Broker und Scheduler. Hierbei dient der Scheduler dazu, die Durchführung von Jobs zu gewährleisten, während man den Broker mehr als eine Art „Verhandler“ ansehen kann, der herausfindet auf welchen Ressourcen ein Job laufen könnte.

Damit ist es möglich auf die Ressourcen zuzugreifen, den Zugriff zu organisieren und diese zu reservieren. In der vorgemerkten Zeit steht die Ressource dem Benutzer ausschließlich zur Verfügung und zusätzlich können Anfragen priorisiert werden. Falls das Grid eine 'homogene' Verwaltung besitzt, ist der Einsatz von Scheduler vorteilhaft um eine Koordination zu gewährleisten. Dies unterscheidet sich zum SETI@home Szenario, in dem Ressourcen völlig eigenständig organisiert werden, somit aber auch eine Planung von Ressourcen nicht möglich ist.

### **Verteilte Objekte**

Mittels „*distributed object systems*“ können Objekte in verteilten Systemen benutzt werden. Diese Systeme stellen eine Infrastruktur zur Verfügung, um Objekte zu erstellen, verteilen und zu aktivieren. Wesentliche Systeme in diesem Bereich sind CORBA (Common Object Request Broker Architecture [17]) und Java's Jini/RMI [18]. Die Java-basierte Technologie Jini wird eingesetzt, um verteiltes Computing zu vereinfachen und wurde für die Kommunikation zwischen Diensten wie Software oder Hardware entwickelt. Mit der Unterstützung des Jini Protokolls können Applikationen einem Netzwerk virtuell beitreten und miteinander kommunizieren und über das Netzwerk mittel RMI (Remote Method Invocation) Ressourcen nutzen. Somit kann eine Grid-Applikation derart entwickelt wird, dass die zu nutzenden Ressourcen als verteiltes Objekt vom Anbieter bereitgestellt werden und von der Applikation (und damit dem Nutzer des Grids) über RMI angesprochen werden.

### **2.2.3 Die dritte Generation des Grids**

Von der zweiten Generation des Grids wurde die erforderliche Infrastruktur bereit gestellt. Das Ziel der dritten Generation des Grids ist es bereits vorhandene Komponenten und Ressourcen wieder benutzen zu können und diese flexibel miteinander zu kombinieren. Es werden verstärkt Service Orientierte Architekturen (SOA [19]) benutzt und zusätzliche Metadaten verwendet.

In der ersten und zweiten Generation des Grids liegt der Schwerpunkt weitgehend auf distributed systems (verteilte Systeme) und distributed super-computing (verteilt Supercomputing). Der Fokus ändert sich in der dritten Generation und liegt nicht mehr so sehr auf der Technologie des Grids selbst – es wird nun versucht die Umsetzung nicht mittels proprietärer Entwicklung umzusetzen, sondern Industriestandards wie Web Services zu verwenden, um damit den SOA-Aspekt sicherzustellen. Stattdessen liegt nun der Augenmerk mehr auf den Applikationen die darauf aufbauen. Begriffe wie distributed collaboration (verteilte Zusammenarbeit) und virtual organization (Virtuelle Organisationen) werden benutzt [10]. Damit können wissenschaftliche Projekte anhand von Grid Ressourcen gelöst werden.

Ein weiteres Merkmal der dritten Generation ist eine noch stärkere Automatisierung. Dies ist natürlich auch für die einzelne PCs und Server, die nicht in einem Grid eingebunden sind, relevant. Hierzu seien auf die Konzepte des Autonomic Computing von IBM zur Selbst-Konfiguration, -Optimierung und – Heilung verwiesen [50].

Speziell für die dritte Generation des Grids ist die OGSA (Open Grid Services Architecture [26]) Spezifikation. (Details siehe Kapitel 2.2.3.1).

## Virtuelle Organisation

Eine *virtuelle Organisation (VO)* ist eine Form der Organisation, bei der sich eine Gruppe von Unternehmen oder Einzelpersonen virtuell zu einem gemeinsamen Geschäftsverbund zusammenschließen und sich auf bestimmte Regeln über die gemeinsame Nutzung von Ressourcen einigen. VOs werden je nach Projekt oder Kundenauftrag für einen gewissen Zeitraum zusammengefasst [21].

Der Schwerpunkt ändert sich im modernen Grid, so sehen Foster et. al. jetzt den Kernpunkt des Grids in „*coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*“ [22]. Dabei bezieht sich die gemeinsame Nutzung von Ressourcen auf den direkten Zugriff auf Software, Daten, Computer und andere Ressourcen. Hier ist es von Bedeutung den Überblick zu behalten, wer Ressourcen anbietet, wer sie verwenden darf und welche Bedingungen dabei gelten (*Authorising*). Ein Unternehmen kann dabei Mitglied in mehreren virtuellen Organisationen sein.

In der Abbildung 2.5. sieht man drei virtuelle Organisationen und zwei Ressourcen-Cluster. Die realen Unternehmen der VOs (A bis J) können sich dynamisch gruppieren, um auf verschiedene Rechenressourcen und Daten zu zugreifen (z.B. Datenbanken). Dabei gelten für Teilnehmer der verschiedenen VOs unterschiedliche Regeln.

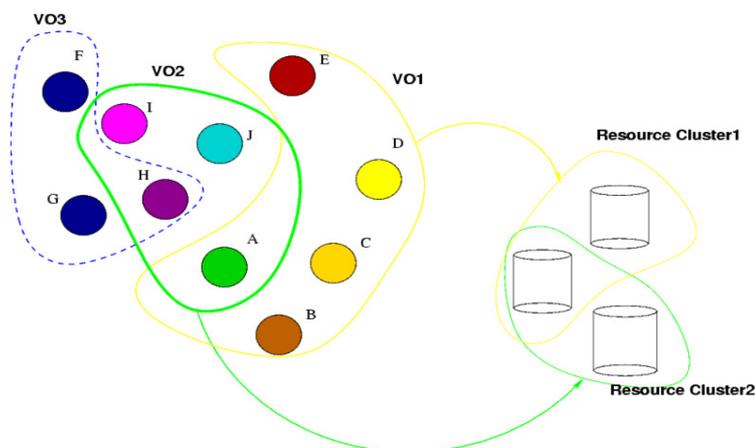


Abbildung 2.5.: Virtuelle Organisationen.

Quelle: [21]

### 2.2.3.1 Open Grid Service Architecture

Open Grid Service Architecture (OGSA) [25, 26] ist eine wichtige Spezifikation der dritten Generation des Grids und basiert auf Web Services Technologien (SOA) und Konzepten. OGSA beschreibt die Architektur für eine Service-orientierte Grid Computing Umgebung.

Das Hauptaugenmerk dieses Frameworks liegt nicht auf den Ressourcen, sondern immer auf den angebotenen Services. Auf diese Weise werden Netzwerk-, Speicher- und Rechenressourcen, Datenbanken, Programme, etc. alle als Service betrachtet.

Nach Ian Foster et. al. werden in OGSA sieben Klassen/Kategorien von Services beschrieben [26]:

**Infrastructure Services** ist die Basis für jedes Grid Service, hier werden die Protokolle, die von allen Services verwendet werden müssen, definiert. Weiters wird festgelegt welche essentiellen Eigenschaften jedes Service haben muss.

Die Open Grid Services Infrastructure (OGSI) war zur Bereitstellung dieses Infrastructure Layers angedacht.

**Execution Management Services** erlauben das Management, die Realisierung und die Ausführung von 'units of work'. 'Units of work' können entweder eine *Legacy*- oder eine OGSA-Applikation enthalten.

**Data Services** sind für das Datenmanagement, den Datenzugriff, Datenveränderungen und Datentransfers innerhalb der Ressourcen zuständig.

**Resource Management Services** führen verschiedene Formen des Managements (z.B. das Einrichten, Reservieren, Virtualisieren, Überwachung, Scheduling, etc.) an Ressourcen im Grid aus.

**Security Services** ermöglichen die Durchführung und Einhaltung von sicherheitsbezogenen Policies innerhalb virtueller Organisation.

**Self-Management Services** erlauben im Sinne des Autonomic Computings eine Selbst-Optimierung und Selbst-Konfiguration der Systemkomponenten mit Hilfe von Policies.

**Information Services** ermöglichen Ressourcen bzw. Metadaten eines Services (z.B. Daten für die Überwachung eines Service) zu verändern oder auf sie zuzugreifen.

Auf der nachfolgenden Seite werden die einzelnen Services in Abbildung 2.6. symbolisch dargestellt.

Einige Services die eine bestimmte Anforderung gemeinsam lösen oder die Mitglied einer virtuellen Organisation sind, werden als Einheit in einer 'virtuellen Domain' zusammengefasst.

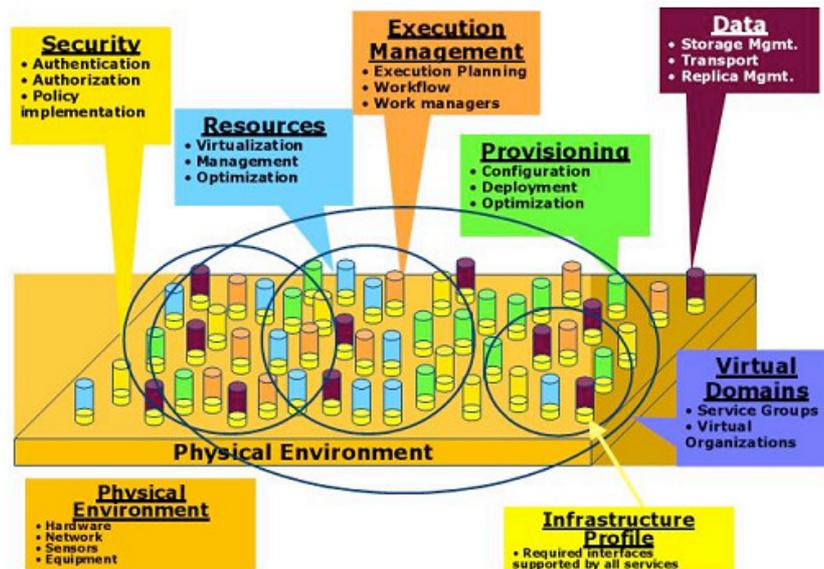


Abbildung 2.6.: OGSA Architektur.

Quelle: [26, Figure 2]

## 2.3 Bestehende Grid Infrastrukturen

Im folgenden Abschnitt sollen nun beispielhaft verschiedene Grid-Infrastrukturen präsentiert werden, die in den letzten Jahren die Vorteile für Grid Computing gezeigt haben, und wie damit Speicher, Rechenressourcen über verschiedenen Organisationen und diese verschiedenen Forschungskreisen mittels klar abgegrenzte Protokollen und Interfaces bereitgestellt werden können.

Eine Reihe von Grid Infrastrukturen, wie EGEE [37] und DEISA [40] in Europa, OSG [41] und TeraGrid [35] in Amerika, APAC [42] und NAREG [43] in Asien-Pazifik-Raum, wurden in den vergangenen Jahren auf dieser Basis gegründet [20].

### **2.3.1 EGEE**

Das EGEE (Enabling Grid for E-sciencE) Projekt führt Experten von mehr als 50 Ländern zusammen, mit dem gemeinsamen Ziel, neueste Forschungsergebnisse im Bereich Grid-Technologien erfolgreich für die Entwicklung einer Dienstleistungs-Grid-Infrastruktur zu verwenden, welche Wissenschaftlern 24 Stunden am Tag, sieben Tage die Woche zur Verfügung steht [37].

Als größtes ursprünglich auf Europa beschränktes Grid-Projekt, hat sich EGEE bereits bis nach Amerika und den Asien-Pazifischen Raum ausgedehnt und soll zukünftig in Richtung einer weltweiten e-Infrastruktur weiterentwickelt werden. Das Projekt ist ein Multi-Phasen Programm und wurde 2004 gestartet und wird voraussichtlich 2010 enden [37].

Die EGEE Infrastruktur wird derzeit von mehr als 5.000 Usern benutzt, die etwa 200 VO bilden, und täglich mehr als 140.000 Rechenaufträge ausführt. EGEE vereinigt derzeit ungefähr 250 Rechenzentren aus über 48 Ländern und bietet mehr als 50.000 CPUs und einigen PetaBytes an Speicher [37].

Unter allen Grid-Projekten, vereinigt das EGEE Projekt thematische, nationale und regionale Grid Initiative, um so allen Wissenschaftlern in Europa eine e-Infrastruktur zur Verfügung zu stellen und somit den Europäischen Wissenschaftsbereich zu unterstützen. Das Projekt ermöglicht Wissenschaftlern in Industrie und Forschung den Zugang zu umfangreichen Rechnerkapazitäten, die unabhängig vom geographischen Einsatzort genutzt werden können.

Die EGEE Benutzer sind Wissenschaftler aus verschiedenen Fachbereichen, wie Archäologie, Astronomie, Astrophysik, Computer Chemie, Geowissenschaft, Finanzwesen, Geophysik, Hochenergiephysik, Biowissenschaft, Materialkunde, uvm.

EGEE arbeitet außerdem mit kommerziellen Unternehmen, speziell solchen aus der Industrie und mit Internet Providern bzw. Rechenzentren zusammen, um so die Technologie abzusichern, indem ein ‚*transfer to business*‘, also ein Wissenstransfer zu Unternehmen, stattfindet [20].

Das EGEE Projekt konzentriert sich darauf, eine Anzahl von neuen Grid-Usern zu gewinnen. Folgende drei Kernpunkte umfasst das Projekt [37]:

1. Ein konsistentes, robustes und sicheres Grid-Netzwerk aufzubauen, um so zusätzliche Rechenleistung zu gewinnen,
2. Wartung und fortlaufende Optimierung der technischen Infrastruktur (z.B. gLite als Middleware), um dem User betriebssichere Dienste bereitzustellen,
3. Gewinnung neuer User aus Industrie und Forschung sowie Bereitstellung von Support und Schulungen auf einem hohen Standard.

Die EGEE Grid Infrastruktur besteht aus einer Gruppierung von Middleware Diensten, verteilt auf eine globale Ansammlung von Rechner- und Speicher-Ressourcen, sowie die dazu benötigten Dienste und Strukturen, um diese zu betreiben [37].

- Die **Production Service Infrastruktur** ist eine große Multi-wissenschaftliche Grid Infrastruktur, die etwa 250 weltweit verteilte Ressourcen Zentrum, mit etwa 40.000 CPUs und einige Petabytes von Speicher, zusammenbringt. Diese Teil ist für den Produktiveinsatz von Anwendungen vorgesehen und ist in seinem Aufbau stabil und gut unterstützt. Hierfür wird die aktuellste Version der gLite [42] Middleware benützt.

- Das ***Pre-Production Service (PPS)*** bietet Zugang zur Infrastruktur für Testzwecke. Dieser Teil dient für interessierte Benutzer zum Testen und Evaluieren bzw. um Feedback über die Änderungen und neuen Funktionalitäten der Middleware zu geben. Weiters, erweitert die Pre-Produktion die Middleware Zertifizierungsvorgang, in dem das Ausrollen, Interoperabilität und Grundfunktionalität unter realen Bedingungen geprüft werden kann.
- Das ***EGEE Network Operations Centre (ENOC)*** kümmert sich um die Koordination zwischen EGEE und den Netzwerkanbietern (GEANT2/NRENs).

Dies ist ergänzt durch die Schulungsinfrastruktur und die Zertifizierungsinstanzen, sowie die benötigte Supportinfrastrukturen und Richtliniengruppen.

### **2.3.2 TeraGrid**

Das TeraGrid ist eine „*open scientific research*“ Infrastruktur und verbindet große Rechenressourcen (wie Supercomputer, Speicher und wissenschaftliche Visualisierungssysteme) auf die Unternehmensstandorte ihrer elf „*Resource Provider*“ Partner, um eine integrierte und persistente Rechenressource zu erzeugen. Durch TeraGrid wird eine dauerhafte große Rechenleistung für wissenschaftliche Berechnungen zur Verfügung gestellt. TeraGrid ist die weltgrößte, umfangreichste und verteilte „*open scientific research*“ Infrastruktur des Internets [35].

TeraGrid wird durch die Grid Infrastructure Group (GIG) der Universität von Chicago mit der Zusammenarbeit der „*Resource Provider*“ koordiniert. Zugang auf TeraGrid kann jeder amerikanische Wissenschaftler nach einer wissenschaftlichen Überprüfung kostenlos erhalten [36].

### **TeraGrid Architektur**

Die TeraGrid Ressourcen werden durch eine Service Orientierte Architektur miteinander integriert. Jede dieser Ressourcen bietet ein Service an, das durch die Schnittstellen und Abläufe definiert wird und die eine Sammlung von Software-Applikationen ausführen, die sogenannten „*Coordinated TeraGrid Software and Services*“ (CTSS).

CTSS bietet hierbei eine standardisierte Benutzerumgebung gegenüber allen TeraGrid Systemen an. Das erleichtert die Portierung von Programmen von einem System zum nächsten. CTSS bietet zusätzlich auch übergreifende Funktionen an, wie Single-Sign-On, Steuerung über Fernzugriff, Workflow Unterstützung, Datenmigrations-Werkzeuge usw. CTSS umfasst auch das Globus Toolkit, Condor, Abrechnung der Nutzung von verteilten Applikationen und Software zur Verwaltung, Verifizierung und Validierung von Benutzern. Ebenso wird eine Compilersammlung, Programmierer-Werkzeuge und Systemkomponenten zur Verfügung gestellt.

TeraGrid Ressourcen sind mittels einer dedizierten optischen Netzwerks verbunden, wobei jedes Rechnerzentrum mit 10 Gigabits per second angebunden ist. TeraGrid Benutzer haben Zugang durch nationale Forschungsinstitute, den Backbone von Abilene oder National LambdaRail [36].

### 2.3.3 VGE

VGE (*Vienna Grid Environment*) [27, 28] ist eine Web-Services basierte Grid Umgebung. Diese an der Universität Wien entwickelten Umgebung erlaubt es, HPC (High-Performance Computing) Applikationen als Grid Services bereit zu stellen. Durch die Verwendung eines high-level APIs wird die Herstellung komplexer Clients vereinfacht [28]. VGE wurde im Projekt GEMSS (*Grid-Enabled Medical Simulation Services* [29]) als Grundlage genutzt. (Details siehe Kapitel 2.3.3.1).

#### VGE Architektur

In Abbildung 2.7. ist die Service-orientierte System Architektur der VGE Umgebung dargestellt.

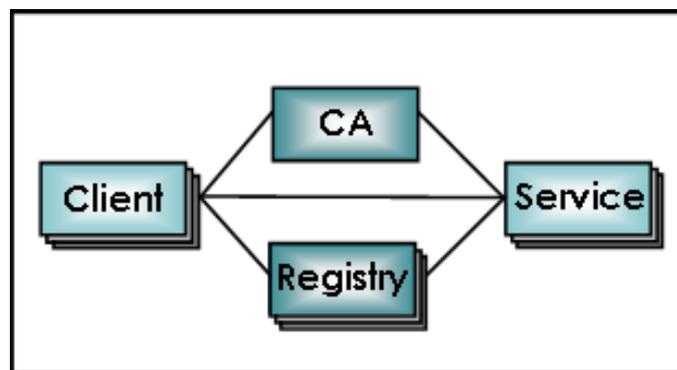


Abbildung 2.7.: VGE System-Architektur.

Quelle: [27]

In Grid Umgebungen können ein oder mehrere optionale Registries verwendet werden, um ein Service zu suchen und es können beliebig viele Services und Clients untereinander interagieren. Für die Interaktion verwendet VGE einen einseitigen Ansatz, der nur vom Client zum Service verläuft, nur dieser kann und muss auf den Service zugreifen. Ein Zugriff in umgekehrter Richtung ist nicht möglich. VGE Client APIs werden von Client Applikationen verwendet, um auf die Middleware zuzugreifen. Dabei muss die Client-Applikation die Input Daten für den Service zur Verfügung stellen, die Ausführung starten und die Output Daten des Service weiterverwerten. Eine Zertifizierungsstelle (certificate authority, CA) ist notwendig, um verschlüsselte und signierte Kommunikation zu erlauben. Demzufolge können geeignete Client- und Server-Zertifikate ausgestellt werden. VGE Services bieten Verfahren an, um Fehler zu bearbeiten (Recovery Service), Befehle auszuführen und zu überwachen (Job Handling) und Quality of Service Parameter festzulegen [28].

### **VGE Service Zugriff**

Die VGE Umgebung ermöglicht einen dynamischen, mehrphasigen Zugriff auf einen Service [28]. Sobald mittels Registry ein Service dem Client zugewiesen wurde, folgen die *Business-Phase*, *Quality of Service-Verhandlungsphase* und *Ausführungsphase*.

Ein Preismodell für den Service kann in der business phase ausgesucht werden (z.B. pay per use) und in der QoS Verhandlungsphase werden mittels dem Client die QoS Parameter (wie z.B. der Preis oder die Ausführungszeit) mit dem Service ausgehandelt. In der Anwendungsphase sendet der Client zunächst die Eingabedaten an den Service, anschließend initiiert er die Ausführung der Applikationen am Service und nach Beendigung der Ausführung ruft er die Ergebnisdaten ab.

