



**Workshop  
On  
Cloud Computing and Big Data –  
A Paradigm Shift in ICT**

**1 ~ 5 December 2014**

Prepared by:

**Prof. Dr. M. A. Mottalib  
Dr. Md. Kamrul Hasan  
Dr. Md. Motaharul Islam**

**Ashraful Alam Khan  
Nafiul Rashid  
Mir Rayat Imtiaz Hossain**

Course Coordinator:

**Prof. Dr. M. A. Mottalib, Head, CSE Dept, IUT**

Course Organizer & Contact Person

**Dr. Md. Motaharul Islam**  
Assistant Professor, CSE Dept, IUT

Organized by:

Department of Computer Science and Engineering (CSE)

**Islamic University of Technology (IUT)**

An organ of Organisation of Islamic Cooperation (OIC)

Board Bazar, Gazipur-1704, Bangladesh

Web site: <http://www.iutoic-dhaka.edu>, E-mail: [scse@iut-dhaka.edu](mailto:scse@iut-dhaka.edu)

---

**ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT), DHAKA**  
**Organisation of Islamic Cooperation (OIC)**  
**Department of Computer Science and Engineering (CSE)**

**P R O G R A M M E**

for the workshop on

***Cloud Computing and Big Data – A Paradigm Shift in ICT***

***1-5 December 2014***

Registration : 08:30 – 08:45 A.M. on the 1<sup>st</sup> December 2014 at Room No 205. , 2nd Academic Building, IUT

Venue: Room No. 201

Date	Time		
	9:00 AM – 10:45 AM	11:15 AM – 1:00 PM	2:30 PM – 5:00 PM
1 /12/2014 Monday	Slot-1 Lecture – 1 Inauguration and Introduction  Dr. Md. Motaharul Islam Asst. Professor, CSE Department, IUT	Slot-2 Lecture - 2 Google as a Cloud Provider  Dr. Md. Motaharul Islam Asst. Professor, CSE Department, IUT	Slot-3 Lab - 1 Google App Engine, Amazon EC2, VMware  Dr. Md. Kamrul Hasan Asst. Professor, CSE Department, IUT
2 /12/2014 Tuesday	Slot-4 Lecture - 3 Green Cloud  Prof. Dr. Md. Abdur Razzaque Professor, CSE, DU	Slot-5 Lecture - 4 Cloud Technology  Dr. Kazi Muheymin Sakib, Director and Associate Professor, IIT, DU	Slot-6 Lab - 2 Cloud Technology  Asif Imran, Lecturer, IIT, DU
3 /12/2014 Wednesday	Slot-7 Lecture - 5 Cloud Services in Business  Syed Zubair Hossain, CEO, Divine IT Limited	Slot-8 Lecture - 6 Cloud Infrastructure  Rana Khandkar Team Lead, Newscred Inc.	Slot-9 Lab - 3 Cloud Computing: Open Stack  Rana Khandkar Team Lead, Newscred Inc.
4 /12/2014 Thursday	Slot-10 Lecture - 7 Big Data  Dr. Mohammad Rezwanul Huq Assistant Professor Department of CSE, EWU	Slot-11 Lecture - 8 Big Data  Dr. Mohammad Rezwanul Huq Assistant Professor Department of CSE, EWU	Slot-12 Lab-4 Big Data  Dr. Mohammad Rezwanul Huq Assistant Professor Department of CSE, EWU
5 /12/2014 Friday	Slot-13 Lecture - 9 Research Trends: Cloud Computing and Big Data  Dr. Md. Kamrul Hasan Asst. Professor, CSE Department, IUT	Slot-14 Site Visit	Slot-15 Lab -5  Certificate Awarding Ceremony