## VGE Service Infrastruktur

Die VGE Umgebung ist im Stande Applikationen als Web Services anzubieten. Somit können verschiedene Applikationen, speziell native HPC-Applikationen, über Web Services genutzt werden. Damit die QoS sichergestellt ist, bietet die VGE hierfür ein dynamisches Verhandlungsmodell an, sowohl aufgrund der Infrastruktur, zusätzlich wird aber auch die QoS auf Applikationsebene sichergestellt. Genauso gibt es Methoden zur Reservierung von Ressourcen und auch Bepreisungsmodelle für die Nutzung von Applikationen. Die wesentlichsten VGE Service-Komponenten sind in der Abbildung 2.8. dargestellt.

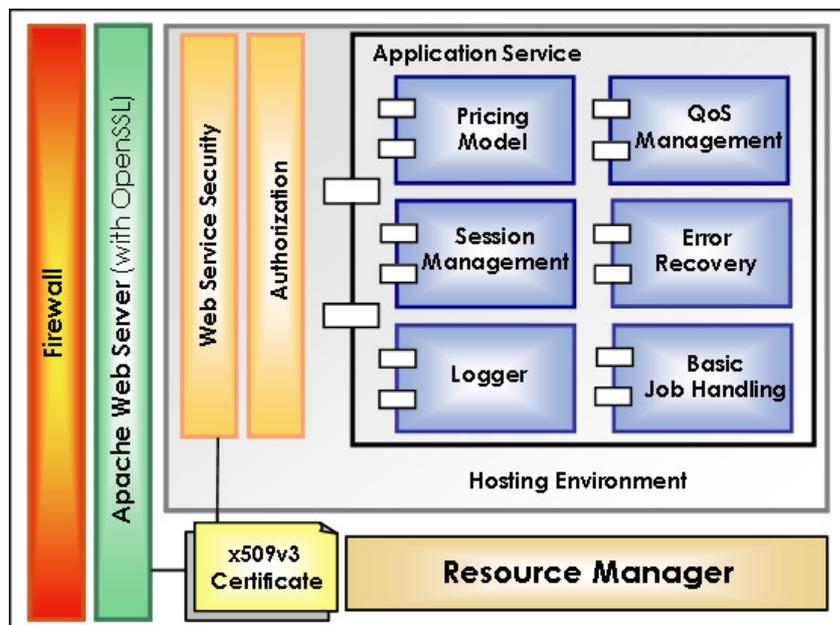


Abbildung 2.8.: VGE Service Infrastruktur.

Quelle: [27]

Wie aus der Grafik ersichtlich wird, ist die Service Infrastruktur sehr ähnlich zu anderen Web Services aufgebaut.

Um ein Hacken der Server zu verhindern, ist ein Zugriff nur über Firewall möglich, damit ist sichergestellt, dass – unabhängig von der Sicherheit des Services selbst – schon keine Zugriffsverletzung durch Änderungen auf Dateiebene erfolgt. Für die Service Infrastruktur selbst, wird die Sicherheit einerseits durch die Verschlüsselung der Datenübertragung über OpenSSL gewährleistet (hierzu wird für die digitalen Zertifikate X.509 als ITU-T-Standard verwendet), andererseits durch die Verpflichtung zur Autorisierung gegenüber der Infrastruktur und weiterer Sicherheitsmaßnahmen (Web Service Security). Letztendlich wird über den Logger die Verwendung mitprotokolliert, um auch auf diese Weise Sicherheitsverletzungen nachverfolgen zu können.

Bei der eigentlichen Bereitstellung der Service Infrastruktur kann man unterschiedliche Funktionalitäten zur Unterstützung der Anwendungen feststellen, dies sind folgende:

- *Pricing Model,*
- *QoS Management,*
- *Session Management,*
- *Basic Job Handling,*
- *Error Recovery,*
- *Logger.*

Wie bereits zuvor erwähnt ist ein wesentlicher Teil der VGE Service Infrastruktur die Bepreisung und das QoS-Management. Hingegen sind die anderen Funktionalitäten auch bei konventionellen Web Services anzutreffen, wo ebenfalls Funktionalitäten zum Session Management notwendig sind (da das Internet-Protokoll an sich zustandslos („*stateless*“) ist), ebenso wird man sich um die Abwicklung von Jobs, um die Behandlung von Fehlern und um das Mitloggen von Benutzerbefehlen bzw. aufgetretenen Fehlern kümmern.

## VGE Client Infrastruktur

Die VGE Client Infrastruktur unterstützt die Entwicklung von Client Applikationen, die VGE Services verwenden. Die Client Applikationen benutzen diese Infrastruktur und ihre API, um auf VGE Services zuzugreifen. Die Komponenten der VGE Client Umgebung sind in der Abbildung 2.9. dargestellt.

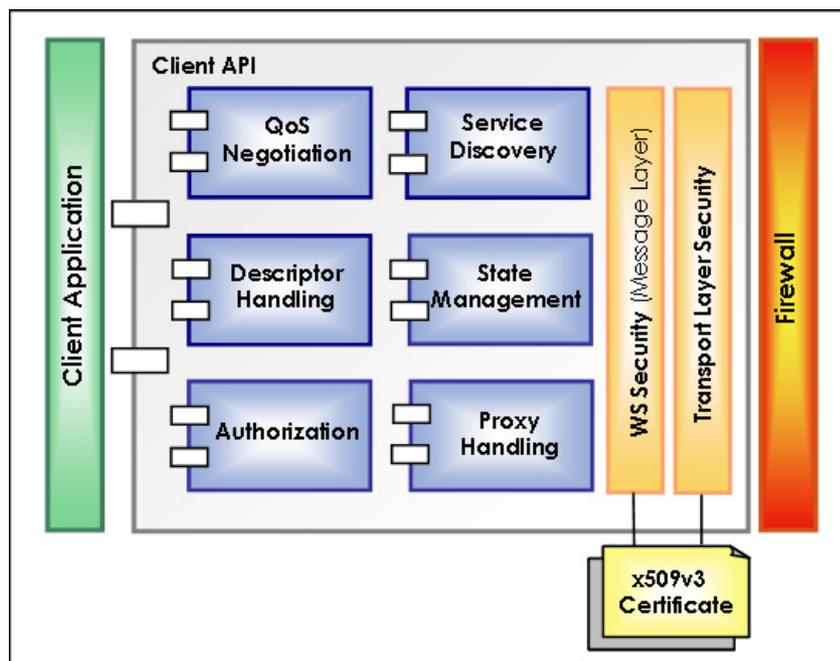


Abbildung 2.9.: VGE Client Infrastruktur.

Quelle: [27]

Auf Seite des Clients findet die Sicherheit ebenfalls auf ähnliche Weise Niederschlag wie server-seitig: um ein Eindringen ohne Autorisierung zu verhindern, wird ebenfalls eine Firewall genutzt und um OpenSSL zu Nutzen wird ebenfalls ein X.509-Zertifikat zur Sicherstellung der Web Security und der Transport Layer Security genutzt.

Die Funktionalitäten für den QoS und die Aufrechterhaltung des Zustandes wird über die *QoS-Negotiation* und *State Management* gewährleistet. Um die Anmeldung gegenüber der VGE Service Infrastruktur zu ermöglichen gibt es weiters Funktionalitäten zur *Authorization*. Um z.B. das Deployment und die Konfiguration zu vereinfachen, gibt es das *Descriptor Handling*. Und letztendlich gibt es speziell auf Client-Seite die Funktionalitäten für *Proxy Handling* (falls ein Zugriff über einen Proxy erfolgt) und natürlich zum *Service Discovery*, denn ohne dass die VGE Service Infrastruktur und deren Funktionalitäten gefunden wird, kann der Client diese nicht nützen.

Aufgrund der Funktionalitäten der Client API kann die Anwendung nun die Service Infrastruktur nach Aushandlung des Preises und der QoS für seine Bedürfnisse nützen.

### **Registry Service**

VGE Registries sind Web Services. Auf diese Weise ist für den User die Implementation nicht ersichtlich. Mittels dem Client API kann auf die Registries zugegriffen werden. Anbieter können ihre Dienste in mehreren Registries, innerhalb einer VGE Umgebung, veröffentlichen. Clients können in diesen Registries nach Diensten suchen.

### **Certificate Authority**

Stellt Server und User Zertifikate nach X.509 aus, die innerhalb der VGE Umgebung zur Identifikation dienen. Hierfür wird eine public key Infrastruktur (PKI) zur Verfügung gestellt. Client-Server Verbindungen können sowohl auf Transport-Ebene mittels TSL (transport layer security [32]) als auch auf Nachrichten-Ebene mittels WSS4J [33] (nach WS-Security [34]) verschlüsselt werden.

## Generic Application Services

VGE verwendet das Konzept der Generic Application Services, um vorhandene HPC Applikationen auf leichte Weise in ein Grid Service einzubauen [28]. Ein Generic Application Service ist eine „konfigurierbare Software Komponente, die eine native Anwendung als Service anbietet und auf die von mehreren Clients über das Internet zugegriffen werden kann“ [28].

Folgende Klassen von Methoden werden angeboten [27]: Job Management, Datentransfer, Fehlerbehandlung und QoS Unterstützung an. In Abbildung 2.10. ist ein Generic Application Service mit den jeweiligen Methoden dargestellt.

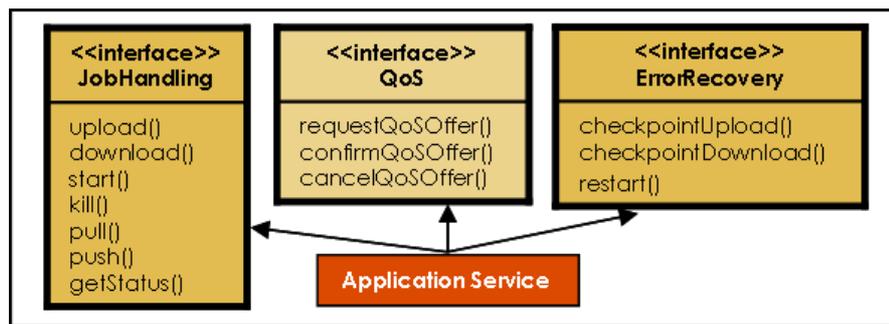


Abbildung 2.10.: VGE Generic Application Service.

Quelle: [27]

Die Methoden des Job Management **upload()** und **download()** unterstützen die Datenübertragung vom Client zum Service, **push()** und **pull()** Methoden die Übertragung von Daten zwischen Services. Mit **start()** und **kill()** kann die Applikation gestartet bzw. beendet werden. Auskunft über den aktuellen Status der Applikation liefert die **getStatus()** Methode.

Die Methoden im Quality of Service Interface unterstützen die QoS Verhandlungen. Im Error Recovery Interface stellt Methoden zur Übertragung von check-point Daten und zum Neustart der Applikation zur Verfügung.

### **2.3.3.1 GEMSS**

Im Projekt GEMSS (*Grid-Enabled Medical Simulation Services* [29]) wurde VGE als Basis verwendet, um medizinische HPC Applikationen in einer Grid Umgebung anzuwenden [30]. Da von diesen medizinischen Applikationen Leben abhängen können, muss eine höhere Verfügbarkeit der Leistungen (Quality of Service) gewährleistet werden können. Dies erfolgt in der VGE über verschiedenste Methoden, wie Mechanismen zur Ausfallssicherheit, Wiederherstellung, Abbildung zeitkritischer Prozesse, aber auch zu Nicht-IT-Themen wie rechtlichen Situationen. (Details zu VGE siehe vorheriges Kapitel).

## **2.4 Grid Business Models - QoS und SLA**

Seit einiger Zeit ist Grid Computing bereits ein gängiges Modell für ressourcenintensive wissenschaftliche Applikationen und es wird auch bereits von einigen kommerziellen Unternehmen eingesetzt. Dies speziell deswegen weil es die gemeinsame Nutzung von Ressourcen und die dynamische Verwendung von Rechenressourcen ermöglicht. Damit erhöht es den Zugang zu verteilten Daten, fördert Flexibilität und Zusammenarbeit bzw. erlaubt es Dienst Anbietern die entsprechenden Größe für die Infrastruktur zu realisieren, um auch unterschiedlich hohe Auslastungen zu verschiedenen Zeitpunkten abzudecken.

Aufgrund dieser Größe sind Grids komplexe Systeme, bestehend aus tausenden von Komponenten über verteilte Server, die auch weltweit verteilt sein können. Je größer ein Grid, desto komplexer wird das System.

Daher ist die Kapazitätsplanung in solchen Umgebungen eine große Herausforderung, aber nur so kann eine gewährleistete *Quality of Service (QoS)* erreicht werden, wobei als zusätzliches Komplexitätskriterium bei weltweit verteilten Grids noch hinzukommt, dass globale *service-level agreements (SLAs)* von lokalen SLAs abhängig sind.

Diese einzelnen Grids sind nun wiederum in der Regel autonom, und schließen sich freiwillig an das globale Grid an [27].

Wie man somit erkennen kann, ist die Sicherstellung der QoS nicht so einfach wie bei der Nutzung eines einzelnen Web Servers. Dies stellt bereits für wissenschaftliche Applikationen ein Problem dar, umso schwerwiegender ist dies natürlich bei kommerziellen Anwendungen. Die Services zur Zusicherung dieses QoS wie z.B. Jobdurchführung und Datenhaltung müssen die vereinbarte QoS liefern. Diese müssen die Verfügbarkeit, Sicherheit und Geschwindigkeit als Faktoren gewährleisten, diese QoS Anforderungen müssen messbar sein, um eine Kontrolle der Vertragseinhaltung gewährleisten zu können.

Diese QoS Zusicherungs-Anforderungen beinhalten [43]:

- **SLA:** QoS sollte durch einen Vertrag zwischen Service-Anforderer und Service-Anbieter im vorhinein ausgehandelt werden. Standard Mechanismen sollten angeboten werden, um solche Verträge zu etablieren und zu verwalten.
- **SLA Erreichung:** Die Ressourcen sollten angepasst werden, um die benötigte Ziele zu erreichen. Dafür müssen Prozesse und Methoden für die Überwachung der Servicequalität, Messen der Ressourcenauslastung sowie Ressourcenplanung und -adjustierung bereitgestellt werden.
- **Migration:** es sollte möglich sein, Services oder Applikationen auf andere Ressourcen zu migrieren, um die Leistung oder Verfügbarkeit zu erweitern.

Nun soll noch das Problem der Heterogenität eines größeren Grid Systems verdeutlicht werden. Wie bereits oben erwähnt, kann die Komplexität der SLA dadurch stark ansteigen. Dabei gibt es zwei generelle Fälle:

- (i) eine Ressource muss mehrere SLA Anforderungen unterstützen und
- (ii) das gesamte SLA eines Grid-Dienstes setzt sich aus den SLAs aller untergeordneter Ressourcen zusammen.

Ad (i)

Jede Ressource in einem Grid kann von beliebig vielen VOs und Diensten genutzt werden. Dabei kann jede VO andere SLA Anforderungen haben, die von der jeweiligen Ressource unterstützt werden müssen (z.B. kann eine Speicher-Ressource in einem Fall eine maximale Downtime als verpflichtende Anforderung haben, in einem anderen Fall wurde hingegen die maximale Response-time vereinbart).

Wird die Ressource nun von mehreren VOs gleichzeitig in Anspruch genommen, und dies mit widersprüchlichen SLA Anforderungen, so muss es Möglichkeiten oder Regeln geben, welche Anforderungen höhere Priorität hat und auf jeden Fall erfüllt werden muss und welche zurückgestellt werden kann, und damit möglicherweise auch die SLA verletzt wird (z.B. im medizinischen Bereich muss eine Anfrage aus der Intensivstation prioritär zu einer Standarduntersuchung behandelt werden).

Ad (ii)

Jeder nicht-triviale Grid-Dienst oder Grid-Applikation nutzt eine Reihe von Grid-Ressourcen. An jede dieser Ressourcen werden SLA Anforderungen gestellt, welche aber nicht zwingend für alle Ressourcen gleich sind (da beispielsweise nicht erfüllbar oder mit zu hohem technischem oder finanziellem Aufwand verbunden).

Durch das Zusammenspiel der einzelnen Ressourcen ergibt sich für den gesamten Dienst damit ein zusammengesetzter SLA.

Prinzipiell gesehen müsste dieser Gesamt-SLA den maximalen Anforderungen aller einzelnen SLAs entsprechen, z.B. wenn in einem SLA für die Speicher-Ressource eine maximale Downtime von 0,01%, in einem zweiten von 0,02% vereinbart wurde, dann ist im Gesamt-SLA eine maximale Downtime von 0,01% zu gewährleisten, weil nur dadurch die Anforderungen beider Einzel-SLAs sichergestellt werden können.

Dadurch kann sich allerdings in manchen Fällen eine nicht akzeptable Gesamt-SLA ergeben, z.B. wenn sich Anforderungen aus Einzel-SLAs widersprechen würden.

Aufgrund dieser Komplexität sowohl was die SLAs und Aufrechterhaltung der QoS betrifft, als auch die Komplexität ein Programm auf mehrere Rechner zu verteilen (manchmal nicht wissend, welcher Rechner dafür verwendet wird), kann eine gewünschte Umsetzung auf Basis von Grid-Technologie problematisch werden. Ein Ansatz wäre es einen Grid unternehmensintern zu betreiben (und damit Probleme mit SLAs als Verträgen mit anderen Unternehmen zu umgehen) oder den Grid nur von einem Provider betreiben zu lassen (somit nur einen SLA zu haben). Ein alternativer Ansatz wäre es stattdessen Cloud-Technologie zu verwenden und damit Komplexität für den Cloud User zu verbergen. Dieser Ansatz soll in den nächsten Kapiteln präsentiert werden.

## 3 Cloud Computing

Nachdem im vorhergehendem Kapitel die theoretischen Konzepte hinter Grid Computing vorgestellt werden, sollen nun in diesem und den nachfolgenden Kapiteln diejenigen von Cloud Computing beschrieben werden und hierbei auch die Unterschiede zum Grid Computing herausgearbeitet werden.

### 3.1 Aspekte

Bei der Definition von Cloud Computing findet man viele Aspekte des On-Demand-Computing wieder. So werden in einer „Cloud“ Hardware-Ressourcen zur Nutzung durch die Cloud User zur Verfügung gestellt. Wissenschaftler der University of California at Berkely heben folgende Besonderheiten hervor [3]:

- die Illusion unendlicher Rechenkapazität, die bei Bedarf zur Verfügung steht,
- die Vermeidung einer Verpflichtung zur Nutzung von Rechenkapazitäten durch den Cloud-Anwender gegenüber dem Anbieter (bzw. in gewisser Weise gegenüber der Soft- und Hardware-Ressourcen bei Nutzung eines eigenen Rechenzentrums) und
- die Möglichkeit Rechenkapazität auf kurzfristiger Basis zu bezahlen.

Somit geht Cloud Computing über andere Ansätze zur Bereitstellung von Rechenkapazität hinaus, kommt gleichzeitig aber der zu Beginn des zweiten Kapitels geschilderten Analogie zur Stromversorgung näher als andere Ansätze.

Die Definition der University of California at Berkeley schränkt den Begriff „Cloud“ nun auf die Hardware und Systemsoftware des Cloud Providers ein. Darauf kann anwendungsspezifische Software aufsetzen und als „*Software as a Service*“ (SaaS) zur Verfügung gestellt werden. Diese kann sowohl vom Cloud Provider als auch von anderen Firmen zur Verfügung gestellt werden. Die Cloud kann aber auch anders verwendet werden. So kann auch eine Plattform für die Entwicklung zur Verfügung gestellt werden („*Platform as a Service*“, PaaS), Kommunikation („*Communication as a Service*“, CaaS) oder auch die Infrastruktur („*Infrastructure as a Service*“, IaaS), siehe hierzu Kapitel 3.3.3. Deshalb ist diese Unterscheidung für die Definition des Begriffs selbst nebensächlich.

Darüber hinaus kann man bezüglich der Definition einer Cloud auch noch unterscheiden, ob sie nur innerhalb eines Unternehmens genutzt wird (Provider und User sind dasselbe Unternehmen) oder ob die Cloud als Dienstleistung anderen Unternehmen zur Verfügung gestellt werden. In ersterem Fall spricht man von „Private Cloud“, in zweiterem von „Public Cloud“. In den weiteren Abschnitten soll, dem Artikel der University of California at Berkeley [3] folgend, nur letzteres betrachtet werden.

### 3.1.1 Arten des Cloud Computing

Die Definition von Cloud umfasst wie beschrieben nur die Bereitstellung von Hardware und systemnaher Software durch den Cloud Provider. Es lassen sich aber für die Bereitstellung von Cloud Computing drei unterschiedliche Arten unterscheiden:

- Cloud: es wird nur die Cloud zur Verfügung gestellt und der Cloud User hat die Kontrolle über alle softwarespezifischen Aspekte, wie Programmiersprachen, Betriebssystem, Datenbanken usw.,
- Cloud und Programmierumgebung: es wird zusätzlich eine Programmierumgebung zur Verfügung gestellt, damit wird zwar einerseits die Handhabung erleichtert, es ist aber nicht mehr möglich direkt die Hardware anzusprechen bzw.,
- Cloud und Anwendung: es wird neben der Cloud bereits die Anwendung zur Verfügung gestellt, die nur mehr konfiguriert werden muss (genaugenommen wird in diesen Fällen meist gar nicht mehr die Cloud selbst als Leistung angeboten, sondern nur mehr die Nutzung der Applikation, die aber ebenso wie bei einer Cloud nutzungsabhängig verrechnet wird). Obwohl die Möglichkeiten für eine Konfiguration dieser Anwendung sehr umfangreich sein können, so ist man dadurch doch eingeschränkter auf Unternehmenseigenheiten einzugehen, als wenn man diese Applikation selbst entwickeln würde (siehe hierzu auch die Diskussion um das Thema „*Make-vs-Buy*“ in der Literatur zur Softwareentwicklung).

In der Realität existierende Cloud Computing-Lösungen werden in Kapitel 4 beschrieben, dort erfolgt auch eine Einordnung dieser Lösungen in die drei unterschiedlichen Arten.

## **3.2 Business Context**

Damit sich Cloud Computing in der Praxis durchsetzen kann, müssen nicht nur informationstechnische Probleme gelöst werden, sondern es ist auch relevant inwieweit sich wirtschaftliche Vorteile für die Nutzung ergeben. Daher soll in den nachfolgenden Abschnitten dargestellt werden, welche Gründe es für Cloud Computing gibt und welche (betriebswirtschaftlichen) Vorteile die Nutzung sowohl für den Anbieter als auch den Nutzer einer Cloud es gibt.

### **3.2.1 Gründe für Cloud Computing**

Nachfolgend wird nun beschrieben, welche Geschäftsmöglichkeiten und Voraussetzungen gegeben sein müssten, damit sich Cloud Computing durchsetzen kann. Aufgrund neuer Technologien, wie sie im Rahmen des Web 2.0 entstanden sind, lässt sich Cloud Computing besser nützen als dies in der Vergangenheit der Fall war. So spielt in vielen Fällen der Anbieter einer Dienstleistung keine Rolle mehr, da ein Zugriff auf Web Services standardisiert möglich ist. Somit kann aber auch von vornherein Software und Hardware getrennt werden und es ist nicht mehr von Relevanz wo die Ressourcen zur Verfügung gestellt werden – ob dies tatsächlich der Anbieter des Services ist oder eben eine Cloud.

Auswirkungen wird dies im speziellen auf folgende Anwendungsgebiete haben, siehe hierzu [3]:

- Mobile, interaktive Applikationen
- Paralleles Batch-Processing
- Geschäftsanalysen (Business Intelligence)
- Erweiterung von rechenintensiven Desktop-Applikationen

Im Gegensatz dazu eignen sich Anwendungen, bei denen der Datentransfer eine große Rolle spielt („Earthbound“), weniger für Cloud Computing. Da in diesen Fällen der Datentransfer länger dauert als durch die Nutzung von mehr Rechenkapazität in der Cloud erspart würde.

### **3.2.2 Vorteile aus Sicht des Cloud Providers**

Ein Cloud Provider kann nun mehrere Gründe haben eine Cloud anderen Unternehmen zur Verfügung zu stellen. Der vorrangige davon ist es natürlich höhere Profite zu erzielen.

Im wesentlichen lassen sich folgende Motive für eine Entscheidung zugunsten der Bereitstellung einer Cloud unterscheiden [3]:

#### **1. Erzielung von Gewinnen**

Aufgrund von Skaleneffekten (z.B. Mengenrabatte für den Ankauf von Hardware, einmalige Softwareentwicklungskosten, die auf mehrere Server bzw. Kunden aufgeteilt werden können) kann ein Cloud Provider den Betrieb seines Rechenzentrums günstiger anbieten als ein kleineres Rechenzentrum (wie es der Fall wäre, wenn jeder sein Kunden ein eigenes betreiben würde).

#### **2. Bessere Nutzung von bereits erfolgten Investitionen**

Da beim Betrieb eines Rechenzentrums Fixkosten in beträchtlicher Höhe anfallen bzw. Kapazität vorhanden ist, die nicht genutzt wird (siehe hierzu auch den nächsten Abschnitt), macht es für den Betreiber Sinn, Rechenleistungen an weitere Kunden anzubieten. Im Zusammenhang mit den Skaleneffekten ergibt sich somit ein weiterer Kostenvorteil.

### **3. Absicherung eines Geschäftsmodells**

Sollte ein Unternehmen bereits Anwendungen erfolgreich verkaufen, die aber auch innerhalb einer Cloud betrieben werden können, so kann es sich für dieses Unternehmen als vorteilhaft herausstellen, selbst eine Cloud zu betreiben.

Hierbei sind zwei Fälle zu unterscheiden:

- Unternehmen, die eine Anwendung des betrachteten Anbieters in ihrer eigenen Cloud anbieten möchten, als auch
- Unternehmen, die eine Anwendung mit derselben bzw. ähnlicher Funktionalität entwickelt haben und dieses nun über ihre Cloud anbieten und sich durch das Alleinstellungsmerkmal „Cloud“ unterscheiden möchten.

Im ersten Fall wird es für das betrachtete Unternehmen weniger Auswirkungen haben, da der andere Cloud-Provider seine Anwendung lizenzieren muss. Es kann sich aber trotzdem entscheiden, davon unabhängig eine Cloud anzubieten, um den Kontakt zu seinen Kunden nicht zu verlieren. Wäre früher das Geschäftsmodell gewesen, dass es die Software direkt an seine Kunden lizenziert und damit direkt Kontakt zu ihnen hat, wäre nun das Geschäftsmodell, dass es die Unternehmen an den Cloud Provider lizenziert und die Kunden mit diesem eine Nutzung vereinbaren. Daher macht es Sinn, dieser Cloud Provider selbst zu sein.

Im zweiten Fall sind die Auswirkungen schwerwiegender, da die Konkurrenz darin besteht, dass Kunden die Software eines anderen Anbieters nutzen können (über die Cloud) und das betrachtete Unternehmen somit Marktanteile verlieren könnte. Daher macht es für das betrachtete Unternehmen Sinn, seine Software ebenfalls über eine Cloud anzubieten, damit es die gleiche Dienstleistung wie sein Konkurrent anbieten kann.

Beispiel für den ersten Fall wäre z.B. die Bereitstellung von SAP-Modulen durch einen Service Provider für Klein- und Mittelbetriebe bzw. durch SAP direkt. Hingegen wäre ein Beispiel für den zweiten Fall die Konkurrenzsituation zwischen Google und Microsoft, wenn Google seine eine Office-Applikation nutzungsabhängig bereitstellt und Microsoft mit Office anschließend nachzieht und diese ebenfalls zur Nutzung über Internet zur Verfügung stellt.

#### **4. Angriff auf einen etablierten Betreiber**

Auch dieser Punkt hängt wieder eng mit den Skaleneffekten beim Betrieb einer Cloud zusammen. Dadurch kann es zu einem Monopol kommen, womit der etablierte Betreiber höhere Preise und damit einen höheren Profit erzielen könnte.

Ein Angriff auf den etablierten Betreiber zur Auflösung dieser Monopolstellung kann für zwei Gruppen interessant werden:

- sowohl für andere Unternehmen, die über einen etwas niedrigeren Preis als den Monopolpreis Kunden gewinnen möchten,
- als auch für Anwender der Cloud selbst, der durch eine eigenständige Bereitstellung niedrigere Kosten haben könnte als durch den Monopolpreis.

Bei ausreichender Größe und damit entsprechen Skaleneffekten bezüglich der Kosten kann dies zu einem Erfolg führen. Der Monopolist wird diesen Versuch aber abzuwehren versuchen, dies kann er einfach indem er den Preis nur soweit senkt, dass sich die Neuschaffung der Cloud durch die anderen Unternehmen nicht mehr rentiert. Dies wird umso schwieriger sein, je schneller die anderen Anbieter ihre Cloud etablieren können und somit für sich selbst niedrigere Kosten durch Größeneffekte lukrieren können.

## 5. Bessere Nutzung von (existierenden) Kundenverbindungen

Da viele IT-Unternehmen bereits in Kundenkontakt stehen, kann es für sie sinnvoll sein ihr Angebot um eine Cloud zu ergänzen. Dies kann aufgrund zwei Faktoren einen Einfluss auf die Kundenverbindung haben:

- aufgrund der bestehenden Kundenverbindung (z.B. bei IBM, die dem Kunden bereits Server oder IT-Dienstleistungen verkaufen), nutzen diese nun zusätzlich die Cloud und Applikationen dafür mit,
- aufgrund der Kundenverbindung durch die Cloud, können dem Kunden auch weitere IT-Dienstleistungen verkauft werden (z.B. Anpassung einer Anwendung für die Cloud oder generell Software-Entwicklung, die auch für lokale Anwendungen erfolgen kann).

Wie man erkennen kann, hängen diese Faktoren auch mit dem 3. und 4. Punkt zusammen: sollte keine Cloud angeboten werden, müsste ein Kunde ein anderes Unternehmen als Cloud Provider finden, der aber wiederum ebenfalls IT-Dienstleistungen anbieten könnte, wodurch nun eine „geteilte“ Kundenbeziehung vorhanden ist und die Gefahr besteht, dass der Kunde nicht nur die Cloud sondern auch IT-Dienstleistungen zukauf. Für das andere Unternehmen ist die Argumentation hingegen genau spiegelverkehrt: durch das Angebot der Cloud kann er einen anderen Kunden abwerben und dadurch einen Angriff auf einen etablierten Betreiber starten.

Dies ist auch im Sinne eines Querverkaufs („*Cross Selling*“) zu betrachten, wo sich ergänzende Produkte und Dienstleistungen verkauft werden, siehe hierzu [54]. So kann nicht nur die Cloud verkauft werden, sondern es wird die Kundenbeziehung dadurch vertieft, dass zusätzlich auch Dienstleistungen z.B. einer Applikation für die Cloud, angeboten werden oder andere Web Services als Draufgabe.

Hierbei sind auch bei der Preisgestaltung verschiedenste Möglichkeiten denkbar, z.B. kann der Preis für die Nutzung der Rechenkapazität sehr niedrig angesetzt werden, während der Gewinn erst über zusätzliche Dienstleistungen erzielt wird oder ein Einstiegsangebot zur Bekanntmachung des Cloud Providers dient (siehe hierzu Kapitel 4 zu den Angeboten kommerzieller Unternehmen).

## **6. Weiterentwicklung zu einer Plattform**

Sollte ein Unternehmen eine Internet-Anwendung, speziell gilt dies für Web 2.0-Anwendungen, zur Verfügung stellen, könnte es für es sinnvoll sein zusätzlich Cloud-Leistungen anzubieten um die Nutzung der eigentlichen Anwendung sicherzustellen.

Dadurch können Nutzer die Cloud zum Betrieb ihrer eigenen Anwendungen verwenden und wenn diese ebenfalls über die eigentliche Internet-Anwendung zur Verfügung gestellt wird, bewegt sich diese hin zu einer Plattform, die nicht mehr nur in Richtung Unternehmen → Kunde funktioniert, sondern vom Kunden wieder zu anderen Kunden.

Ein Beispiel hierfür ist Second Life, wo der ursprüngliche Zweck darin bestand, dass ein Avatar die Welt besuchen konnte und dort kommunizieren konnte. Im Laufe der Weiterentwicklungen wurde aber auch die Möglichkeit geschaffen, dass die Nutzer selbst Weiterentwicklungen durchführen konnten, was letztendlich auch von Unternehmen, z.B. für Marketingpräsentationen, genutzt wurde. Somit wurde aus der Internet-Anwendung „Second Life“ eine Plattform für die Entwicklung von kommerziellen Angeboten. Auch wenn sich dies im Grunde hauptsächlich auf den Marketing- und Vertriebsbereich bezieht und somit nicht ganz der eigentlichen Definition einer Cloud entspricht, so erkennt man doch das Prinzip dahinter.

### **3.2.3 Vorteile aus Sicht des Cloud Users**

Genauso wie es von Seiten des Cloud Providers pekuniäre Motive sind, ist das Gegenstück zu den Profiten der Anbieter für einen Cloud User natürlich die Einsparung von Kosten.

Im folgenden sollen die Einsparungsmöglichkeiten in folgende drei Kostenvarianten eingeordnet werden:

1. Kosten für die Bereitstellung von IT-Infrastruktur,
2. Ausgaben für die Abdeckung bekannter Variabilität der Ressourcennutzung und
3. Kosten für die Abdeckung unbekannter Variabilität der Rechenleistung.

Unter dem ersten Kostenblock sind Ausgaben zu verstehen, die alleine durch die Bereitstellung von IT-Infrastruktur erfolgt. Hierbei werden die Einsparungsmöglichkeiten bei Nutzung eines Cloud Providers am geringsten sein und der bestehenden Infrastruktur der jeweiligen Unternehmen abhängen. So kann ein größerer IT-Anbieter Hard- und Software günstiger ankaufen als ein kleiner Anbieter. Daher kann es für ein Unternehmen günstiger kommen, diese Mengenrabatte „mitzunutzen“ als selbst ein Datacenter aufzubauen. Wenn hingegen die Kosten durch Datentransfer diese Einsparungen überwiegen, wird es für das Unternehmen mehr Sinn machen, ein eigenes Datacenter aufzubauen.

Der zweite Kostenblock entsteht dadurch, dass ein Rechenzentrum derart dimensioniert werden muss, dass die Kapazität für die Spitzenzeiten reicht. Ein Beispiel hierfür wäre, wenn ein Unternehmen zu jedem Monatsultimo Berichte über die Geschäftssituation erstellen muss, während des Monats aber kaum rechenintensive Aufgaben anfallen. Wenn nun zum Beispiel zehn Server deswegen angeschafft werden müssen, obwohl die meiste Zeit nur ein Server genützt wird, fallen die Kosten trotzdem für alle Server an. Wenn nun stattdessen eine Cloud genützt wird, daher für die rechenintensive Reporterstellung On-Demand die Rechenkapazität zugekauft wird, können somit weniger Server für den laufenden Betrieb angeschafft werden. Es sind somit die Kosten für die Anschaffung und Betrieb von Rechenkapazität zu Spitzenzeiten denjenigen Kosten für die Rechenkapazität zu normalen Zeiten plus der zugekauften Cloud-Kapazität gegenüberzustellen. Wenn letztere niedriger sind, zahlt es sich für das Unternehmen aus einen Cloud zu nutzen.

Dies lässt sich auch anhand folgender Grafiken demonstrieren.

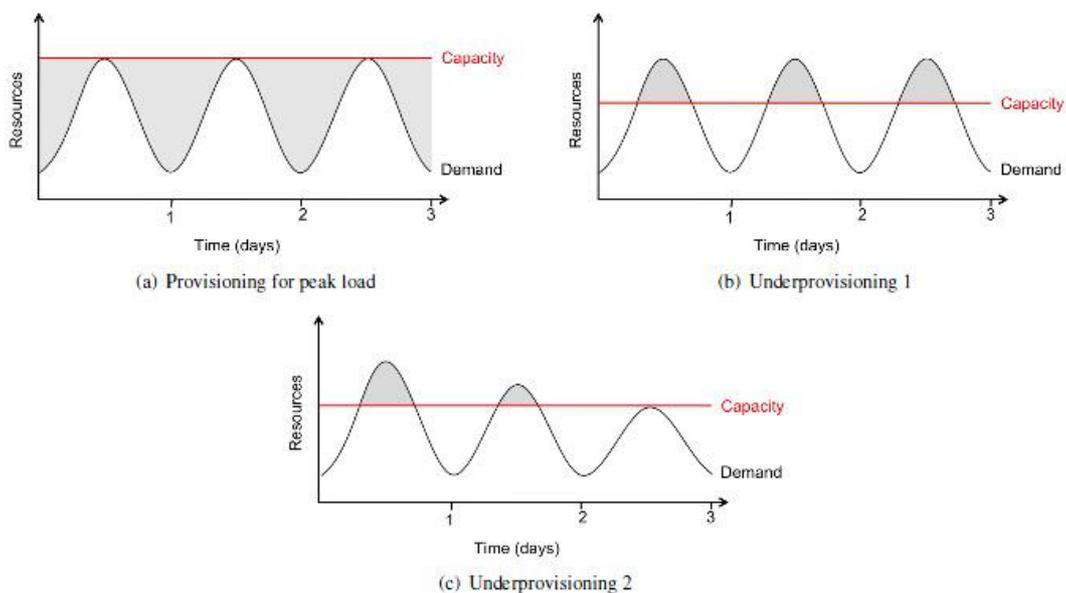


Abbildung 3.1.: Kapazität für Spitzenzeiten.

Quelle: [3]

Wie daraus ersichtlich wird, muss bei einer Kapazitätsbereitstellung für die Spitzenlast mehr Ressourcen zur Verfügung gestellt werden, wodurch aber auch höhere Kosten anfallen. Wird aber weniger Kapazität zur Verfügung gestellt, kann es hingegen passieren, dass manche Anfragen nicht bedient werden können oder es zu längeren Wartezeiten für User kommt. Im schlimmsten Fall werden sich Anwender gänzlich vom Angebot abwenden, daher würden zwar nun wieder weniger Ressourcen benötigt werden und die Kapazität auch zu Spitzenauslastungen wieder ausreichend sein (siehe Grafik c). Durch die verlorenen Anwender gehen aber gleichzeitig Einkünfte verloren.

Im Grunde könnte man die Argumentation nun wieder bei Grafik a) beginnen lassen und behaupten, dass die Kapazität nun wieder überdimensioniert sei, wodurch nach einer Kürzung wieder eine Wenigernutzung wie in Grafik c) erfolgen würde. Letztlich wäre dann der Traum eines jeden Kostensparers, nämlich Kosten von Null, erreicht, aber nachdem es keine Anwender mehr gibt, wäre auch der Ertrag gleich Null.

Der letzte Kostenblock entsteht dadurch, dass in vielen Fällen die benötigte Rechenkapazität zu Spitzenzeiten nicht bekannt ist. So kann es passieren, dass aufgrund unvorhergesehener Ereignisse es plötzlich zu einer starken Nachfrage nach Serverleistung kommt (wie zum Beispiel bei News-Websites bei den Ereignissen vom 11. September 2001). Somit muss das Rechenzentrum noch größer dimensioniert werden und es entstehen Kosten für den Betrieb von Servern, die eventuell nicht genutzt werden. Gleichzeitig besteht aber das Risiko, dass bei einer zu kleinen Dimensionierung Nutzer verloren gehen, weil die Serverkapazität in diesen außergewöhnlichen Fällen nicht bereitgestellt werden kann. Durch die Nutzung einer Clouds kann dieses Risiko auf den Cloud Provider übertragen werden.

Bei all diesen Überlegungen ist aber immer in Betracht zu ziehen, dass durch die Nutzung der Clouds nicht nur für die Nutzung desselbigen Kosten entstehen, sondern auch noch weitere Kosten in Betracht zu ziehen sind (zum Beispiel für den Datentransfer zwischen den Rechenzentren). Weiters hängt das Einsparungspotenzial in der Gesamtwirtschaft auch davon ab, inwieweit die Nachfrage nach Rechenkapazität zwischen den Unternehmen korreliert. Wenn diese sehr stark negativ korreliert, werden sich die höchsten Einsparungspotenziale ergeben, bei starker positiver Korrelation die wenigsten.

Wenn zum Beispiel ein Unternehmen nur in der ersten Monatshälfte Rechenkapazität nachfrägt, ein anderes Unternehmen in der zweiten Monatshälfte, würde ein Cloud Provider mit der Hälfte der Rechenkapazität auskommen als wie wenn jedes Unternehmen ein eigenes Rechenzentrum betreiben würde und seine Ressourcen die jeweils andere Monatshälfte brach liegen lassen würde. Hingegen würde ein Cloud Provider, wenn jedes Unternehmen die Rechenkapazität in der gleichen Monatshälfte nachfrägt, dieselbe Anzahl an Server bereitstellen müssen, wie bei alleinigen Betrieb der Rechenzentren.

Diese Überlegungen fassen die Autoren des Artikels [3] in folgender vereinfachten Formel zusammen:

$$\text{UserHours}_{\text{cloud}} \times (\text{revenue} - \text{Cost}_{\text{cloud}}) \geq \text{UserHours}_{\text{datacenter}} \times (\text{revenue} - \text{Cost}_{\text{datacenter}} / \text{Utilization})$$

Die linke Seite gibt den Gewinn bei Nutzung eines Cloud Providers wieder, während die rechte Seite denselbigen bei Einsatz eines eigenen Rechenzentrums darstellt. Da für die Cloud nur die tatsächliche Nutzung pro Stunde gezahlt wird (Nutzung = 1), ist dieser Term auf der linken Seite nicht notwendig.