Tea Break : 10:45 AM – 11:15 AM

Prayer and Lunch Break : 1:00 PM – 2:30 PM

# TABLE OF CONTENTS

<b>RESOURCE PERSONNEL.....</b>	<b>VI</b>
<b>WORKSHOP HANDOUT PREPARATION.....</b>	<b>VII</b>
<b>1. INTRODUCTION TO CLOUD COMPUTING .....</b>	<b>1</b>
1.1 INTRODUCTION .....	1
1.1.1 <i>Layers of Cloud Computing</i> .....	2
1.1.2 <i>Types of Cloud Computing</i> .....	4
1.1.3 <i>Cloud Computing Versus Cloud Services</i> .....	5
1.1.4 <i>Categories of Cloud Providers</i> .....	6
1.2 ENABLING TECHNOLOGIES.....	7
1.2.1 <i>Virtualization</i> .....	7
1.2.2 <i>Web Service and Service Oriented Architecture</i> .....	7
1.2.3 <i>Service Flow and Workflows</i> .....	7
1.2.4 <i>Web 2.0 and Mashup</i> .....	8
1.3 CLOUD COMPUTING FEATURES.....	8
1.3.1 <i>Cloud Computing Standards</i> .....	9
1.3.2 <i>Cloud Computing Security</i> .....	10
1.4 CLOUD COMPUTING PLATFORMS .....	11
1.5 PRICING.....	11
1.6 CLOUD COMPUTING COMPONENTS AND THEIR VENDORS.....	12
1.7 EXAMPLE OF WEB APPLICATION DEPLOYMENT .....	12
1.8 CLOUD COMPUTING CHALLENGES .....	13
1.8.1 <i>Performance</i> .....	13
1.8.2 <i>Security and Privacy</i> .....	13
1.8.3 <i>Control</i> .....	14
1.8.4 <i>Bandwidth Costs</i> .....	14
1.8.5 <i>Reliability</i> .....	14
1.9 AN OVERVIEW OF NETWORK ARCHITECTURES FOR CLOUDS .....	14
1.9.1 <i>Data Center Network</i> .....	14
1.9.2 <i>Data Center Interconnect Network</i> .....	15
1.9.3 <i>Network Architecture for Hybrid Cloud Deployments</i> .....	17
1.9.4 <i>Cloud-in-a-Box</i> .....	17
1.10 CONCLUSIONS AND FUTURE DIRECTIONS .....	18
1.11 REFERENCES .....	18
<b>2. CASE STUDY: GOOGLE AS A CLOUD PROVIDER.....</b>	<b>21</b>
2.1 INTRODUCTION .....	21
2.2 INTRODUCING THE CASE STUDY: GOOGLE.....	21
2.3 OVERALL ARCHITECTURE AND DESIGN PHILOSOPHY .....	25
2.3.1 <i>Physical model</i> .....	25
2.3.2 <i>Overall system architecture</i> .....	26
2.4 UNDERLYING COMMUNICATION PARADIGMS .....	27
2.4.1 <i>Remote invocation</i> .....	28
2.4.2 <i>Publish-subscribe</i> .....	30
2.5 DATA STORAGE AND COORDINATION SERVICES .....	31
2.5.1 <i>The Google File System (GFS)</i> .....	32
2.5.2 <i>Chubby</i> .....	36
2.6 DISTRIBUTED COMPUTATION SERVICES.....	39
2.6.1 <i>MapReduce</i> .....	39
2.7 SUMMARY .....	42
2.8 REFERENCES .....	43
<b>3. TUTORIAL ON CLOUD SERVICES: VMWARE, EC2, APP ENGINE.....</b>	<b>45</b>
3.1 VMWARE: A TOOL TO CREATE IAAS.....	45
3.1.1 <i>How to Install VMware Workstation and Create a Virtual Machine on Your PC</i> .....	45

3.1.2	Sources:.....	51
3.2	AMAZON EC2: INFRASTRUCTURE AS A SERVICE.....	52
3.2.1	Features of Amazon EC2.....	52
3.2.2	Setting Up with Amazon EC2.....	52
3.2.3	Sign Up for AWS.....	52
3.3	TO CREATE AN AWS ACCOUNT.....	53
3.4	CREATE AN IAM USER.....	55
3.4.1	To create the Administrators group.....	56
3.4.2	To create the IAM user, add the user to the Administrators group, and create a password for the user.....	56
3.4.3	Create a Key Pair.....	57
3.4.3.1	To create a key pair.....	57
3.4.3.2	To connect to your instance using your key pair.....	58
3.4.3.3	(Optional) To prepare to connect to a Linux instance from Windows using PuTTY.....	58
3.4.4	Create a Virtual Private Cloud (VPC).....	58
3.4.4.1	To create a nondefault VPC.....	59
3.4.5	Create a Security Group.....	59
3.4.5.1	To create a security group with least privilege.....	59
3.5	LAUNCHING AN AMAZON MACHINE INSTANCE (AMI).....	60
3.5.1	Adding plugins to Netbeans 8.0:.....	63
3.5.2	Configuring GoogleAppEngine SDK:.....	64
3.6	DEVELOPING WEB APPLICATIONS USING GOOGLEAPPEENGINE:.....	64
<b>4.</b>	<b>GREEN CLOUD COMPUTING.....</b>	<b>67</b>
4.1	INTRODUCTION.....	67
4.2	GREEN CLOUD ARCHITECTURE.....	69
4.3	GREEN CLOUD TECHNOLOGIES.....	70
4.4	OPEN RESEARCH CHALLENGES IN GCC.....	78
4.5	CONCLUSION.....	79
4.6	REFERENCES.....	79
<b>5.</b>	<b>CLOUD TECHNOLOGY.....</b>	<b>82</b>
5.1	OVERVIEW.....	82
5.2	RESEARCH TRENDS IN CLOUD COMPUTING.....	82
5.2.1	Software Engineering of Large-Scale systems.....	82
5.2.2	Cloud Service Platform.....	83
5.2.3	Cloud Scalability.....	83
5.2.4	Fault tolerance of cloud environment.....	83
5.3	INSIDE OPENSTACK.....	84
5.4	BEFORE YOU BEGIN.....	85
5.5	WHAT YOU NEED.....	86
5.6	INSTALLATION STEPS.....	86
5.7	TROUBLESHOOTING.....	89
5.7.1	Cannot ssh to the instance:.....	89
5.7.2	When installing nova essex into a new box dpkg error occur and then mysql configuration take a long time and fail:.....	89
5.7.3	Cannot shutdown the instance:.....	89
5.7.4	Error returned when creating the very first instance.....	89
5.7.5	Timeout: Timeout while waiting on RPC response.....	90
5.7.6	Successfully added compute node but cannot create instances in that node.....	90
5.7.7	Disaster Recovery.....	90
5.8	INSTANCE BOOT SETUP.....	90
5.9	FUTURE PERSPECTIVE OF CLOUD.....	91
5.10	CONCLUSION.....	91
<b>6.</b>	<b>CLOUD COMPUTING IN BUSINESS.....</b>	<b>93</b>
6.1	INTRODUCTION.....	93
6.2	THE EMERGING MARKET OF CLOUD SERVICES.....	93
6.3	DIFFERENT TYPES OF CLOUD SERVICES AVAILABLE IN THE MARKET.....	95
6.3.1	Infrastructure as a Service (IaaS).....	95
6.3.2	Platform as a Service (PaaS).....	96