Hingegen ist, wie bereits in den vorhergehenden Absätzen beschrieben, bei einem eigenständigen Betrieb des Rechenzentrums notwendig, mehr Kapazitäten zur Verfügung zu stellen als genutzt wird. Daher müssen die Kosten noch um die Nutzung bereinigt werden ( $\text{Nutzung} < 1$ , gemäß den Autoren [3] im Normalfall zwischen 0,6 und 0,8). Zwar muss auch der Cloud Provider eine mögliche Nicht-Nutzung seiner Server in den Kosten berücksichtigen. Da er aber mehrere Kunden hat, hat dies im Normalfall geringere Auswirkungen.

Nach Einsetzen von Werten anstatt der Variablen kann der Cloud User eine Entscheidung über die Nutzung der Cloud treffen: sollte die Ungleichung erfüllt sein, wird er die Cloud nützen, bei Nicht-Erfüllung hingegen ein eigenes Rechenzentrum betreiben. Sollten beide Seiten zufällig genau gleich sein, kann er anhand der Formel keine der beiden Alternativen bevorzugen.

Die Werte der einzelnen Parameter sind immer situationsabhängig und können bei jeder Entscheidung bezüglich der Nutzung einer Cloud unterschiedlich aussehen. Beispielhaft soll dies anhand folgender zwei Fälle illustriert werden.

Im ersten Fall handelt es sich um ein Unternehmen, das einen Web Server betreiben möchte, um damit seine Produkte an die Kunden verkaufen zu können. Hierbei wird der Server aber nur in einem Monat pro Jahr benötigt (da es sich um den Hersteller von Weihnachtskugeln handelt, nach denen während des restlichen Jahres keine Nachfrage besteht, zur Vereinfachung wurde angenommen, dass die Zugriffe zwischen 9 bis 21 Uhr erfolgen, dann aber gleichmäßig verteilt sind). Somit beträgt die Utilization  $1/24$ . Da es sich um ein kleines Unternehmen handelt, wären die Kosten für den Betrieb des Servers in einem eigenen Rechenzentrum relativ hoch im Vergleich zur Nutzung einer Cloud, daher wurden als Kosten für das Unternehmen 10 Geldeinheiten pro benutzter Stunde und für die Cloud 2 Geldeinheit pro benutzter Stunde angenommen.

Die Einkünfte sind unabhängig von der Wahl des Serverstandortes und betragen immer 20 Geldeinheiten pro benutzter Stunde. Aufgrund der geplanten Nutzungsdauer von vier Jahren ergibt sich die Anzahl der genutzten Stunden über 4 Jahre \* 30 Tage \* 12 Stunden = 1440 Stunden, die in wiederum davon unabhängig ist, ob eine Cloud oder ein eigenes Datacenter genutzt wird.

Nach Einfügen in die Formel ergibt sich folgende Ungleichung:

$$1.440 \times (20 - 2) \geq 1.440 \times (20 - 10 / 0,04166)$$
$$25.920 \geq -316.800.$$

Wie auch zu erwarten war, ist der Betrieb in einer Cloud vorteilhafter. Aufgrund der extremen Annahme einer sehr geringen Nutzung während eines Monats wäre der Betrieb im eigenen Rechenzentrum sogar mit Verlusten für das Unternehmen verbunden. Aber auch bei einer stärkeren Nutzung wäre die Cloud immer noch vorteilhafter, bei einer Nutzung von 1 (also der unrealistischen Annahme, dass auch in den Nachtstunden der Server gleichmäßig genutzt wird) wäre der Gewinn mit 14.400 immer noch niedriger als in der Cloud.

Im zweiten Fall soll dem nun ein größeres Rechenzentrum gegenübergestellt werden, bei dem die Kosten niedriger als in der Cloud sind (nur 1 Geldeinheit, weil nur die tatsächlichen Kosten ohne Gewinnaufschlag des Providers verrechnet werden) und gleichzeitig die Nutzung sehr hoch ist (gleich 0,9, weil auch während der Nacht Batch-Läufe zur Kalkulation von Berichtsergebnissen ausgeführt werden). Die Einkünfte und Nutzungsstunden seien wiederum unabhängig davon, ob die Cloud oder das eigene Rechenzentrum genutzt werden und seien 7.884 Stunden und Einkünfte von 10 Geldeinheiten. Somit ergibt sich folgende Ungleichung:

$$7.884 \times (10 - 2) \geq 7.884 \times (10 - 1 / 0,9)$$
$$63.072 \geq 70.080.$$

Nun ist die Entscheidung zugunsten eines eigenen Rechenzentrums vorteilhafter als zugunsten der Cloud. Damit das Unternehmen indifferent wird, müsste die Nutzung auf 50% fallen, dann wären die Einkünfte bei beiden Betriebsarten gleich. Bei unter 50% würde sich das Unternehmen zugunsten der Cloud entscheiden.

Wie bei den Beispielen ersichtlich ist, sind zwei wesentliche Faktoren entscheidend: einerseits die Kostendifferenz zwischen einer Nutzungsstunde in der Cloud bzw. im eigenen Rechenzentrum und der Nutzungsgrad im eigenen Rechenzentrum. Da in der Cloud nur die tatsächlich angefallenen Stunden bezahlt werden, müssen die Kosten normalerweise unter denjenigen der Cloud liegen. Je weniger dann das eigene Rechenzentrum genutzt wird, desto eher macht ein Wechsel in eine Cloud Sinn. In der Praxis müsste man auch noch berücksichtigen, ob man aufgrund von Datentransfers, Administrationsaufgaben eventuell mehr Nutzungsstunden in einer Cloud benötigt oder ob durch Reputationsverlust oder ähnlichem die Einkünfte in einer Cloud niedriger anzusetzen wären als im eigenen Rechenzentrum (siehe hierzu auch den nächsten Abschnitt).

### **3.2.4 Mögliche Probleme und Lösungsansätze**

Auch wenn sich in der theoretischen Betrachtung der letzten beiden Abschnitte sowohl für Cloud Provider als auch Cloud User Einsparungspotenziale ergeben, so gibt es für die Nutzung von Cloud Computing in der Realität noch einige Fallstricke.

In diesem Abschnitt sollen nun die wichtigsten davon, hierbei folgend [3], präsentiert, aber auch mögliche Lösungsmöglichkeiten erläutert werden.

Tabelle entnommen aus [3]:

	Probleme	Lösungsansätze
1	Verfügbarkeit des Services	Verwende mehrere Cloud Anbieter, um <i>Business Continuity</i> anbieten zu können; Verwende Elastizität, um sich gegen DDOS Attacken zu schützen
2	Einschränkung der Daten	Standardisiere APIs; Stelle kompatible Software bereit, um <i>Surge Computing</i> zu ermöglichen
3	Datenvertraulichkeit und Revisionssicherheit	Wende Verschlüsselung, VLANs und Firewalls an; Vereinbare gesetzliche Regelungen für geographische Serverstandorte
4	Engpässe beim Datentransfer	Nutzung von Versanddiensten, um Festplatten physisch zu transportieren; Daten Backup/Archivierung; Geringere WAN Router Kosten; Höhere Bandbreite LAN Switches
5	Unvorhersagbarkeit der Performance	Verbesserte Virtual Machine Unterstützung; Flash Memory; Gruppen Scheduling VMs für HPC Applikationen
6	Skalierbare Speicherung	Erfinde skalierbaren Speicher
7	Fehler in großen, verteilten Systemen	Erfinde Debugger, das auf verteilte VMs baut
8	Schnelle Skalierung	Erfinde Auto-Skalierer, die auf Lernprozesse aufbauen; Snapshots um Cloud Computing Konsumenten zu unterstützen
9	<i>Reputation Fate Sharing</i>	Biete <i>reputation-guarding</i> Services, z.B. wie für E-Mail
10	Software Lizenzierung	<i>Pay-for-use</i> Lizenzen; Mengenrabatte durch größere Anzahl von Lizenzierungen

## 1. Verfügbarkeit des Services

Wie jede IT-Hardware können auch die Server der Cloud jederzeit ausfallen. Daher ist die Verfügbarkeit – speziell im SLA – ein Thema zwischen dem Anbieter einer Cloud und den Kunden. Wenn diese nicht ausreichend ist, wird ein Kunde auch bei ansonsten gegebenen Kostenersparnissen die Cloud nicht nützen. Hierbei kann die Verfügbarkeit sowohl aus objektive Gründen, z.B. weil ein schlechter Anbieter erwiesenermaßen die Verfügbarkeit nicht gewährleisten kann, als auch aus subjektiven Gründen, nicht als genügend angesehen werden. So kann ein Kunde auch nur glauben, dass die Verfügbarkeit nicht gewährleistet werden kann oder er die Angst eines Kontrollverlustes hat, weil nicht mehr die eigenen IT-Arbeiter die Wartung der Server durchführen. Weiters kann es – speziell bei bekannten Anbietern – zu einem verstärkten Anreiz kommen, diesen zu hacken oder über Distributed Denial of Service-Attacken lahmlegen zu wollen.

Speziell letzteres kann zu Problemen führen, da ja im Unterschied zu einem In-House-Server ein Zugang über Internet (auch wenn dies ein VPN ist) ermöglicht werden muss. Dieses Problem kann aber durch Sicherheitsexperten beim Cloud Anbieter, wenn nicht schon gänzlich behoben, zumindest durch kontrolliert und minimiert werden. Dieser Aspekt des Expertentums spielt eine wesentliche Rolle bei der Relativierung dieses Hindernisses. Da aufgrund der Größenvorteile eines Cloud Anbieters mehr Expertenwissen vorhanden ist (je größer ein Anbieter sein wird, desto tiefer können einzelne Mitarbeiter sich speziellen Problemen zuwenden). Im Gegensatz hierzu muss dies, auch bei relativ großen Unternehmen, nicht der Fall sein. Ganz besonders wird dies aber nicht der Fall sein bei Klein- und Mittelbetrieben, wo ein IT-Administrator neben Server-Wartung und Sicherheitsfragen auch noch andere Aufgaben haben kann oder es von vornherein keinen eigenen IT-Administrator gibt, sondern dies ein Mitarbeiter als „Nebenjob“ macht.

## **2. Einschränkung der Daten**

Ein weiteres Hindernis stellt proprietäre Datenhaltung durch den Cloud Anbieter dar. Dadurch kann eine Migration zu einem anderen Betreiber verhindert oder durch höhere Migrationskosten erschwert werden. So werden z.B. die im folgenden Kapitel beschriebenen Amazon Mirror Images (AMI) wohl nicht, so wie sie gespeichert wurden, auf einen Server eines anderen Anbieters kopiert werden können. Dadurch kann der Anbieter höhere Kosten verlangen, weil sich ein Wechsel nicht rentieren würde.

## **3. Datenvertraulichkeit und Revisionsicherheit**

Ein weiteres Problem stellt eine mangelnde Datenvertraulichkeit und Revisionsicherheit dar. Durch den Wechsel zu einem Cloud Anbieter haben nun nicht mehr nur die Mitarbeiter des eigenen Unternehmens Zugang zu den Daten, sondern nun auch diejenigen des Cloud Anbieters. Dies kann sogar der Fall sein, wenn vereinbart wurde, dass nur der Kunde Root-Zugang auf die Server bekommt. Da die Mitarbeiter des Cloud Anbieters physisch Zugang zur Hardware haben, könnten sich diese unbefugt Zugang verschaffen. Somit wären nun nicht mehr nur die eigenen Mitarbeiter zu kontrollieren, sondern auch von Fremdunternehmen. Dies ist vergleichbar zur Problematik, dass heutzutage für Reinigungstätigkeiten keine eigenen Mitarbeiter mehr beschäftigt werden. Diese Probleme können aber durch Verschlüsselungen und Firewalls gelöst werden, es können auch Vereinbarungen getroffen werden, dass eigene Mitarbeiter bei Vor-Ort-Problemlösungen zugegen sind. Oder falls dies aus gesetzlichen oder Revisionsgründen notwendig ist, kann auch ein Hosting aufgrund des geographischen Serverstandorts vereinbart werden.

#### **4. Engpässe beim Datentransfer**

Ein wesentliches Problem stellen mögliche Engpässe beim Datentransfer dar. So könnte es aufgrund der benötigten Rechenleistungen Sinn machen, z.B. Ultimo-Berichte in einer Cloud rechnen zu lassen (wie bereits oben erwähnt macht eine Cloud bei dieser Art von Anwendungen Sinn: die Rechenleistung wird nur einmal pro Monat, ungefähr ein bis zwei Tage genützt, während sie das restliche Monat nicht benötigt wird). Aber wenn hierfür die gesamten angefallenen Monatsdaten an den Anbieter übertragen werden müssen, kann diese längern dauern als die Berechnung der Berichte selbst. Eine Alternative wäre diese Daten auf einer Festplatte zu speichern und diese physisch über einen Versanddienst an den Cloud Anbieter zu senden. Eine andere Möglichkeit wäre es die Daten bereits inkrementiell während des Monats in kleineren Datenmengen zu übermitteln.

#### **5. Unvorhersagbarkeit der Performance**

Speziell wenn auf einem Hardware-Server mehrere Virtual Machines laufen, kann es zu Performanceeinbussen kommen. Sowohl hinsichtlich physischen Zugriffen auf die Festplatte, als auch weil nicht bekannt ist, ob ein Nutzer einer anderen Virtual Machine am selben Server gerade zum Zeitpunkt der gewünschten Rechenleistung ebenfalls einen rechenintensiven Job startet. Dies kann sowohl softwareseitig (bessere programmierte Virtual Machines), als auch hardwaremäßig gelöst werden (z.B. Flash-Memory für die Festplatte zur Unterstützung der Zugriffsgeschwindigkeit).

## 6. Skalierbare Speicherung

Ein weiteres Hindernis stellen noch ungeklärte Fragen bezüglich der Skalierung des Speichers dar. So sind zwar ein bis mehrere Server jederzeit zuschaltbar, der darin verbaute Speicher (RAM) hingegen ist fix eingebaut. Da aber der Fall auftreten kann, dass zwar die CPU-Leistung für die Anwendung ausreichend ist, nicht aber der Speicher, würde es Sinn machen nur zusätzlichen Speicher einzubauen, nicht aber einen zusätzlichen Server zuzuschalten. Da hierzu aber das Gerät geöffnet werden müsste und der Speicher eingesetzt werden muss, ist dies kein gangbarer Weg. Speziell weil es ja nur eine kurzfristige Belastungsspitze sein könnte und nach Fertigstellung einiger komplexerer Rechnungen der ursprüngliche Speicher wieder reichen würde.

Erst wenn es möglich wäre auch den Speicher skalierbar zur Verfügung zu stellen, wäre es möglich auch diesen effizient zur Nutzung in einer Cloud bereitzustellen. Ähnlich zu Netzwerk-Festplatten, die durch mehrere Server genutzt werden könnten, müsste hierzu „Netzwerk-RAM“ geschaffen werden, der damit aber zwangsläufig aber auch zu einer längeren physischen Verbindung zwischen CPU und RAM führen würde.

## **7. Fehler in großen, verteilten Systemen**

Je komplexer ein System, desto fehleranfälliger ist es auch. Das ist eine Grundregel, die jeder Programmierer bereits als Anfänger lernt. Da die Skalierung in einer Cloud komplexer ist, können daher Fehler auftreten an die vorher noch nicht gedacht wurde. Zum Beispiel könnte ein Programmierer einen Internet-Auftritt für einen Server optimiert haben. Während auf diesem alles einwandfrei läuft, treten bei Verwendung auf mehreren Servern und Nutzung von Load Balancing plötzlich Fehler auf. Dieses Problem wird noch dadurch erschwert, dass Effekte erst ab einer gewissen Anzahl an Instanzen auftreten können.

Für derartige Fragenstellung gibt es Tools, die gleichzeitige Zugriffe auf einen Internet-Auftritt simulieren können. Damit ist es möglich vor dem Produktiveinsatz die Reaktion des Auftritts bei Bereitstellung in der Cloud zu testen.

## **8. Schnelle Skalierung**

Gerade aufgrund der Bezahlung pro genutzter Rechenzeit, wäre eine schnelle Skalierung wünschenswert. So kann es passieren, dass die Spitzenlast nur für kurze Zeit auftritt und die Last danach wieder abnimmt. In manchen Fällen kann dies vorhersagbar sein, z.B. wenn bei Wahlen User sofort die Ergebnisse der ersten Hochrechnung sehen möchten. In diesem Fall ist bekannt, wann diese veröffentlicht wird und es können rechtzeitig weitere Server bereitgestellt werden. In anderen Fällen wird es aber nicht bekannt sein (aus unterschiedlichsten Gründen und sei es nur, weil ein Internet-Nutzer einer größeren Gruppe einen Hinweis auf eine spezielle Information gibt). In diesem Fall wäre es notwendig, den Server sofort zur Verfügung zu stellen.

Wenn es hierzu aber erst notwendig ist, den Server hochzufahren, oder weil er von mehreren Anwendern genutzt wird, das Image für den Auftritt zu laden, wird einige Zeit vergehen, in der einige User enttäuscht werden.

## **9. *Reputation Fate Sharing***

Ein ganz spezielles Problem stellt die Nutzung derselben Cloud durch mehrere Anwender dar. Im Gegensatz zum firmeneigenen Betrieb der Server und damit auch Verwendung der firmenspezifischen IP-Adresse, kann es in einer Cloud dazu kommen, dass ein anderer Nutzer kriminelle Aktivitäten unternimmt, die Auswirkungen auf alle anderen Nutzer hat. Zum Beispiel könnte ein Nutzer Spam-Mails über seine IP-Adresse versendet, und obwohl zwar die IP-Adressen der anderen Kunden im Ganzen unterschiedlich sind, könnte ein E-Mail-Provider den IP-Adresskreis des Cloud Anbieters auf seine Blacklist gesetzt haben. Dadurch verlieren die „braven“ Nutzer ihren guten Ruf und können keine E-Mails mehr versenden, obwohl der Spam-Versender das Verbrechen begangen hat, aber damit auch ihren Ruf geschädigt hat.

## **10. *Software Lizenzierung***

Als letztes angeführtes Hindernis soll abschließend noch Probleme mit der Software Lizenzierung genannt werden. Das einfachste Beispiel wäre ein Lizenzmodell, wo die Kosten auf der Anzahl der genutzten PC basieren. Somit muss die Software für jeden der Server lizenziert werden, auch wenn einer der Server vielleicht nur eine halbe Stunde jedes Monats zur Spitzenzeit in Betrieb ist. Etwas abgeschwächt werden könnte das Problem durch die Nutzung von Mengenrabatten, so würde zwar die gleiche Anzahl an Lizenzen benötigt werden, diese wäre aber durch den geringen Preis pro Lizenz in Summe billiger als wie wenn jeder Cloud User die Lizenz einzeln kaufen würde.

Eine mögliche Lösung wäre es, wenn der Software-Anbieter das Lizenzmodell an die Nutzung in einer Cloud anpasst (z.B. ebenfalls ein zeitabhängiges Modell einführt). Eine weitere Alternative wäre es Freeware oder Open-Source-Software einzusetzen. Damit würden keine Lizenzkosten anfallen und damit das Problem der zeitabhängigen Nutzung elegant gelöst.

### **3.3 Virtualisierung**

Wie im vorhergehenden Abschnitt beschrieben wurde, stellt die Möglichkeit zusätzliche Ressourcen für das Cloud Computing bereitzustellen einen wesentlichen Faktor dar. Je schneller die benötigte Software-Umgebung bereitgestellt werden kann, desto besser ist dies. Anstatt dafür nun jedes Mal einen dedizierten Server (Hardware) zu starten und dort die Anwendungen für das Cloud Computing auszuführen, ist es vorteilhafter stattdessen virtuelle Maschinen zur Verfügung zu stellen, die jederzeit auf einen der verfügbaren Server mit freien Ressourcen laufen kann.

Durch diese Trennung zwischen Hardware und virtuellem Server (bzw. Betriebssystem und virtuellem Server) lässt sich auch weit leichter eine Portabilität sicherstellen. Dies betrifft sowohl Aspekte betreffend der Hardware (z.B. wenn aufgrund von Defekten oder des technischen Fortschritts ein älterer Server durch ein aktuelles Modell ersetzt wird oder sich das Cloud-Rechenzentrum für eine andere Hardware-Architektur entscheidet), als auch des Betriebssystems (z.B. Update auf eine neue Version des eingesetzten Betriebssystems oder Migration auf ein anderes Betriebssystem). Wenn eine Anwendung ohne Virtualisierung laufen würde, müsste die Anwendung selbst migriert werden, bei einer Virtualisierung hingegen kann sie weiterhin in der Virtualisierungssoftware laufen, nur diese muss für das neue Betriebssystem verfügbar sein.