6.3.3	<i>Software as as Service (SaaS)</i> .....	96
6.4	IMPACT OF CLOUD SERVICES IN AN ORGANIZATION .....	97
6.5	PRODUCTIVITY: MORE BUSINESS WITH LESS IT .....	97
6.5.1	<i>Resource Utilization</i> .....	97
6.5.2	<i>Improved Communication</i> .....	98
6.5.3	<i>Flexible working environment</i> .....	98
6.5.4	<i>Costing: Pay for only what you use</i> .....	98
6.5.4.1	<i>Usage-Based Pricing</i> .....	98
6.5.5	<i>Lifetime Cost Models</i> .....	99
6.5.6	<i>Specialization and Scale</i> .....	100
6.6	SPEED: GETTING THERE MORE QUICKLY .....	100
6.6.1	<i>Time to Deployment</i> .....	100
6.6.2	<i>IT Asset Management</i> .....	100
6.7	SIZE: ADAPTING TO BUSINESS NEEDS .....	101
6.7.1	<i>Entering New Markets</i> .....	101
6.7.2	<i>High-Value Services</i> .....	101
6.8	QUALITY: IMPROVED MARGIN FROM BETTER SERVICE .....	101
6.8.1	<i>Competitive Pressure</i> .....	102
6.8.2	<i>The Importance of Quality</i> .....	102
6.9	SECURITY: ORGANIZATION'S TOP PRIORITY .....	103
6.9.1	<i>Privileged user access</i> .....	103
6.9.2	<i>Physical security</i> .....	103
6.9.3	<i>Regulatory compliance</i> .....	103
6.9.4	<i>Data location</i> .....	103
6.9.5	<i>Data segregation</i> .....	103
6.10	RISKS: YOUR DATA, YOUR CONCERN .....	103
6.10.1	<i>Data loss</i> .....	103
6.10.2	<i>Availability</i> .....	104
6.10.3	<i>Long-term viability</i> .....	104
6.11	CASE STUDIES .....	104
6.11.1	<i>How cloud services can affect product pricing</i> .....	104
6.12	CLOUD SERVICES IN SOFTWARE DEVELOPMENT - ENHANCING AGILE SOFTWARE DEVELOPMENT .....	105
6.12.1	<i>Cloud Computing Provides an Unlimited Number of Testing and Staging Servers</i> .....	105
6.12.2	<i>It Turns Agile Development Into a Truly Parallel Activity</i> .....	105
6.12.3	<i>It Encourages Innovation and Experimentation</i> .....	105
6.12.4	<i>It Enhances Continuous Integration and Delivery</i> .....	105
6.12.5	<i>It Makes More Development Platforms and External Services Available</i> .....	105
6.13	USE OF CLOUD SERVICES IN DIFFERENT INDUSTRIES .....	106
6.13.1	<i>Small and medium industries</i> .....	106
6.14	LARGE INDUSTRIES .....	106
6.15	COMMON PITFALLS IN USING CLOUD SERVICES .....	107
6.15.1	<i>Not fully understanding cloud security</i> .....	107
6.15.2	<i>Not understanding the cost model and usage</i> .....	107
6.15.3	<i>Disasters and lack of recovery plan</i> .....	107
6.15.4	<i>Not preparing for outages</i> .....	107
6.15.5	<i>The fine print of Terms and Conditions</i> .....	107
6.15.6	<i>Underestimating the impacts of organizational change</i> .....	107
6.15.7	<i>Migrating applications to the cloud solely to drive down costs</i> .....	107
6.15.8	<i>Having inflated expectations of suddenly becoming a digital enterprise</i> .....	108
6.15.9	<i>Selecting a favorite vendor, not an appropriate vendor</i> .....	108
6.15.10	<i>Not bringing in enough of the right skills</i> .....	108
6.15.11	<i>Misunderstanding customer requirements</i> .....	108
6.16	FINAL WORDS .....	108
6.17	REFERENCES .....	109
<b>7.</b>	<b>CLOUD INFRASTRUCTURE: OPENSTACK</b> .....	<b>111</b>
7.1	INTRODUCTION TO OPENSTACK .....	111
7.1.1	<i>Getting Started with OpenStack</i> .....	111
7.1.2	<i>Using OpenStack</i> .....	111
7.1.3	<i>Plug and Play OpenStack</i> .....	111