Bei einem eigenen Rechner für jede Cloud-Anwendung wäre der Aufbau des Systems folgendermaßen:

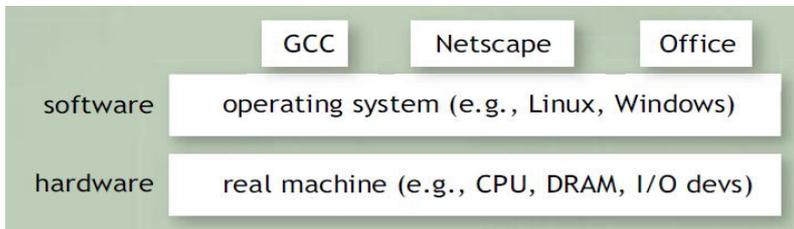


Abbildung 3.2.: Rechensystem ohne Virtualisierung.

Quelle: [58]

Die Hardware muss in diesem Fall für jede unterschiedliche Cloud-Anwendung extra zur Verfügung gestellt werden. Im Unterschied dazu wäre der Aufbau bei einer Virtualisierung in dieser Weise ausgestaltet:

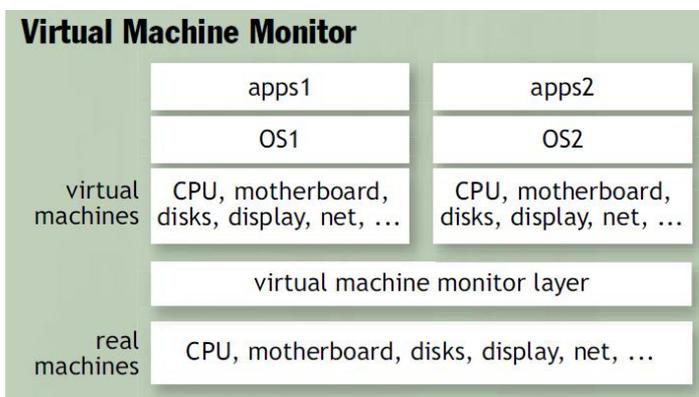


Abbildung 3.3.: Rechensystem mit Virtualisierung.

Quelle: [58]

Auf diese Weise können auf jedem realen Rechner (Hardware) mehrere virtuelle Rechner (softwaremäßig) laufen. Die Anzahl der virtuellen Maschinen ist abhängig sowohl von den Ressourcen der realen Rechner, als auch der Nutzung jeder einzelnen virtuellen Maschine.

Sollte nun eine neue Cloud-Applikation gestartet werden, muss in diesem Fall nur ein Rechner gefunden werden, bei dem freie Ressourcen verfügbar sind. Andere Cloud-Anwendungen die bereits auf diesem Rechner laufen, werden von der neuen Applikation nicht beeinflusst – sie läuft in einer anderen, davon getrennten virtuellen Maschine. Nun wenn alle Rechner ausgelastet sind, würde ein zusätzlicher Hardware-Rechner benötigt werden.

Die hier dargestellte Vorgehensweise stellt die ursprüngliche Form der **hardware-level virtualization** dar. Davon zu unterscheiden sind noch die **operating system-level virtualization**, bei der die Virtualisierungsebene zwischen Betriebssystem und Anwendung sitzt, somit alle Anwendungen das gleiche Betriebssystem verwenden, aber untereinander getrennt sind und die **high-level language virtual machines**, bei der die Virtualisierungsebene eine Anwendung ist, die eine Programmiersprache darstellt für die wiederum darauf laufende Programme geschrieben werden. Das bekannteste Beispiel hierfür ist Java und die JVM.

### 3.3.1 Eigenschaften von virtuellen Maschinen

Die Virtualisierung weist nun folgende Eigenschaften auf [58]:

- **Kompatibilität der Software:** Aufgrund der Abstraktion zwischen der Applikation und der darunterliegenden Ebene durch die virtuellen Maschinen, muss nur darauf geachtet werden, dass die Anwendung in der virtuellen Maschine läuft und sie ist damit automatisch auf kompatibel mit den darunter liegenden Ebenen. Je nach Art der Virtualisierung handelt es sich dabei um Hardware, Betriebssystem und/oder Laufzeitumgebung der Programmiersprache. So wäre eine Java-Applikation immer gleich, egal ob die JVM auf einem Windows- oder einem Linux-Betriebssystem installiert wurde, man beachte dazu das Java-Motto „Write once, run anywhere.“

- **Isolation:** Darüber hinaus trennt die Virtualisierung die Applikationen jeder einzelnen virtuellen Maschine von denjenigen der anderen virtuellen Maschine bzw. vom realen Rechner. Somit führt ein Absturz einer virtuellen Maschine (bzw. einer darin laufenden Applikation) nicht zum Absturz oder Verlangsamung von Applikationen einer anderen virtuellen Maschine. Ebenso werden die Daten und die Performance isoliert.

Ersteres führt dazu, dass keine Applikation einer virtuellen Maschine auf die Daten einer anderen zugreifen kann. Dies hat speziell Vorteile, wenn eine virtuelle Maschine gehackt wird. So ist es auch für Private möglich, dass sie sich eine virtuelle Maschine für Surf-Aktivitäten und eine andere für Internet-Banking installieren. Wenn man nun beim Surfen auf eine präparierte Website kommt und dadurch diese virtuelle Maschine virenverseucht wird, wäre die Internet-Banking-VM weiterhin sicher.

Bei der Isolierung der Performance erfolgt die Zuweisung von Rechnerressourcen durch den Virtualisierungslayer weiterhin in gleicher Weise, auch wenn eine Applikation in einer der virtuellen Maschine diese stärker in Anspruch nimmt. Speziell Windows ist dafür bekannt, dass dies dort nicht in gleicher Weise funktioniert: bei Beanspruchung der CPU durch eine Applikation kommt es in diesem Betriebssystem sehr oft zu einer negativen Beeinflussung der anderen laufenden Applikationen.

- **Kapselung der Applikation:** Durch die zusätzliche Ebene der Virtualisierung kommt es zu einer Kapselung der Applikation, wodurch deren Ausführung beeinflusst werden kann. Bekannt ist so zum Beispiel beim Laufzeitumgebungen von Programmiersprachen die *Garbage Collection*, oder es können Überprüfungen während der Laufzeit zur Vermeidung von Klassen- oder anderen Programmierfehlern stattfinden. Im Allgemeinen bietet der Virtualisierungslayer somit eine bessere Ausführungsumgebung als das Betriebssystem direkt.

- **Performanceeinbußen:** Den einzigen Nachteil der Virtualisierung stellen die Performanceeinbußen durch die zusätzliche Ebene dar. Die Virtualisierungssoftware muss die Applikationsausführung an die virtualisierten Ressourcen überwachen und weiterleiten. Dadurch kommt es zwangsläufig zu einer längeren Ausführungszeit als bei einer direkten Ausführung auf den Hardware-Ressourcen bei Weglassung der Virtualisierungsebene.

### 3.3.2 Anwendungsbeispiele von virtuellen Maschinen

Im Folgenden sollen nun einige Anwendungsbeispiele für die Nutzung von virtuellen Maschinen dargestellt werden:

- **Server-Rechenzentren:** Speziell beim Betrieb von Websites kleinerer Unternehmen oder von privaten Websites, wird ein eigener Rechner für jede Website - auch wenn dies nicht nur statische, sondern dynamische sind - überdimensioniert sein. Trotzdem kann es vorteilhaft sein, dass ein eigener Server zur Verfügung gestellt würde, z.B. weil regelmäßige Jobs auf Betriebssystem-Ebene laufen sollen, wenn fehlerhafte Programmierungen nicht vorab vermieden werden können und diese den Betrieb für die anderen Kunden negativ beeinflussen würden.

Mit Hilfe der Virtualisierung kann nun jedem Kunden ein eigener (virtueller) Server zur Verfügung gestellt werden, auf dem er (mehr oder weniger) machen kann, was er möchte. Wenn dieser Kunde nun eine fehlerhafte Applikation produktiv schaltet, die zu einem Absturz des Betriebssystems führt, stürzt dadurch nur dasjenige in seiner virtuellen Maschine ab, während die virtuellen Maschinen der anderen Kunden auf diesem Server unbeeinflusst davon weiterlaufen.

Weitere Vorteile können sich durch ein vereinfachtes Backup jedes virtuellen Servers ergeben.

- **Snapshots einer bestimmten Konfiguration:** Eine weitere Möglichkeit ist es für bestimmte Applikationen eine virtuelle Maschine einzurichten und dieses als Snapshot abzuspeichern. So könnte es, wie bereits im vorhergehenden Abschnitt beschrieben, sinnvoll sein eine virtuelle Maschine nur für den Besuch von Internet-Seiten zu konfigurieren. Dies könnte z.B. ein abgespecktes Linux-System nur mit installierten Internet-Browser sein. Genau dies wird als Snapshot gespeichert. Sollte es einem Hacker nun tatsächlich gelingen, diese VM mit Viren zu verseuchen, muss nun nicht das gesamte Betriebssystem mit allen Applikationen neu installiert werden. Stattdessen ist es ausreichend nur die Surf-VM zu löschen und den Snapshot neuerlich einzurichten. Somit hat sich der Arbeitsaufwand entscheidend verringert, zusätzlich hat diese Vorgehensweise den Vorteil, dass weniger Viren für Linux (dem Surf-Betriebssystem) existieren, während die Vielzahl an Applikationen für Windows als Gast-Betriebssystem genützt werden können.
- **Software-Entwicklung und Tests:** Für eine professionelle IT-Entwicklung hat die Virtualisierung den Vorteil, dass Tests vereinfacht werden können. Anstatt für jedes Betriebssystem, Version davon bzw. von unterschiedlichen Konfigurationen einen Rechner vorrätig zu haben, genügt ein Rechner mit mehreren virtuellen Maschinen auf denen die Applikation getestet werden kann oder darüber Fehler behoben werden können.
- **Cluster-Simulation:** Eine weitere Möglichkeit zum Einsatz von virtuellen Maschinen eines (homogenen) Clusters. Anstatt nun wieder jeden Rechner als Hardware zur Verfügung zu haben, werden stattdessen gleichartige virtuelle Maschinen auf einem Rechner eingerichtet. Darüber ist nun die Simulation des gewünschten Clusters möglich.

### 3.3.3 „Everything as a Service“-Konzept

Diese Trennung von bereitstellbaren Ressourcen von den tatsächlich physikalisch vorhandenen Ressourcen stellt nun die Basis für ein Konzept, das als „*Everything as a Service*“ beschrieben wird [59]. Durch diese Verfügbarmachung von Ressourcen als Services wird erst Cloud Computing ermöglicht. Für den Anwender der Cloud ist nur deren Schnittstelle zu den Diensten sichtbar, im Hintergrund existiert ein virtualisiertes Grid. So können im Grid einige wenige Superrechner vorhanden sein, auf denen Tausende virtuelle Server laufen, es können aber auch eine Vielzahl haushaltsüblicher Rechner stehen, auf denen die gleiche Anzahl an virtuellen Servern laufen kann. Dies ist für den Anwender des Grids aber unerheblich, da er nur die Schnittstelle zur Cloud kennenlernt / kennenlernen muss. Letztlich können dadurch auch physikalischen Ressourcen innerhalb des Grids ausgetauscht werden, ohne dass sich für den Anwender Änderungen in der Programmierung von Software, die in der Cloud laufen soll, ergeben.

Die wichtigsten Beispiele für das „*Everything as a Service*“-Konzept sind folgende [59]:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Communication as a Service (CaaS) und
- Software as a Service (SaaS).

Speziell die ersten beiden Services werden mit dem Cloud Computing assoziiert. Doch auch die anderen beiden Punkte können durch eine Cloud bereitgestellt werden, bieten aber für den professionellen Anwender weniger Freiheiten als die ersten beiden.

Im Wesentlichen unterscheiden sich die Möglichkeiten im „*Everything as a Service*“-Konzept darin, was und wie umfangreich etwas als Service bereitgestellt wird. Je mehr bereitgestellt wird, um desto weniger muss man sich kümmern, hat aber natürlich gleichzeitig auch weniger Freiheiten, wofür man es nutzen möchten.

Die oben angeführten Beispiele lassen sich hierbei in folgende Reihenfolge bringen. Am wenigsten wird beim „Infrastructure as a Service“ bereitgestellt. Hierbei handelt es sich im Großen und Ganzen nur um bereitgestellte physikalische Ressourcen, wie auch an dem in der Vergangenheit verwendeten Begriff des „Hardware as a Service“ für diese Leistungen erkennbar ist.

Beim „Platform as a Service“ wird eine vollständige Plattform bereitgestellt, d.h. neben den physikalischen Ressourcen auch Betriebssystem, Entwicklungsumgebungen u.ä., also alles um auf dieser Plattform Software entwickeln zu können.

Beim „Software as a Service“ wird bereits diese Software bereitgestellt. Dies können CRM-Applikationen, ERP-Software oder auch Office-Lösungen sein. Hierbei ist die gesamte Bandbreite an möglicher Software denkbar. Ein Spezialfall stellt darin die Bereitsstellung von Kommunikationslösungen, wie Voice over IP, Instant Messaging oder Videokonferenzen dar. Daher kann anstelle von „Communication as a Service“ synonym auch „Unified communications“ as SaaS verwendet werden.

Im folgenden Kapitel sollen nun kommerzielle Cloud-Umgebungen dargestellt werden, die auf diesen Konzepten beruhen.

## 4 Kommerzielle Cloud Umgebungen

Nachdem im vorhergehenden Kapitel das theoretische Konzept hinter einer Cloud beschrieben wurde, sollen nun bereits kommerziell verfügbare Cloud Umgebungen präsentiert werden. Hierzu wurde eine Einschränkung auf die folgenden Anbieter vorgenommen:

- Amazon,
- Google,
- IBM und
- Microsoft.

Hierbei soll im speziellen herausgearbeitet werden, worin die Unterschiede in den Angeboten liegen und ob sich aufgrund des Hauptgeschäftsfeldes der Unternehmen auch eine unterschiedliche Betonung in den angebotenen Dienstleistungen ergibt. Dies erfolgt in Hinblick auf die in Kapitel 3.2.2 beschriebenen Vorteile aus Sicht eines Cloud Providers.

## **4.1 Amazon EC2**

Amazon vertreibt seine Server-Rechenleistung unter dem Namen Elastic Compute Cloud (EC2). Amazon hebt als die wichtigsten Eigenschaften folgende hervor [46]:

- Elastizität: Bereitstellung von veränderter Kapazität innerhalb von Minuten
- Komplette Kontrolle, bis auf Root-Level
- Ausgelegt für die Zusammenarbeit mit anderen Amazon Web Services
- Verlässlichkeit (99,95% Verfügbarkeit im SLA vereinbart)
- Funktionalitäten für die Erstellung fehlerresistenter Anwendungen (z.B. über persistente Speicherung mittels sogenannten „Elastic Block Stores“, mehreren Rechenzentrum weltweit verteilt, variabel zuordenbaren IP-Adressen)
- Sicherheit durch Konfigurationsmöglichkeiten für Firewalls
- Günstige Bepreisung

Diese Aussagen von Amazon sollen nun näher erläutert werden und sollen - soweit dies sinnvoll ist - auch kritisch betrachtet werden.

Als erster Punkt ist erwähnenswert, dass Amazon explizit eine Trennung zwischen der CPU-Leistung (dies ist EC2) und Speicherleistungen vornimmt, siehe hierzu auch die nachfolgende Grafik [47].



Abbildung 4.1.: Simple Storage Service / Elastic Compute Cloud.

Quelle: [47]

Für die Speicherleistungen gibt es hierzu ein eigenes Angebot namens Simple Storage Service (S3). Dieses kann nicht nur zur Speicherung von Daten dienen und unabhängig von EC2 genutzt werden, sondern wird für die Cloud zur Sicherung von Virtual Machine Images verwendet. Diese werden von Amazon als Amazon Machine Image (AMI) bezeichnet und enthält die Anwendung, Bibliotheken, Daten, Konfigurationen usw. zur Verwendung innerhalb der EC2. Solange die Server-Leistung nicht benötigt wird, liegt dieses AMI gesichert innerhalb des S3 vor. Erst wenn die Rechnerleistung benötigt wird, muss dieses Image auf den Server geladen werden und auch erst dann wird die CPU-Leistung verrechnet.

Neben den beiden Dienstleistungen EC2 und S3 bietet Amazon auch weitere Infrastrukturservices an, wie z.B. SimpleDB zur Abfrage auf strukturierte Daten, CloudFront zur Verteilung von Inhalten. Diese Dienstleistungen stellen aber im Grunde logische Erweiterungen zu EC2 dar bzw. wären aufgrund der Definition einer Cloud aus Kapitel 3 bereits integrale Bestandteile ebendieser und wurden nur aus marketingtechnischen Gründen als eigene Dienstleistungen aus dem EC2-Angebot herausgelöst.

Nachdem Amazon als wesentlichen Vorteil ihres Angebots die Zusammenarbeit mit anderen Amazon Web Services hervor streicht, ist es hingegen interessanter diese Dienstleistungen zu betrachten:

- Amazon Flexible Payments Service bzw. Amazon DevPay als Möglichkeit Bezahlungen abzuwickeln
- Alexa Web Services, die zur Sammlung von Informationen verwendet und somit von Marketing-Abteilungen zur Analyse der Website-Besuche verwendet werden kann
- Amazon Fulfillment Web Service zur Lieferung von durch den Nutzer angebotenen Produkten über die Logistik von Amazon
- Amazon Associates Web Service als Möglichkeit für den Nutzer Provisionserträge über den Verkauf von durch Amazon angebotenen Produkten zu erzielen.

Wie man an dieser Liste unschwer erkennen kann, setzt Amazon auf sein Hauptgeschäft des Versandhandels auf und bietet nicht nur eine Cloud an, sondern spezielle Leistungen, die für andere Versandhändler interessant sein könnten. Daher könnte für ein Unternehmen, dass an und für sich bezüglich der Nutzung einer Cloud indifferent zwischen unterschiedlichen Anbietern ist, interessant werden trotzdem Amazon zu wählen, wenn sie eine der anderen Web Services nutzen möchten.

Aber auch für Amazon ergeben sich Vorteile, so mussten die Funktionalitäten für die Bezahlung, die Logistik-Kette usw. bereits für das Hauptgeschäft aufgebaut werden.

Diese Investitionen können durch die Bereitstellung als Service besser genutzt werden (siehe hierzu auch Kapitel 3.2.2, Punkt 2), weil sie nun nicht mehr durch Amazon selbst eingesetzt werden, sondern auch durch andere Versandhändler. Aber auch die anderen Vorteile können für Amazon relevant werden, hierbei sei speziell auf den Amazon Marketplace verwiesen. Anstatt andere Unternehmen als Konkurrenten anzusehen, werden diese in das Angebot integriert, wodurch diese die Bekanntheit von Amazon nützen können, während Amazon wiederum Vorteile aus einem breiteren Angebot bzw. durch die Verfügbarmachung von Nischenprodukten ziehen kann.

#### **4.1.1 Vorgehensweise für die Nutzung von EC2**

Im folgenden Abschnitt soll nun beschrieben werden, wie eine Applikation in der Cloud von Amazon genutzt werden kann. Hierzu wird einerseits eine Beschreibung gegeben, wie der Prozess funktioniert, andererseits die Möglichkeiten für die Nutzung beschrieben.

Der Prozess besteht grundsätzlich aus folgenden vier Schritten (siehe [55]):

1. Erzeugung des virtuellen Images (Amazon Machine Image), Amazon beschreibt dies in der Vorgehensweise als „Packen“ der Festplatte – es enthält also Betriebssystem, Programme, Konfigurationen usw.,
2. Speicherung des AMI im Simple Storage Service, damit ein Zugriff durch die EC2 erfolgen kann,
3. Registrierung des AMI mit Amazon EC2 – nach einer Verifizierung der Inhalte auf Funktionsfähigkeit erhält das AMI eine eindeutige Identifikationsnummer.

4. mit Hilfe der Identifikationsnummer des AMI kann nun eine oder mehrere Instanzen desselbigen gestartet, überprüft und beendet werden. Hierzu gibt es unterschiedliche Tools auf Kommandozeilenebene, als Java-Bibliotheken, SOAP oder *Query-based APIs*.

Im folgenden Abschnitt werden nun diese Schritte näher beschrieben.

#### **4.1.1.1 Erzeugung des Amazon Machine Images**

Die einfachste Möglichkeit ein AMI zu erzeugen ist die Verwendung eines bestehenden AMI, das öffentlich zugänglich ist. Dieses kann dann den eigenen Bedürfnissen angepasst werden. Alternativ kann auch von Null auf ein eigenes AMI aufgebaut werden.

Für beide Varianten muss zuerst eine Entscheidung über das verwendete Betriebssystem getroffen werden, hierbei gibt es die Wahlmöglichkeit zwischen Windows, Linux und Unix, wobei folgende angeboten werden:

- Windows Server 2003,
- Red Hat Enterprise Linux, Oracle Enterprise Linux, openSUSE Linux, Ubuntu Linux, Fedora, Gentoo Linux und Devian als Linux-Betriebssysteme und
- OpenSolaris als Unix-System.