7.2	ARCHITECTURE .....	112
7.2.1	<i>Example Architectures</i> .....	113
7.2.1.1	Example Architecture—Legacy Networking (nova) .....	113
7.2.1.2	Example Architecture—OpenStack Networking .....	117
7.2.2	<i>Parting Thoughts on Architectures</i> .....	129
7.2.3	<i>Provisioning and Deployment</i> .....	129
7.2.3.1	Automated Deployment .....	129
7.2.3.2	Automated Configuration .....	131
7.2.3.3	Remote Management .....	132
7.2.3.4	Parting Thoughts for Provisioning and Deploying OpenStack .....	132
7.2.3.5	Conclusion .....	132
7.2.4	<i>Designing for Cloud Controllers and Cloud Management</i> .....	132
7.2.4.1	Hardware Considerations .....	133
7.2.4.2	Separation of Services: .....	134
7.2.4.3	Database .....	135
7.2.4.4	Message Queue .....	135
7.2.4.5	Conductor Services .....	135
7.2.4.6	Application Programming Interface (API) .....	136
7.2.4.7	Extensions .....	136
7.2.4.8	Scheduling .....	136
7.2.4.9	Images .....	136
7.2.4.10	Dashboard .....	137
7.2.4.11	Authentication and Authorization .....	137
7.2.4.12	Network Considerations .....	137
7.2.5	<i>Network Design</i> .....	138
7.2.5.1	Management Network .....	138
7.2.5.2	Public Addressing Options .....	138
7.2.5.3	IP Address Planning .....	138
7.2.5.4	Network Topology .....	140
7.2.5.5	Services for Networking .....	141
7.2.5.6	Conclusion .....	142
<b>8.</b>	<b>BIG DATA .....</b>	<b>144</b>
8.1	WHAT IS BIG DATA? .....	144
8.1.1	<i>Characteristics of Big Data</i> .....	144
8.1.2	<i>Can There Be Enough? The Volume of Data</i> .....	145
8.1.3	<i>Variety Is the Spice of Life</i> .....	146
8.1.4	<i>How Fast Is Fast? The Velocity of Data</i> .....	147
8.1.5	<i>Data in the Warehouse and Data in Hadoop</i> .....	147
8.1.6	<i>Conclusion</i> .....	149
8.2	MAPREDUCE AND THE NEW SOFTWARE STACK .....	149
8.2.1	<i>Distributed File Systems</i> .....	150
8.2.2	<i>Physical Organization of Compute Nodes</i> .....	150
8.2.3	<i>Large-Scale File-System Organization</i> .....	151
8.2.4	<i>MapReduce</i> .....	152
8.2.4.1	The Map Tasks .....	153
8.2.4.2	Grouping by Key .....	154
8.2.4.3	The Reduce Tasks .....	154
8.2.4.4	Combiners .....	154
8.2.5	<i>Details of MapReduce Execution</i> .....	154
8.2.6	<i>Coping with Node Failures</i> .....	155
8.2.7	<i>Algorithms Using MapReduce</i> .....	156
8.2.8	<i>Relational-Algebra Operations</i> .....	156
8.2.9	<i>Computing Selections by MapReduce</i> .....	158
8.2.10	<i>Computing Projections by MapReduce</i> .....	158
8.2.11	<i>Union, Intersection, and Difference by MapReduce</i> .....	159
8.2.12	<i>Computing Natural Join by MapReduce</i> .....	159
8.2.13	<i>Grouping and Aggregation by MapReduce</i> .....	160
8.3	HADOOP FRAMEWORK AND BIG DATA .....	160
8.3.1	<i>Properties of Hadoop</i> .....	160
8.3.2	<i>Business Case for Hadoop</i> .....	161
8.3.3	<i>Hadoop = HDFS + MapReduce</i> .....	161
8.3.4	<i>Why Hadoop?</i> .....	162