Diese Betriebssysteme können dann zur Installation von Software genutzt werden, hierbei können u.a. folgende Programme genutzt werden:

- als Datenbanken: IBM DB2, IBM Informix Dynamic Server, MySQL Enterprise, Oracle 11g, Microsoft SQL Server Standard 2005,
- für die Batch-Verarbeitung: Hadoop, Condor, Open MPI,
- für das Web Hosting: Apache HTTP, IIS/Asp.Net, IBM Lotus Web Content Management, IBM WebSphere Portal Server,
- als Application Development Environments: Java Application Server, Jboss Enterprise Application Platform, IBM sMash, Ruby on Rails, Oracle WebLogic Server bzw.
- als Video Encoding & Streaming-Software: Wowza Media Server Pro, Windows Media Server.

Dies stellt aber nur einen Auszug aus den möglichen Anwendungen dar. Interessant hierbei ist, dass nicht nur die Cloud User selbst ihr AMI zusammenstellen können, sondern dies auch durch Amazon-Partner, z.B. ein Software-Anbieter, erfolgt. Dadurch erspart sich der Cloud User den Aufwand für die Zusammenstellung der Software, sondern wählt unter den verschiedenen AMIs dasjenige aus, das am besten seinen Zwecken entspricht und erhält dadurch ein abgestimmtes Paket (z.B. Betriebssystem, Web Server, Datenbank, Entwicklungsumgebung, die bereits aufeinander abgestimmt sind). Diese AMIs sind entweder kostenlos verfügbar oder sind kostenpflichtig.

Bei der Erzeugung eines AMI wird nicht unterschieden, ob dieses für Produktions- oder Testzwecke eingesetzt werden soll, insofern kann es sich als sinnvoll herausstellen für die Entwicklung das gleiche Betriebssystem und die gleiche Software auch auf einem lokalen Rechner zu installieren, womit aber eventuell Vorteile einer durchgehend konzipierten Entwicklungsumgebung verlorengehen können.

#### 4.1.2 Bepreisung der Cloud-Nutzung

Das Preissystem von Amazon ist relativ kompliziert. Die Kosten hängen sowohl vom Ort der Server-Standortes, der Vertragslaufzeit, der Serverauslastung, der genützten Hardware und dem Betriebssystem ab.

Daher sollen hier nur einige Preisbeispiele mit Stand März 2010 dargestellt werden [62]:

- der Standort kann in Nord-Virginia, Kalifornien oder Irland sein, für die Default-Hardwarekonfiguration ohne Reservierung des Servers betragen die Kosten für einen Linux-Server 0,085 USD pro CPU-Stunde in Nord-Virginia, während eine Stunde in Kalifornien und Irland mit 0,095 USD etwas teurer ist, bei einem Windows-Rechner hingegen sind die Preise in Nord-Virginia und Irland gleich und betragen 0,12 USD pro CPU-Stunde, während Kalifornien wiederum mit 0,13 USD etwas teurer ist.
- ein weiterer Einflussfaktor ist die Vertragslaufzeit: entweder kann der Server *On-Demand* nachgefragt werden oder es kann ein Server reserviert werden und ein Vertrag mit einer Laufzeit von einem bzw. drei Jahren über diese Reservierung abgeschlossen werden. In diesem Fall ist eine einmalige Gebühr fällig, die für die Default-Konfiguration bei Laufzeit über einem Jahr 227,50 USD beträgt, bei drei Jahren 350 USD (die dann auch nur alle drei Jahre fällig ist).

Der Vorteil bei einer Reservierung ist, neben technischen, dass die Kosten pro CPU-Stunde geringer sind. Wiederum für die Default-Konfiguration betragen die Kosten nun 0,03 USD (Linux) und 0,05 USD (Windows) in Nord-Virginia bzw. 0,04 USD (Linux) und 0,06 USD (Windows) in Kalifornien und Irland (wobei an diesen beiden Server-Standorten die Kosten für jedes Betriebssystem gleich sind, und sich nicht wie zuvor bei der On-Demand-Nachfrage unterscheiden).

- als weitere Möglichkeit können sogenannte „*Spot Instances*“ genutzt werden, bei denen um ungenützte Server-Ressourcen geboten werden kann. Erst wenn Ressourcen aufgrund geringer Serverauslastung verfügbar sind, läuft auch die Anwendung, diese dann aber zu günstigeren Kosten. Diese Kosten ändern sich regelmäßig in Abhängigkeit der verfügbaren freien Server-Ressourcen, in Nord-Virginia betragen zum Zeitpunkt der Untersuchung die Preise dann nur ca. ein Drittel der normalen „On-Demand“-Kosten, in Kalifornien und Irland knapp über die Hälfte.
- zusätzlich hängen die Kosten auch von der genutzten Hardware ab, hierbei wird Standardrechnern in den Varianten S, L, XL, Rechnern mit zusätzlichem Speicher (XL, 2x XL, 4x XL) und Rechnern mit höherer CPU-Leistung (M, XL) unterschieden, am teuersten sind hierbei Rechner mit vierfacher Speicherleistung (Kosten in Irland mit Windows bei 2,88 USD pro CPU-Stunde, im Vergleich zu XL-CPU-Leistung von 1,16 USD pro CPU-Stunde und den Standardrechner in einer Bandbreite von 0,12 USD pro CPU-Stunde (Small) bis 0,96 USD (XL).
- als letzter Faktor hängen die Kosten noch vom Betriebssystem ab: Preisbeispiele wurden bereits oben angeführt, wobei Windows aufgrund der zusätzlichen Lizenzierungskosten in allen Variationen die teurere Möglichkeit ist.

Damit aber noch nicht genug, fallen noch weitere Kosten an [62]:

- für den Datentransfer: In-Coming ab 1. Juli 2010 bei 0,10 USD pro GB, Out-Going 0,15 USD pro GB für die ersten 10 TB pro Monat, danach je nach Menge weniger, bis zu 0,08 USD pro GB bei über 150 TB pro Monat. Weiters teilt Amazon ihre eigenen Server noch in sogenannte „Availability Zones“ ein (entspricht in etwa den möglichen Server-Standorten), sollten zwischen diesen Zonen ein Datentransfer stattfinden, z.B. wegen eines automatischen Load Balancings, fallen Kosten von 0,01 USD pro GB an.
- für die Datenspeicherung: in Irland 0,11 USD pro GB und 0,11 USD pro einer Million I/O-Anfragen, für Snapshots innerhalb der EC2 zu S3 0,15 USD pro GB, 0,01 USD pro 1.000 Put-Anfragen (bei der Speicherung) und 0,01 USD pro 10.000 GET-Anfragen (beim Laden eines Snapshots).
- für die IP-Adresse: 0,01 USD pro elastischer IP-Adresse und Stunde und 0,10 USD für das Remapping der IP-Adresse (wobei die ersten Hundert pro Monat gratis sind).
- für das Administrations-Tool CloudWatch: 0,015 USD pro Stunde.
- für das Load-Balancing: 0,028 USD pro Load Balancer-Stunde und 0,008 USD pro verarbeiteten GB.

Zum Glück ist zumindest die Auto-Skalierung innerhalb von CloudWatch gebührenfrei. Über die Gründe für dieses komplizierte System lässt sich nur spekulieren, bei einigen Punkten ist dies noch verständlich (Windows-Lizenzierungskosten, aber auch der höhere Preis in Kalifornien, da dort die Nachfrage nach Rechnerkapazitäten höher als in Nord-Virginia sein wird und somit nach volkswirtschaftlicher Theorie sich auch ein höherer Preis ergeben sollte), einige Punkte wie die „Spot“-Instanzen sind sicherlich ebenfalls sinnvoll, um eine gleichmäßigere Nutzung der Server zu gewährleisten, aber insgesamt gesehen ist das Preissystem doch zu kompliziert.

## ***4.2 Google App Engine***

Google bietet seine Cloud unter dem Namen „App Engine“ zur Nutzung an. Gegenüber dem Angebot von Amazon und auch den beiden nachfolgenden Ausgestaltungen unterscheidet sich das Angebot von Google durch die Einschränkung auf Web-Applikationen. Während man bei Amazon sehr viele Entscheidungsfreiheiten für den Betrieb von Applikationen in der Cloud hat – sogar eine Auswahlmöglichkeit zwischen verschiedenen Betriebssystemen hat –, schränkt Google die Art der Anwendung auf Java-Applikationen ein. Somit hat man weder ein Mitspracherecht bezüglich des Betriebssystems oder alternative Entwicklungsumgebungen, noch kann man Funktionalitäten nutzen, die von der JVM nicht zur Verfügung gestellt werden (z.B. Funktionen, die einen Zugriff auf Filesystem-Ebene bedürfen würden).

Laut Google bietet die *App Engine* folgende Funktionalitäten [51]:

- Darstellung von dynamischen Web-Sites mit allen gebräuchlichen Web-Technologien,
- Dauerhafte Speicherung mit der Möglichkeit für Anfragen, Sortierung und Transaktionen,
- Automatische Skalierung und Load-Balancing,
- Programminterfaces für die Authentifizierung von Anwendern und das Versenden von Mail,
- Entwicklungsumgebung, die lokal installiert werden kann und die produktive Umgebung der App Engine simuliert,
- Funktion für die Einplanung von Jobs, die zu bestimmten Zeitpunkten oder regelmäßigen Intervallen laufen.

Wie man anhand der Funktionsbeschreibung erkennen kann, sind somit viele Anwendungen im Web-Bereich abdeckbar, aber auch wenn die dauerhafte Speicherung erwähnt wird, ist damit z.B. nicht die Verwendung von relationalen Datenbanken gemeint, sondern dieser *Datastore* verwendet einen objektorientierten Ansatz und nutzt die Funktionalitäten von Java zur Persistierung von Objekten. Auch wenn dies für viele Anwendungen ausreichend sein wird, erkennt man doch, dass das Angebot eher auf einen semi-professionellen Bereich abzielt, als auf größere Applikationen. So lassen sich aufgrund der Einschränkungen aber keine auf Datenbanken basierenden Applikationen betreiben und dies sind inzwischen sehr viele, wobei z.B. auch schon KMU ihre Buchhaltungs- und CRM-Systeme mittels relationalen Datenbanken betreiben.

## 4.2.1 Bepreisung der App Engine

Im Vergleich zu Amazon sind die Kosten für die Nutzung recht simpel gestaltet. Hierbei sind 500 MB Speicher und 5 Mio. Seitenaufrufe (inkl. den damit verbundenem Datentransfer) überhaupt einmal frei. Für die zusätzliche Nutzung fallen folgende Kosten an [63]:

- Datentransfer: Outgoing 0,12 USD pro GB, Incoming 0,10 USD pro GB
- CPU-Zeit: 0,10 USD pro Stunde
- Speicherplatz: 0,15 USD pro GB und Monat
- E-Mail Versand: 0,0001 USD pro Empfänger

Wie auch noch in Kapitel 6 ersichtlich werden wird, ist für kleinere Anwendungen auf jeden Fall Google die günstigere Lösung. Speziell kleinere Unternehmen bzw. Vereine werden die freien Seitenaufrufe und Speicherplatz nicht überschreiten (besonders da man statische Teile auch bei einem anderen Anbieter hosten könnte, und nur die dynamischen Teile der Website über die AppEngine laufen müsste).

Amazon hat nur dann Kostenvorteile, wenn man die Reservierungsvariante wählt, somit wäre deren Lösung für mittlere (Web-)Anwendungen eventuell die beste Lösung. Für Großunternehmen wird es wiederum schon wieder interessant sich direkt von einem IT-Berater ein spezifisches Cloud-Konzept für ihre spezielle Situation anbieten zu lassen, hierbei wäre ein möglicher Anbieter IBM.

### **4.3 IBM Blue Cloud**

IBM selbst bietet zwar auch selbst Cloud-Leistungen an, aber bei ihrem Angebot sieht man einen dritten Aspekt bezüglich der Herangehensweise der einzelnen Anbietern. IBM sieht sich mehr als „*Technology Enabler*“ denn als Anbieter von Cloud. Dies erkennt man daran, dass sie bereits in ihrem Überblick ihres Angebots [52] explizit ihre Beteiligung an der Entwicklung der „*Amazon Machine Images (AMI)*“ zur Verwendung in der EC2 (siehe Kapitel 4.1) hinweisen. Weiters sind sie an der Erstellung des „*Open Cloud Manifestos*“ wesentlich beteiligt.

Hierfür bietet IBM sowohl Leistungen im Bereich der Infrastruktur an, diese unterteilen sich nach folgenden Punkten:

- Dynamische Infrastruktur,
- Service Management,
- *Service Oriented Architecture*,
- *Information On Demand*,

als auch eigenständige Services und Applikationen, wie LotusLive (zur Zusammenarbeit unter Personen), *Computing On Demand* oder Services zur cloud-basierten Speicherung an.

Der Unterschied zu den anderen Anbietern stellt sich darin dar, dass jede dieser Leistungen relativ unabhängig voneinander gekauft werden können. Während z.B. bei Google die Nutzung der App Engine auch eine Nutzung des Google-Rechenzentrums voraussetzt, kann bei IBM ein Kunde auch nur einzelne Software-Pakete kaufen und z.B. eine eigene Cloud in seinem Rechenzentrum betreiben. Daher stellt IBM auch keine Standard-Preislisten zur Verfügung, da für jede Nachfrage ein individuelles Angebot, zugeschnitten auf die spezifische Kundensituation, gestellt wird.

So ist aber auch die Angebotspalette teilweise nur eine Zusammenfassung der bisherigen Aktivitäten und Angeboten von IBM, die bereits heute Rechenzentrum weltweit beliefern. Es kann aber alternativ auch die Cloud als Service in einem IBM-eigenen Rechenzentrum genutzt werden. Hierbei ist IBM der derzeit weltweit größte Anbieter von Cloud-Rechenzentren – IBM nennt diese *Cloud Labs* -, u.a. in Dublin, Peking, Johannesburg, Tokio, Bangalore und natürlich in Silicon Valley.

## 4.4 Microsoft Azure

Microsoft benennt seine Cloud-Umgebung Azure und geht mit seinen Ansatz weiter als Google, schränkt aber mehr ein als Amazon oder IBM.

Azure selbst ist eine Plattform, die als Basis für darauf basierende Applikationen dient. Dies lässt sich anhand folgenden Schaubilds illustrieren:



Abbildung 4.2.: Azure Services Platform.

Quelle: [53]

Die auf Azure aufsetzten Services sind optional und können auch durch Anwendungen von Fremdanbietern ersetzt werden, Microsoft erwähnt hier Eclipse, Ruby, PHP und Python [53].

Somit ist Microsoft nur einschränkend in Bezug auf das Betriebssystem – Microsoft bezeichnet Azure als „*Cloud Services Operating System*“ –, die aber auf Windows als zugrundeliegendem Betriebssystem aufsetzt (und nicht auf Linux oder Unix-Derivate).

Im Gegensatz zu Google ist aber nicht nur die Microsoft-eigene Entwicklungssprache bzw. die Entwicklung über Visual Studio möglich, sondern auch Fremdanbieter – und nicht nur über die JVM wie bei Google.

Je nach Bedarf können aber auch weitere Dienstleistungen in Anspruch genommen werden, neben Services wie die .NET Services, SQL Services, auch vollständige Applikationen wie SharePoint oder die Microsoft CRM-Applikation. Und darin sieht man meiner Meinung nach schon die wichtigere Zielgruppe der Cloud-Aktivitäten von Microsoft: nämlich mehr Klein- und Mittelbetriebe, die Geschäftsanwendungen wie CRM oder einen Mail-Server benötigen, aber nicht den Willen oder die Kapazität haben, diese selbst zu betreiben. Und damit auch im Sinne der in Kapitel 3 beschriebenen Vorteile für Cloud-Anbieter auch die Hebung von zusätzlichen Kundengruppen für die bestehenden Microsoft-Applikationen.

#### 4.4.1 Bepreisung von Windows Azure

Derzeit ist Windows Azure noch in einem Technology Preview, wo die Leistungen noch nicht verrechnet werden, sondern nur Limits für die Nutzung gesetzt sind.

Mögliche Preisindikationen (Stand Juli 2009) finden sich in einer ähnlichen Bandbreite wie die Kosten der Google AppEngine [64]:

- Rechenzeit: 0,12 USD pro Stunde,
- Speicher: 0,15 USD pro GB (Speichertransaktionen 0,01 USD pro 10.000),
- Datentransfer: für Incoming 0,10 USD pro GB und für Outgoing 0,15 USD pro GB,
- Datenbank: für die SQL Azure Web Edition 9,99 USD pro Monat (bis 1 GB große Datenbank), für die Business Edition (bis 10 GB große Datenbank) 99,99 USD pro Monat.

## 5 Anwendungen von Cloud Computing

Grundsätzlich können in einer Cloud alle verschiedenen Applikationen laufen, angefangen von einer simplen Software zur Erfassung von Messdaten bis hin zu komplexer mathematischer Software zur Analyse verschiedener Probleme. Daher soll in diesem Kapitel keine umfassende Aufzählung aller Möglichkeiten erfolgen, sondern es werden Beispiele dargestellt, wie Unternehmen eine Cloud derzeit verwenden und welche Vorgehensweisen von IT-Managern als sinnvoll erachtet werden.

In einer Studie von Forrester („TechRadar for Infrastructure & Operations Professionals: Cloud Computing, Q3/2009“) werden elf verschiedene Cloud-Technologien betrachtet. Dies sind folgende [60]:

1. Business Process Management-as-a-Service: Internetbasierte Anwendung, mittels deren geschäftsrelevante Prozesse selbständig durchgeführt werden können
2. Cloud Billing: Abwicklung von Bezahl-Prozessen über die Cloud, damit diese z.B. auf der Firmen-Website zur Verfügung gestellt werden können
3. Cloud Labs: als eine Art “Internet-as-a-Service“-Lösung für die Entwicklung und den Test von Applikationen
4. Database-as-a-Service: Bereitstellung von Datenbanken durch die Cloud und Verwendung sowohl für Testzwecke bei der Applikationsentwicklung, für Internet-Applikationen, aber auch als Archiv oder Backup von außerhalb der Cloud befindlichen Datenbanken

5. Desktop-as-a-Service: Bereitstellung von virtuellen Desktops, entweder als vollständiger Ersatz (wenn die Möglichkeit besteht alle benötigten Applikationen auf diese Weise zur Verfügung zu stellen) oder als alternativer Desktop
6. Disaster Recovery-as-a-Service: Wiederherstellung, speziell für Server, bei Schadensfällen, Verwendung bei geschäftskritischen Applikationen
7. Infrastructure-as-a-Service: Bereitstellung von Infrastruktur als Service, für den Einsatz zu verschiedensten Zwecken wie Web-Applikationen, Collaboration und auch Applikationstests
8. Integration-as-a-Service: Bereitstellung von Diensten zur Kommunikation zwischen verschiedenen Web-Diensten, daher auch notwendig zur Integration von SaaS-Applikationen
9. Platform-as-a-Service: zur Entwicklung und produktiven Verwendung von Web- oder SOA-Applikationen
10. Software-as-a-Service: Angebot einer Applikation über die Cloud, eventuell auch als On-Demand-Alternative für traditionelle Software.
11. Storage-as-a-Service: Anwendung der Cloud zur Speicherung von Daten, die entweder von Internet-basierten Applikationen stammen oder von anderen Anwendungen in der Cloud

Mit Ausnahme von IaaS und SaaS sieht Forrester für diese Anwendungen in der Cloud nur minimalen oder mäßigen Erfolg und grundsätzlich sehr wenig bzw. manchmal sogar negativen Nutzen. Abgesehen davon, dass die Auswahl der Anwendungen durch Forrester als sehr selektiv und komisch angesehen werden kann – einerseits wird eine sehr konkrete Applikation, wie Cloud Billing, genannt, andererseits werden sehr verallgemeinernde Begriffe wie IaaS, PaaS, SaaS als konkrete Anwendungen dargestellt -, zeigt diese Studie doch deutlich, dass in der IT-Branche derzeit noch nicht ganz klar ist, was die konkreten Vorteile für die Nutzung einer Cloud im Vergleich zu traditionellen Rechenzentrum und Internet-Applikationen ist. Der zweite Kritikpunkt wäre noch, dass nicht auf die speziellen Vorteile einer Cloud eingegangen wird. Wie in den letzten Kapiteln beschrieben wurde, ist ein Vorteil der Nutzung einer Cloud (bzw. auch Grids), dass zusätzliche Ressourcen jederzeit genutzt werden können, aber nicht ständig verfügbar sein müssen. So wären sinnvollere Anwendungen in einer Cloud, diejenigen die entweder zu Spitzenzeiten benötigt werden oder nur an speziellen Tagen im Monat, z.B. jederzeit zuschaltbare Server bei einer Vervielfachung von Website-Zugriffen oder Lösungen für monatliches Reporting. Dieser Aspekt kommt aber in der Forrester-Studie nicht vor.