8.3.5	<i>HDFS - Hadoop Distributed File System</i> .....	164
8.3.6	<i>MapReduce</i> .....	164
8.3.7	<i>HBase, the database for Big Data</i> .....	164
8.3.8	<i>ZooKeeper</i> .....	164
8.3.9	<i>Hive - data warehousing</i> .....	164
8.3.10	<i>Pig - Big Data manipulation</i> .....	164
8.3.11	<i>Hadoop alternatives</i> .....	165
8.3.12	<i>Large data storage alternatives</i> .....	165
8.3.13	<i>Large database alternatives</i> .....	165
8.3.14	<i>Alternatives for distributed massive computations</i> .....	166
8.3.15	<i>Arguments for Hadoop</i> .....	167
8.4	<b>HADOOP DISTRIBUTED FILE SYSTEMS (HDFS)</b> .....	167
8.4.1	<i>HDFS Concepts</i> .....	167
8.4.2	<i>HDFS Architecture</i> .....	169
8.4.2.1	Master / worker design.....	169
8.4.2.2	Runs on commodity hardware.....	170
8.4.2.3	HDFS is resilient (even in case of node failure).....	170
8.4.2.4	Data is replicated.....	170
8.4.2.5	HDFS is better suited for large files.....	170
8.4.2.6	Files are write-once only (not updateable).....	170
8.5	<b>PRACTICAL SESSION ON MYSQL REPLICATION AND PARTITION SCHEME AND HBASE INSTALLATION</b> .....	170
8.5.1	<i>Replication in MySQL</i> .....	170
8.5.2	<i>Partitioning in MySQL</i> .....	172
8.5.3	<i>Hadoop and HBase Installation in Windows 7 using Cygwin</i> .....	174
8.5.4	<i>Software</i> .....	175
8.5.4.1	Cygwin.....	175
8.5.4.2	Hadoop.....	176
8.5.4.3	ZooKeeper.....	177
8.5.4.4	HBase.....	178
<b>9.</b>	<b>CLOUD RESEARCH CHALLENGES</b> .....	<b>181</b>
9.1	<b>INTRODUCTION</b> .....	181
9.2	<b>THE ADVENT OF THE “CLOUDS”</b> .....	182
9.2.1	<i>What is a “Cloud”</i> .....	183
9.2.2	<i>Terminology</i> .....	183
9.2.2.1	Types of Clouds.....	184
9.2.2.2	Deployment Types (Cloud Usage).....	185
9.2.2.3	Cloud Environment Roles.....	186
9.2.3	<i>Specific Characteristics / Capabilities of Clouds</i> .....	187
9.2.3.1	Non-Functional Aspects.....	187
9.2.3.2	Economic Aspects.....	188
9.2.3.3	Technological Aspects.....	189
9.2.4	<i>Related Areas</i> .....	190
9.2.4.1	Internet of Services.....	190
9.2.4.2	Internet of Things.....	190
9.2.4.3	The Grid.....	191
9.2.4.4	Service Oriented Architectures.....	192
9.3	<b>GAPS AND OPEN AREAS</b> .....	192
9.3.1	<i>Technical Gaps</i> .....	192
9.3.1.1	Manageability and Self -*.....	192
9.3.1.2	Data Management.....	193
9.3.2	<i>Privacy and Security</i> .....	194
9.3.3	<i>Federation and Interoperability</i> .....	194
9.3.4	<i>Virtualization, Elasticity and Adaptability</i> .....	195
9.3.5	<i>Apis, Programming Models and Resource Control</i> .....	196
9.4	<b>NON-TECHNICAL GAPS</b> .....	196
9.4.1	<i>Legislation, Government and Policies</i> .....	196
9.4.2	<i>Economic Concerns</i> .....	197
9.4.3	<i>Business Models and Expert Systems</i> .....	197
9.5	<b>REFERENCES &amp; SOURCES</b> .....	198

## **RESOURCE PERSONNEL**

---

**Prof. Dr. Md. Abdur Razzaque**

Professor, CSE, University of Dhaka

**Dr. Kazi Muheymin Sakib**

Director and Associate Professor, IIT, University of Dhaka

**Dr. Md. Kamrul Hasan**

Asst. Professor, CSE Department, Islamic University of Technology (IUT)

**Dr. Md. Motaharul Islam**

Asst. Professor, CSE Department, Islamic University of Technology (IUT)

**Dr. Mohammad Rezwanaul Huq**

Assistant Professor

Department of CSE, East West University (EWU)

**Rana Khandkar**

Team Lead, Newscred Inc.

**Syed Zubair Hossain, CEO,**

Divine IT Limited

**Asif Imran**

Lecturer, IIT, University of Dhaka



## WORKSHOP HANDOUT PREPARATION

---

1. Chapter 1: **Introduction to Cloud Computing**  
**Dr. Md. Motaharul Islam**  
Asst. Professor, CSE Department, Islamic University of Technology (IUT)
2. Chapter 2: **Case Study: Google as a Cloud Provider**  
**Dr. Md. Motaharul Islam**  
Asst. Professor, CSE Department, Islamic University of Technology (IUT)
3. Chapter 3: **Tutorial on Cloud Services: VMWare, EC2, App Engine**  
**Dr. Md. Kamrul Hasan**  
Asst. Professor, CSE Department, Islamic University of Technology (IUT)
4. Chapter 4: **Green Cloud Computing**  
**Prof. Dr. Md. Abdur Razzaque**  
Professor, CSE, University of Dhaka
5. Chapter 5: **Cloud Technology**  
**Dr. Kazi Muheymin Sakib**  
Director and Associate Professor, IIT, University of Dhaka  
  