Trotz dieser Problematik der Forrester-Studie, lässt sich aber auch in anderen Quellen wiederfinden, dass sich die Anwendungsmöglichkeiten noch nicht klar herauskristallisiert haben. So sieht auch die Gartner Group Clouds in ihrem „Hype Cycle 2009“ noch im „Gipfel der überzogenen Erwartungen“ [61]:

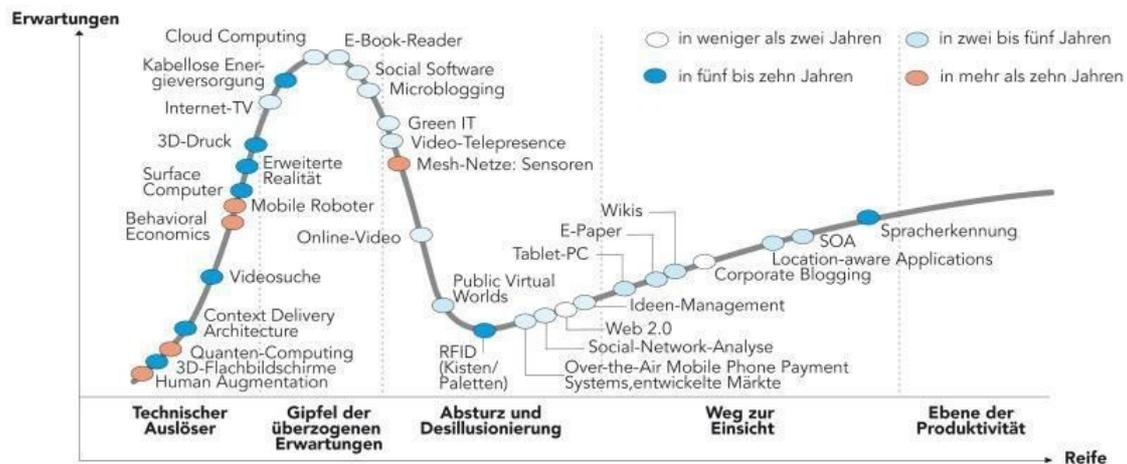


Abbildung 5.1.: Gartner Hype Cycle 2009.

Quelle: [61]

So verwenden, laut einer Studie von Avanade, Unternehmen in den meisten westeuropäischen Staaten Clouds zu ca. zwei Dritteln nur für Storage-Anwendungen [61], obwohl man wie soeben beschrieben den Hauptnutzen von Clouds woanders sehen könnte (Daten müssen dauerhaft gespeichert sein, somit wird dieser angemiete Speicherplatz auch jederzeit und nicht nur zu Spitzenzeiten genützt – daher sollte eigentlich auch der dauerhafte Ankauf von Speicherplatz die bessere Alternative darstellen). Im konkreten wären die Prozentsätze für Deutschland 63%, für Schweiz und Italien 67%, die Niederlande 60%, während er in den USA bei 56% liegt. Man könnte daher auch vermuten, dass je weiter fortgeschritten ein Land in der Nutzung von Clouds ist, desto mehr andere Anwendungen werden genützt und desto niedriger ist damit automatisch der Prozentsatz für Storage-Anwendungen. Die USA wäre somit am fortgeschrittensten und dort werden Clouds bereits in stärkerem Ausmaß für andere Applikationen verwendet, was plausibel erscheint.

Als zweithäufigste Applikation in der Cloud werden in der Avanade-Studie [61] die Bereitstellung von Collaboration-Anwendungen wie Online-Kommunikation oder Applikationen für den Datenaustausch. Auch dies eigentlich keine „logische“ Cloud-Anwendung. Eine mögliche Erklärung dafür ist, dass sich solange sich die Anwender noch nicht im Klaren sind, wozu Clouds am besten verwendbar sind, der Markt noch stärker Anbieter-getrieben ist. Derzeit werden von den Cloud-Anbieter aber gerade derartige „allgemeine“ Dienste angeboten – sei es Google Docs, LotusLive von IBM, die CRM-Applikation von Salesfoce.com, die Collaboration- oder Office Live-Anwendungen von Microsoft. Nicht aber unternehmens- oder branchenspezifische Anwendungen vorherrschen würden.

Im abschließenden Kapitel wird nun noch ein Experiment mit der Cloud von Google dargestellt.

## 6 Experiment Google App Engine

### ***6.1 Test: Hello World!***

Als erster Versuch zum Test der Google App Engine soll das berühmte „Hello World“-Beispiel verwendet werden.

Wie bereits im vorhergehenden Kapitel beschrieben, könnte eine Umsetzung in Python oder in Java erfolgen. Zum derzeitigen Zeitpunkt ist Java aber nur als „Early Draft“-Version verfügbar, daher wurde Python für das Experiment verwendet.

Vor der Verwendung der App Engine wird zuerst einmal die Programmiersprache selbst benötigt, diese kann von [python.org](http://python.org) heruntergeladen werden. Obwohl bereits Version 3.0 verfügbar wäre, setzt die App Engine auf die Version 2.5.3 auf. Hierbei enthält das Python Software Development Kit (SDK) einen Interpreter, der die Programmausführung ermöglicht. Im Gegensatz zu Java ist die Verwendung noch einfacher gestaltet - es wird der Code direkt interpretiert und kein eigener Bytecode erstellt.

Nach der Installation der Programmiersprache kann das Google App Engine SDK heruntergeladen und installiert werden. Dieses enthält nützliche Tools zum Test und Deployment der Applikation. Die Erstellung der Dateien kann im einfachsten Fall über einen Texteditor erfolgen. Es gibt aber auch eine eigene IDE für Python, alternativ können aber auch beliebige Entwicklungsumgebungen wie z.B. Eclipse zur Programmierung verwendet werden. Eine Entscheidung darüber wird in der Praxis von der Komplexität des Projekts abhängen.

So wird es, wenn z.B. im Team gearbeitet werden soll oder eine Versionsverwaltung eingesetzt werden soll, kein Weg um eine professionelle Entwicklungsumgebung geben. In unserem einfachen Experiment ist für die Erstellung der Dateien hingegen der Texteditor ausreichend.

Sobald das App Engine SDK installiert wurde, kann eine Anwendung entwickelt werden. Hierzu legt man ein Unterverzeichnis mit dem Programmcode an. Standardmäßig wird das SDK im Verzeichnis „C:\Programme\Google\google\_appengine“ installiert. Darunter wurde das Unterverzeichnis „cloud\helloworld“ angelegt. In diesem werden die benötigten Dateien gespeichert.

### 6.1.1 Applikationsentwicklung

Unsere erste Test-Applikation ist sehr einfach gestaltet, es werden folgende Bestandteile benötigt.

Als erste Datei wird das Programm benötigt. Dieses bezeichnen wir als helloworld mit der Dateiendung „py“, was die Abkürzung für Python ist.

Dieses Python File **helloworld.py** enthält folgenden Inhalt [56]:

```
print 'Content-Type: text/plain'  
print "  
print 'Hello, world!'
```

Als zweite Datei wird **app.yaml** als Konfigurationsfile benötigt.

Hierzu wird eine einfache Definition verwendet, die auf YAML [57] aufsetzt. Hierbei steht YAML für „YAML Ain't Markup Language“, dies ist ein Standard für eine benutzerfreundliche Daten-Serialisierung für alle Programmiersprachen.

Das File enthält folgenden Code [56]:

```
application: helloworld
version: 1
runtime: python
api_version: 1

handlers:
- url: /*
  script: helloworld.py
```

Hier eine kurze Beschreibung der einzelnen Zeilen [56]:

- Der Applikation-Identifizier ist *helloworld*. Um im letzten Schritt die Applikation bei Google App Engine zu registrieren, wird ein eindeutiger Identifizier benötigt.
- Die zweite Zeile identifiziert die Version der Applikation selbst, also die erste.
- Die nächsten beiden Zeilen geben an, welche Ausgestaltung der Google App Engine verwendet wird. In diesem Fall wäre die Runtime Python (diese könnte alternativ auch Java sein), und diese wird in Version 1 eingesetzt. Zukünftig könnten neben Python und Java weitere Runtimes hinzukommen bzw. andere Versionen eingesetzt werden (so könnte zukünftig auch Python 3.0 unterstützt werden,) dies wäre durch eine andere Versionsnummer kenntlich gemacht

Dieser erste Teil des Konfigurationsfiles ist für jede Applikation gleich aufgebaut, der zweite Teil enthält nun unterschiedliche Variablen und Werte abhängig von der Applikation. In unserem einfachen Beispiel werden alle URL-Anfragen auf die Datei **helloworld.py** umgeleitet.

In komplexeren Applikationen könnten diese Definitionen weit umfangreicher sein. So würde der zweite Teil des Konfigurationsfile für eine Anwendung, die über eine Startseite einen Link auf eine Produktseite und eine Kontaktseite zur Verfügung stellt, bereits folgende Zeilen enthalten:

```
handlers:  
- url: /index  
  script: index.py  
- url: /produkte  
  script: produkte.py  
- url: /kontakt  
  script: kontakt.py
```

Es gibt nun bereits drei Python-Skripts, die ebenfalls HTML generieren würden. Je nach Eingabe der URL würde das entsprechende Skript aufgerufen werden.

Damit ist die „Entwicklung“ der HelloWorld-Applikation abgeschlossen.

## 6.1.2 Testen der Applikation

Im Verzeichnis des App Engine SDK findet sich zum Testen ein Entwicklungs-WebServer, der lokal über das Python-Skript **dev\_appserver.py** gestartet werden kann.

Als Parameter muss dem Skript die zu startende Applikation mitgegeben werden. Dies ist unserem Beispiel das Verzeichnis „cloud\helloworld“ mit den Applikationsdateien. Daher musste es als Unterverzeichnis im google\_appengine-Ordner angelegt werden. Aufgerufen in diesem Ordner ergibt sich folgender Befehl:

```
dev_appserver.py cloud\helloworld\
```

Sobald der Web Server gestartet wurde, kann die Applikation lokal über jeden beliebigen Browser getestet werden. Da der Web Server ohne weitere Parameter automatisch derart konfiguriert wird, dass er auf Port 8080 auf Anfragen reagiert, kann die Applikation über die URL <http://localhost:8080/> aufgerufen werden.

Da im Konfigurationsfile alle URLs mit `/*` auf das Skript `helloworld.py` umgeleitet werden, wird dieses ausgeführt und gibt im Browser den Text „Hello, world!“ aus (Abbildung 6.1.):

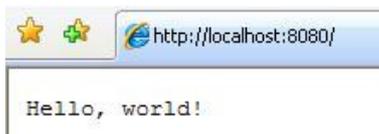


Abbildung 6.1.: Screenshot: Hello World!

Ein Aufruf über <http://localhost:8080/jedebeliebigebezeichnung> würde aufgrund des URL-Handler Mappings `/*` zum gleichen Ergebnis führen.

### 6.1.3 Deployment der Applikation

Für die Produktivschaltung einer Applikation gibt es im App Engine SDK ebenfalls ein Python-Skript, das die vollständige Arbeit des Deployments übernimmt.

Vorab muss aber eine Applikation auf der Appengine-WebSite von Google registriert werden. Hierzu wird nur ein Google Account benötigt, wie man es bereits besitzt, wenn man sich die Startseite von google.com individualisiert oder wenn man Google Mail verwendet.

Im folgenden Screenshot (Abbildung 6.2.) werden die benötigten Eingaben visualisiert, es sind dies nur die zwei folgenden:

- Application Identifier: dies ist eine eindeutige Bezeichnung für die Anwendung, die keine Leerzeichen enthalten darf und – da sie für den Aufruf der erzeugten WebSite verwendet wird – den Namenskonvention für eine URL entsprechen muss. Alle Anwendungen die für die App Engine produktiv geschaltet werden soll, erhalten diese Identifier. Somit sind sie nicht nur aus Sicht des einzelnen Registrars eindeutig, sondern über alle Anwendung, die von jedem der vielen App Engine-Nutzer registriert wurden.
- Application Title: dies ist eine beliebige Bezeichnung, die den Titel der Anwendung darstellt. Wenn man sich für die Anwendung authentifizieren muss, wird dieser auf der Login-Seite dargestellt.

So sieht die Google-Seite für die Registrierung der Anwendung aus:

## Create an Application

### Application Identifier:

.appspot.com

You can map this application to your own domain later. [Learn more](#)

### Application Title:

Displayed when users access your application.

### Authentication Options (Advanced): [Learn more](#)

Google App Engine provides an API for authenticating your users. If you choose not to use this, anyone in the world will be able to access your application. However, if you choose to use this, you'll need to specify now who can sign in to your application:

#### Open to all Google Accounts users (default)

If your application uses authentication, anyone with a valid Google Account may sign in. (This includes all Gmail Accounts, but does \*not\* include accounts on any Google Apps domains.)

[Edit](#)

Abbildung 6.2.: Screenshot: Hello World!

Nach dem Speichern – bei Eindeutigkeit des Identifiers – ist die Applikation bereits registriert.

Bevor mit dem Python-Skript das Deployment erfolgen kann, muss nun noch die Konfigurationsdatei angepasst werden. Damit Google erkennen kann, welche Applikation gemeint ist, muss dies in der Datei angegeben sein. Da als Identifier „dawien2009test1“ gewählt wurde, sieht **app.yaml** nun folgendermaßen aus:

```
application: dawien2009test1
version: 1
runtime: python
api_version: 1

handlers:
- url: /*
  script: helloworld.py
```

Nun wird das Deployment über den Befehl

```
appcfg.py update cloud\helloworld\
```

gestartet.

Aufgrund der Application-Bezeichnung in app.yaml erkennt dieses Skript nun, welche Applikation am Google-Server gemeint ist und aufgrund des Parameters update werden die Dateien am Server mit den lokalen Applikationsdateien überschrieben bzw. falls noch nicht vorhanden dort erstellt.

Nach Ausführung des Befehls erfolgt noch eine Authentifizierung über Eingabe der E-Mail-Adresse und des Passworts des Google-Accounts, das für die Registrierung der Applikation verwendet wurde.

Nachdem noch in der Eingabeaufforderung Informationen über das durchgeführte Deployment ausgegeben wurde, wurde die Applikation am Google-Server hochgeladen. Bei der einfachen Ausgestaltung von HelloWorld tritt im Normalfall auch kein Fehler auf.

Der Aufruf erfolgt nun über die URL <http://dawien2009test1.appspot.com/>. Da keine Authentifizierung für das Skript erforderlich ist, wird sofort „Hello, world!“ im Browser dargestellt, in der gleichen Form wie im oben dargestellten Screenshot (Abbildung 6.1.).

Im nächsten Abschnitt soll nun eine etwas komplizierte Anwendung produktiv geschaltet werden. Hierzu wurde die Demo-Applikation eines Schachspiels verwendet.

## **6.2 Produktion: Schachspiel**

Als zweites, komplexeres Beispiel soll nun die Vorgehensweise für die Produktivschaltung eines Schachspiels umgesetzt als Internet-Applikation beschrieben werden. Der *Source-Code* findet sich auf der Demo-Seite von Google Appengine wieder.

Zuerst soll nun die Funktionsweise der Applikation beschrieben werden, danach erfolgt ein lokaler Test und abschließend wieder das Deployment in die Produktionsumgebung.

### **6.2.1 Applikationsentwicklung**

In dieser Umsetzung wurde das Schachspiel derart gestaltet, dass ein Spiel über Internet ausgeführt werden kann, wobei beide Spieler an geographisch unterschiedlichen Orten sitzen können. Hierzu gibt es sowohl Programmteile, die lokal ablaufen (z.B. Prüfung, ob ein gewählter Zug gültig ist oder ob eine Bedingung für das Ende des Spiels erfüllt ist), als auch solche, die auf einen Server laufen (so z.B. die Kommunikation zwischen den Spielern, als auch der Austausch der Spielzüge).

Im Einzelnen sind dies serverseitig folgende Python-Skripts:

- gamemodel.py
- chat.py
- lobby.py
- blitz.py
- ajax.py
- lobby\_ajax.py
- game\_ajax.py

Hierbei dienen die ersten drei Skripts für die Speicherung von Daten und die anderen Skripts zur Behandlung von Serveranfragen durch *Clients*.

Die Client-Seite der Applikation wurde in JavaScript programmiert, da auf diesen natürlich nicht die Installation der Python-Programmiersprache vorausgesetzt werden kann.

Hierbei gibt es folgende Skripts, die teilweise mit den Server-seitigen Skripts korrespondieren, es sind dies folgende:

- `init.js`
- `blitz.js`
- `chat.js`
- `lobby.js`
- `game.js`
- `chess.js`

Um Doppelentwicklungen zu vermeiden wurden für die Programmierung bereits bestehende Open Source-Bibliotheken und Erweiterungen verwendet. Zum Einsatz kamen diesbezüglich die folgenden:

- jQuery ([www.jquery.com](http://www.jquery.com))
- jQuery curvy corners ([www.curvycorners.net](http://www.curvycorners.net))
- simplemodal ([www.ericmartin.com](http://www.ericmartin.com))
- simplejson ([simplejson.googlecode.com](http://simplejson.googlecode.com))
- Jin Chess ([jinchess.com](http://jinchess.com))

Die bisher beschriebenen Programmteile könnten in dieser Form auf jeden Server bereitgestellt werden, wenn dieser richtig konfiguriert ist (so muss z.B. ein Aufruf der Website auf das Skript `blitz.py` umleiten) würde es auch ohne weitere Änderungen dort funktionieren.

Für ein Deployment auf die Google Appengine ist hingegen noch zusätzlich das Konfigurationsfile **app.yaml** notwendig. Da dies für unsere Beschreibung in dieser Arbeit am wesentlichsten ist, soll noch - bevor auf die Funktionalitäten der Schach-Applikation eingegangen wird und wie die einzelnen Programmteile für deren Bereitstellung zusammenspielen - vorab diese im Detail erläutert werden.

Der Source-Code des Files sieht folgendermaßen aus:

```
application: dawien2009test1
version: 1
runtime: python
api_version: 1
default_expiration: 1h
```

handlers:

- url: /images

static\_dir: images

- url: /css

static\_dir: css

- url: /javascript

static\_dir: javascript

- url: /\*

script: blitz.py

Wie bereits im Abschnitt über das Konfigurationsfile der HelloWorld-Applikation beschrieben, ist der erste Abschnitt für jede Applikation gleich:

- Die erste Zeile gibt wieder den Applikation-Identifizier an, dieser könnte naheliegenderweise auf Bezeichnungen wie „blitz“ oder „schachspiel“ gesetzt werden. In diesen Fall müsste genau diese Bezeichnungen für das Deployment in der Google App Engine registriert werden. Einfacherweise wurde aber die gleiche Bezeichnung wie bei der vorigen Applikation gewählt, damit ist keine neue Registrierung notwendig, sondern es wird der Speicherplatz für die HelloWorld-Applikation verwendet (die dann beim Deployment überschrieben wird).
- Die zweite bis vierte Zeile gibt die Version der Applikation und die Parameter für die Programmierumgebung der Google Appengine wieder, nachdem die Server-seitigen Skripts wieder in Python entwickelt wurden, sind diese Angaben gleich zur HelloWorld-Applikation.

Neu ist hier nur die Zeile mit der Angabe der „default\_expiration“, diese Angabe ist optional und gibt die Dauer für das Caching von statischen Files im Browser der Spieler an (soweit diese nicht von einer „expiration“-Angabe bei einem Handler übersteuert wird). Aufgrund der Einfachheit der HelloWorld-Applikation war diese Angabe dort nicht notwendig.

Nun folgen die Zeilen mit den applikationsspezifischen Details. Der Abschnitt über die „Handlers“ gibt an, wie ein http-Request abgearbeitet wird, aufgrund der URL wird erkannt wie vorzugehen ist.

Da Bilder, aber auch die Textdateien für Cascading Style Sheets und Javascripts nicht über den Python-Interpreter laufen müssen, weil sie bereits in dieser Form verständlich für den Browser sind, wurde für diese der Parameter „static\_dir“ gesetzt. Dadurch wird der Google Appengine mitgeteilt, dass in diesem Verzeichnis nur statische Files enthalten sind. Somit werden diese nicht als Script (für den Python-Interpreter) verstanden, sondern werden direkt an den Browser gesendet, was zu Einsparen bei der Server-Rechenkapazität führt.

Hierbei gibt der Parameter „url“ wieder den regulären Ausdruck an, mit dem ankommenden http-Requests interpretiert werden müssen, damit dieser Handler gültig ist. Aufgrund des Parameterwerts werden URLs, die „/images“, „/css“ oder „/javascript“ enthalten, als Verzeichnisse mit statischen Files behandelt. Das entsprechend aufzurufende Verzeichnis ist als Parameterwert zu „static\_dir“ angegeben.

Der Einfachheit halber heißen die Verzeichnisse gleich wie der URL-Bestandteil im regulären Ausdruck. Somit wird z.B. bei einem Aufruf des Bilds für den schwarzen König als URL „/images/black\_king.gif“ angegeben und der Handler leitet diese Anfrage an das Verzeichnis „/images“ weiter, der dort das Bild „black\_king.gif“ findet und an den Client ausliefert.

Die Schach-Applikation selbst ist nun so aufgebaut, dass es ein zentrales Python-Skript gibt, das alle Anfragen behandelt, daher enthält der reguläre Ausdruck den Stern-Operator, womit alle Anfragen - die nicht schon vorher behandelt wurden, also Bilder, CSS und JavaScripts – an dieses weitergeleitet werden. Dies ist das Skript „blitz.py“, das als Wert beim Parameter „script“ angegeben ist.

## 6.2.2 Testen der Applikation

Der Prozess für das Testen der Applikation unterscheidet sich grundsätzlich nicht vom bereits beschriebenen Prozess für die HelloWorld-Applikation. Es sind wieder folgende Schritte notwendig:

- Start des lokalen Entwicklungsservers über die MS-DOS-Konsole, nun mit dem Verzeichnis der Schach-Applikation:

```
dev_appserver.py cloud\blitz\
```

- Aufruf der Applikation in einem beliebigen Browser über die URL <http://localhost:8080/>, danach ist eine Login-Maske sichtbar.

Das Testen der Applikation gestaltet sich nun etwas schwieriger, da die Schach-Applikation zwei unterschiedliche, angemeldete Anwender benötigt. Dies ist zwar grundsätzlich kein Problem, da für den Login selbst auf der Entwicklungsumgebung jede beliebige E-Mail-Adresse/Name gewählt werden kann (es erfolgt keine Authentifizierung gegen ein Benutzerverzeichnis). Problematisch ist nur, dass die Anwender jeweils nacheinander angemeldet sein müssen. So muss sich z.B. User A anmelden, lädt dort in der Lobby zu einem Schachspiel ein, meldet sich wieder ab. Nun ist User B an der Reihe, meldet sich an, tritt dem Schachspiel bei, meldet sich wieder ab, dann muss sich wieder User A anmelden und seinen ersten Zug tätigen, nun ist wieder User B an der Reihe – nur so kann abwechseln getestet werden, ob der Zug für die jeweiligen Figuren (König, Dame, Springer usw.) korrekt funktioniert, ob Endbedingungen korrekt funktionieren, ob gechattet werden kann, usw.

In der Praxis wird man diese Tests auf andere Weise durchführen. Es vereinfacht die Angelegenheiten bereits, wenn zwei Rechner im Netzwerk vorhanden sind, an denen zwei Personen sich am lokalen Server anmelden. Ansonsten gibt es aber natürlich die gesamte Bandbreite an professionellen Entwicklungs- und Testumgebungen, mit denen sowohl während der Entwicklung, als auch vor der Produktivschaltung Test durchgeführt werden können.

### 6.2.3 Deployment der Applikation

Nachdem wie im Kapitel 6.2.1 beschrieben, der Application Identifier gleich zur HelloWorld-Applikation gesetzt wurde, verkürzt sich das Deployment der Schach-Applikation auf den Befehl zur Durchführung des Deployments. Dieser lautet aufgrund der Verwendung eines anderen Verzeichnisses nun:

```
appcfg.py update cloud\blitz\
```

Nach der Authentifizierung mit dem Google Appengine-Benutzerkonto, werden die Dateien wieder auf den Google-Server geladen. Der einzige Unterschied zur HelloWorld-Applikation ist, dass dies nun aufgrund der höheren Anzahl an Dateien länger dauert. Der Aufruf erfolgt danach wieder über die bereits bekannte URL <http://dawien2009test1.appspot.com/>.

Da der Server in der Schach-Applikation wissen muss, welche zwei Spieler gegeneinander spielen, erscheint nun eine Login-Seite, auf der man den Titel der Applikation laut Registrierung sieht:

Google accounts

**Diplomarbeit2009-Test1 uses Google Accounts for Sign In.**

Google is not affiliated with the contents of **Diplomarbeit2009-Test1** or its owners. If you sign in, Google will share your email address with **Diplomarbeit2009-Test1**, but not your password or any other personal information.

**Diplomarbeit2009-Test1** may use your email address to personalize your experience on their website.



The screenshot shows a login form with the following elements: a header 'Sign in with your Google Account', an 'Email:' input field, a 'Password:' input field, a checked checkbox for 'Stay signed in', a 'Sign in' button, and a link at the bottom that reads 'I cannot access my account'.

Abbildung 6.3.: Screenshot: Login-Maske Schach-Applikation

Alternativ könnte, wie in Kapitel 6.1.3. beschrieben, eine weitere Applikation registriert werden. Dementsprechend müsste in der Konfigurationsdatei der Application-Identifizier abgeändert werden und natürlich würde sich auch die URL ändern, ansonsten wäre der Deployment-Prozess aber genauso schnell erledigt, wie soeben beschrieben.

Die nachfolgenden zwei Screenshots stellen Beispiele für die Applikation dar.

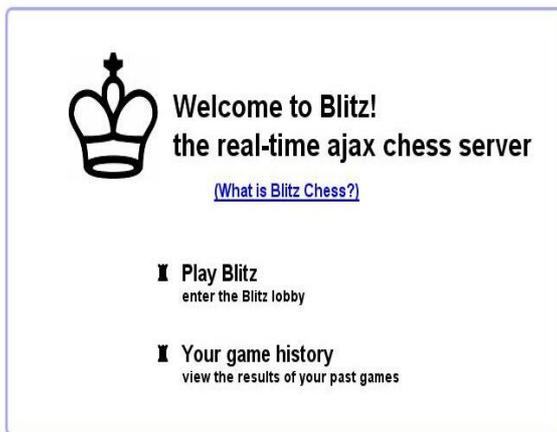


Abbildung 6.4.: Screenshot: Startseite Schach-Applikation

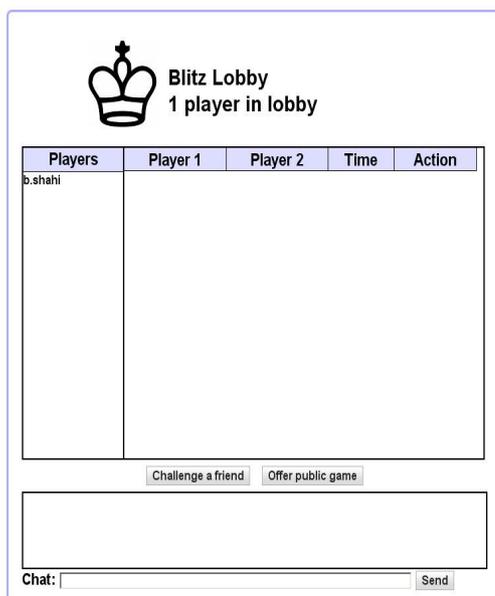


Abbildung 6.5.: Screenshot: Blitz Lobby

### **6.3 Resultat**

Wie aus den Erläuterungen zu den beiden Applikationen ersichtlich ist, stellt die Verwendung der Google Appengine kein Problem dar und wäre sogar von Laien schnell erlernbar. Dies ist natürlich dadurch bedingt, dass Google dem Entwickler starke Einschränkungen unterwirft und den Großteil der Plattform fertig ohne viele weiteren Konfigurationsmöglichkeiten vorgibt – so ist die Entwicklungssprache vorgegeben, mit dem Betriebssystem und der installierten Software kommt ein Entwickler überhaupt nicht in Berührung. Dadurch sind aber andererseits auch keine speziellen Anwendungen möglich – was man als „traditionelle“ Cloud-Anwendungen sehen könnte, wie z.B. Report-Erstellungen mittels proprietärer Software zum Ultimo eines Monats. Google konzentriert sich rein auf Internet-Anwendungen und sobald, wie geplant, auch Java als Programmiersprache verfügbar ist, wird dies sicherlich zu einem interessanten Ansatz anstatt eigene Server zu betreiben.

Wie man auch erkennen kann, ist das aufwändigste bei der Verwendung der Google Appengine die Entwicklung der Web-Applikation selbst. Aber meine Vermutung wäre, dass sich zukünftig Firmen auf Applikationsentwicklung für die Google Appengine spezialisieren werden. Aber auch die ganze Bandbreite an Open Source-Anwendungen in Python (bzw. Java) wie Content-Management-Systeme, Webshops usw. sind ohne großartige Anpassungen einsetzbar (im Wesentlichen ist nur die Konfigurationsdatei `app.yaml` anzupassen).

## 7 Zusammenfassung

Wenn man die Entwicklung von Grid und Cloud Computing in einer historischen Perspektive betrachtet, kann man eine natürliche Entwicklung hin zu einer Vereinfachung der Nutzung von Computer-Ressourcen erkennen. Zu Beginn (und auch heute noch) hatten viele Unternehmen ihre eigenen Server, bei denen sie sich sowohl um die Hardware (Austausch defekter Festplatten, Neukauf von Servern und vieles mehr), als auch um Betriebssysteme und Applikationen kümmerten. Wenn zum Beispiel eine CRM-Applikation benötigt wurde, ist diese von einem betriebsinternen Programmiererteam entwickelt worden. Dies war zwar nicht überall der Fall, aber in der Entscheidung zwischen „Make“ oder „Buy“ wurde öfter auf das Selber machen entschieden.

Im Zuge des immer mehr vorherrschenden „Outsourcing“-Trends wurden immer mehr Kompetenzen ausgelagert und auch in größeren Unternehmen weniger selbst übernommen. Dieser Trend führte schließlich zum „Everything as a Service“-Konzept und zum Cloud Computing.

Man kann somit beide Konzepte als Vereinfachung der Nutzung von Computerressourcen sehen. Anstatt sich mit den „Innereien“ als Anwender auseinanderzusetzen, mietet man nur eine Cloud und nutzt deren Ressourcen. Was im Hintergrund abläuft, interessiert nur den Anbieter. Für den Cloud-Nutzer stellt sich nur die zu lösende Geschäftsanwendung als betrachtenswert heraus.

Im Vergleich dazu ist die Vereinfachung beim Grid Computing nicht so weitgehend, je nach Art der Umsetzung muss man sich in manchen Lösungen auch um die Administration von einzelnen Rechnern kümmern (obwohl dies dann durch die Verwendung von haushaltsüblichen PCs anstatt von Superrechnern keine spezialisierten Kenntnisse voraussetzt), während bei anderen Lösungen die Grid-Software derart programmiert ist, dass sie auf unterschiedlicher Hardware / Betriebssystemen laufen kann und eine einfache Aufspaltung in Arbeitspaketen ermöglicht, ohne dass sich der Anwender des Grids um die dahinterstehenden Rechner selbst kümmern muss.

Damit wären wir auch beim wesentlichen Unterschied zwischen dem Konzept des Grid und des Cloud-Computings:

- bei einem Grid ist die Lösung verteilt, wobei dies bis hin zu einem weltweit verteiltem Rechnernetzwerk gehen kann,
- während bei einer Cloud die Lösung zentralisiert ist und von einem Cloud-Anbieter die Rechenleistung zur Verfügung gestellt wird.

In Hinblick zu dem der zu Beginn betrachteten Analogie der Stromerzeugung würde beim Grid eine Anwendung in kleinere Arbeitspakete zerlegt werden und danach für ein, zwei Arbeitspakete der „Strom“ (Rechenleistung) im Kleinkraftwerk A erzeugt werden, für ein anderes im Kleinkraftwerk B und einige könnten im Großkraftwerk C erzeugt werden. Bei einer Cloud hingegen würde die Anwendung nur in dieser einzigen zentral laufen und nicht auf mehrere Clouds verteilt werden. Der Anwender hat nur einen Vertrag mit dem Großkraftwerk.

Durch diese Zentralisierung wird auch das Problem von mehrfachen SLAs, siehe hierzu Abschnitt 2.4, gelöst.

### III Literaturverzeichnis

- [1] W. Leinberger und V. Kumar: *Information power grid: The new frontier in parallel computing?*. IEEE Concurrency, 7(4):75–84, Okt./Dez. 1999.
- [2] Ian Foster und Carl Kesselmann: *The Grid: Blueprint for a New Computing Infrastructure*. Morgen Kaufmann, 1. Auflage 1999.
- [3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, M. Zaharia: *Above the Clouds: A Berkeley View of Cloud Computing*, Feb. 2009.
- [4] Free Software Foundation: *GNU Free Documentation License*. Online <http://www.gnu.org/copyleft/fdl.html>, Version 1.2. Nov.2002.
- [5] Ian Foster und Carl Kesselmann: *The Grid: Blueprint for a New Computing Infrastructure*. Morgen Kaufmann, 2. Auflage 2003.
- [6] Ian Foster: *What is Grid? A Three Point Checklist*. Grid today, 1(6), Juli 2002.
- [7] Fran Berman, Geoffrey Fox und Tony Hey: *The Grid: past, present, future*. In: Fran Berman et. al. [8], Kap. 1, S. 9–50.
- [8] Fran Berman, Geoffrey Fox und Tony Hey: *Grid Computing – Making the Global Infrastructure a Reality*. John Wiley & Sons, 2002.
- [9] Ian Foster und Carl Kesselmann: *Computational Grid*: In: *The Grid: Blueprint for a New Computing Infrastructure* [2], Kap.2.

[10] De Roure, D., M. A. Baker, N. R. Jennings und N. R. Shadbolt: *The evolution of the Grid*. In: Fran Berman et al. [8], Kap. 3, S. 65–100.

[11] *RSA130: Getting started with FAFNER*.  
<http://cs-www.bu.edu/cgi-bin/FAFNER/tutorial.pl>.

[12] Korpela, E., D. Werthimer, D. Anderson, J. Cobb und M. Lebofsky: *SETI@home – Massively Distributed Computing for SETI*. *Computing in Science & Engineering*, 3(1): S. 78–83, Jan. 2001.

[13] *Distributed.net*.  
<http://www.distributed.net>.

[14] DeFanti, T. A., I. Foster, M. E. Papka und R. Stevens: *Overview of the I-WAY: Wide Area Visual Supercomputing*. *International Journal of Supercomputer Applications*, 10(2): S. 123–130, 1996.

[15] Cruz-Neira, C., D. J. Sandin und T. A. DeFanti: *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*. In: *Proceedings of SIGGRAPH 93 Computer Graphics Conference*, S. 135–142, 1993.

[16] Grimshaw, A. S. und W. A. Wulf: *The Legion vision of a worldwide virtual computer*. *Communications of the ACM*, 40(1): S. 39–45, Jan. 1997.

[17] *Common Object Request Broker Architecture: Core Specification*. Specification Version 3.0.3, Object Management Group, März 2004.

[18] *Jini Architecture Specification*.  
[http://www.jini.org/wiki/Jini\\_Architecture\\_Specification](http://www.jini.org/wiki/Jini_Architecture_Specification).

- [19] Booth, D., H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris und D. Orchard: *Web Services Architecture*. Working Group Note, W3C, Feb. 2004.
- [20] A.J. Chakravarti: *Enhancing e-Infrastructures with Advanced Technical Computing: Parallel MATLAB on the Grid*, May 2008.
- [21] Wikipedia: *Virtuelle Organisation*.  
[http://de.wikipedia.org/wiki/Virtuelle\\_Organisation](http://de.wikipedia.org/wiki/Virtuelle_Organisation). Lizensiert unter GNU FDL [4].
- [22] Ian Foster, Carl Kesselman und S. Tuecke: *The anatomy of the Grid*. In: Fran Berman et. al. [8], Kap. 6, S. 171–197.
- [23] Wikipedia: *Peer-to-peer*.  
<http://de.wikipedia.org/wiki/Peer-to-Peer>. Lizensiert unter GNU FDL [4].
- [24] Ian Foster und Carl Kesselman: *Globus: A Metacomputing Infrastructure Toolkit*. The International Journal of Supercomputer Applications and High Performance Computing, 11(2): S. 115–128, 1997.
- [25] Ian Foster, Carl Kesselman, J. M. Nick und S. Tuecke: *The physiology of the Grid*. In: Fran Berman et al. [8], Kap. 8, S. 217–249.
- [26] Ian Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell und J. V. Reich: *The Open Grid Services Architecture, Version 1.5*. Informational GDF 80, Global Grid Forum, July 2006.
- [27] VGE: *Vienna Grid Environment*.  
<http://www.par.univie.ac.at/project/vge/>.

[28] Benkner, S., I. Brandic, G. Engelbrecht und R. Schmidt: *VGE – a service-oriented grid environment for on-demand supercomputing*. In: *2004. Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing*, S. 11–18, Nov. 2004.

[29] *The GEMSS Project: Grid-Enabled Medical Simulation Services*.  
<http://www.it.neclab.eu/gemss/>.

[30] Benkner, S., G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. Middleton und R. Schmidt: *GEMSS: Grid-infrastructure for Medical Service Provision*. In: *HealthGRID 2004, Clermont-Ferrand, France*, Jan. 2004.

[31] Wikipedia: *SETI@home*.  
<http://de.wikipedia.org/wiki/SETI@home>.

[32] T. Dierks und C. Allen: *The TLS Protocol Version 1.0*. RFC 2246, IETF, Jan.1999.

[33] *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)*. Standard 200401, OASIS, März 2004.

[34] *Web Service Security for Java – WSS4J*.  
<http://ws.apache.org/wss4j/>.

[35] *TeraGrid*.  
<http://www.teragrid.org>.

[36] Wikipedia: *TeraGrid*.  
<http://en.wikipedia.org/wiki/TeraGrid>.

[37] *EGEE: Enabling Grids for E-science.* .

<http://www.eu-egee.org/>

<http://www.gup.uni-linz.ac.at/research/egee/>.

[38] *Globus Toolkit.*

<http://www.globus.org>.

[39] Ian Foster: *Globus Toolkit Version 4: Software for Service-Oriented Systems.* In: *IFIP International Conference on Network and Parallel Computing*, Nr. 3779 in *Lecture Notes in Computer Science*, S. 2–13. Springer, 2005.

[40] *Deisa: Distributed European Infrastructure for Supercomputing Applications.*

<http://www.deisa.org>

[41] *OSG: Open Science Grid.*

<http://www.opensciencegrid.org>

[42] *APAC: Australian Partnership for advanced Computing*

<http://www.apac.edu.au/>

[43] *Naregi: National Research Grid Initiative*

<http://www.naregi.org>

[44] *gLite: Lightweight Middleware for Grid Computing.*

<http://glite.web.cern.ch/glite/>

[45] Ian Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell und J. V. Reich: *The Open Grid Services Architecture, Version 1.5.* Informational GDF 80, Global Grid Forum, Jul. 2006.

[46] *Amazon Web Services.*

<http://aws.amazon.com/>

[47] Jonathan Weiss: *Skalieren von Rails Anwendungen mit Amazon EC2 und S3.* Jun. 2007.

[48] *Power grid.*

[http://en.wikibooks.org/wiki/Distribution\\_of\\_Electric\\_Energy](http://en.wikibooks.org/wiki/Distribution_of_Electric_Energy)

[49] *World Community Grid.*

<http://www.worldcommunitygrid.org/>

[50] Wikipedia: *Autonomic Computing.*

[http://de.wikipedia.org/wiki/Autonomic\\_Computing/](http://de.wikipedia.org/wiki/Autonomic_Computing/)

[51] Google App Engine.

<http://appengine.google.com/>

[52] IBM Blue Cloud.

<http://www.ibm.com/ibm/cloud/>

[53] Microsoft Azure.

<http://www.microsoft.com/azure/>

[54] Wikipedia: *Querverkauf.*

<http://de.wikipedia.org/wiki/Querverkauf>

[55] Amazon EC2 – Getting Started Guide.

<http://docs.amazonwebservices.com/AWSEC2/2009-03-01/GettingStartedGuide/>

[56] Google App Engine: *helloworld*

<http://code.google.com/intl/deDE/appengine/docs/python/gettingstarted/helloworld.html>

[57] YAML Ain't Markup Language: *YAML*

<http://www.yaml.org/>

[58] M. Rosenblum: *The Reincarnation of Virtual Machines*: Queue July/August 2004.

[59] Wikipedia: *Everything as a Service*.

[http://en.wikipedia.org/wiki/Everything\\_as\\_a\\_service](http://en.wikipedia.org/wiki/Everything_as_a_service)

[60] A. König; *11 Cloud-Anwendungen im Check*. CIO. Oktober 2009.

<http://www.cio.de/knowledgecenter/server/2210470/>

[61] Storage-Anwendungen liegen bei Cloud-Computing ganz vorne. Tec-Channel.

[http://www.techchannel.de/storage/news/2022397/storage\\_ist\\_die\\_top\\_cloud\\_anwendung/](http://www.techchannel.de/storage/news/2022397/storage_ist_die_top_cloud_anwendung/)

[62] Amazon EC2: *Preissystem*.

<http://aws.amazon.com/ec2/pricing/>

[63] Google App Engine: *Preissystem*.

<http://code.google.com/intl/de-AT/appengine/docs/billing.html>

[64] Microsoft Azure: *Preissystem*.

<http://www.dotnetpro.de/news2981.aspx>

<http://www.microsoft.com/windowsazure/pricing/>