**Asif Imran**  
Lecturer, IIT, University of Dhaka
6. Chapter 6: **Cloud Computing in Business**  
**Syed Zubair Hossain**  
CEO, Divine IT Limited
7. Chapter 7: **Cloud Infrastructure-Openstack**  
**Rana Khandkar**  
Team Lead, Newscred Inc.
8. Chapter 8: **Big Data**  
**Dr. Mohammad Rezwanul Huq**  
Assistant Professor  
Department of CSE, East West University (EWU)
9. Chapter 9: **Cloud Research challenges**  
**Dr. Md. Kamrul Hasan**  
Asst. Professor, CSE Department, Islamic University of Technology (IUT)

**CHAPTER 1**  
**INTRODUCTION TO CLOUD COMPUTING**

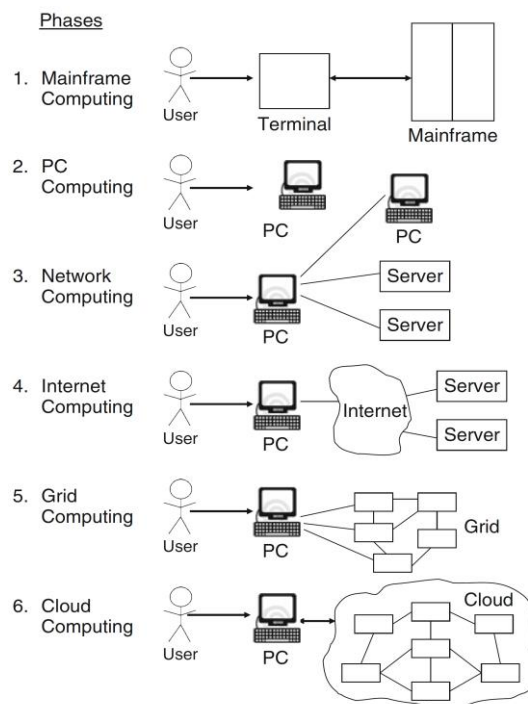
## 1. INTRODUCTION TO CLOUD COMPUTING

### 1.1 Introduction

Cloud computing can be defined as a new style of computing in which dynamically scalable and often virtualized resources are provided as a services over the Internet. Cloud computing has become a significant technology trend, and many experts expect that cloud computing will reshape information technology (IT) processes and the IT marketplace. With the cloud computing technology, users use a variety of devices, including PCs, laptops, smartphones, and PDAs to access programs, storage, and application-development platforms over the Internet, via services offered by cloud computing providers. Advantages of the cloud computing technology include cost savings, high availability, and easy scalability.

Fig.1.1, adapted from Voas and Zhang (2009), shows six phases of computing paradigms, from dummy terminals/mainframes, to PCs, networking computing, to grid and cloud computing. In phase 1, many users shared powerful mainframes using dummy terminals. In phase 2, stand-alone PCs became powerful enough to meet the majority of users' needs. In phase 3, PCs, laptops, and servers were connected together through local networks to share resources and increase performance. In phase 4, local networks were connected to other local networks forming a global network such as the Internet to utilize remote applications and resources. In phase 5, grid computing provided shared computing power and storage through a distributed computing system. In phase 6, cloud computing further provides shared resources on the Internet in a scalable and simple way.

Comparing these six computing paradigms, it looks like that cloud computing is a return to the original mainframe computing paradigm. However, these two paradigms have several important differences. Mainframe computing offers finite computing power, while cloud computing provides almost infinite power and capacity. In addition, in mainframe computing dummy terminals acted as user interface devices, while in cloud computing powerful PCs can provide local computing power and caching support.



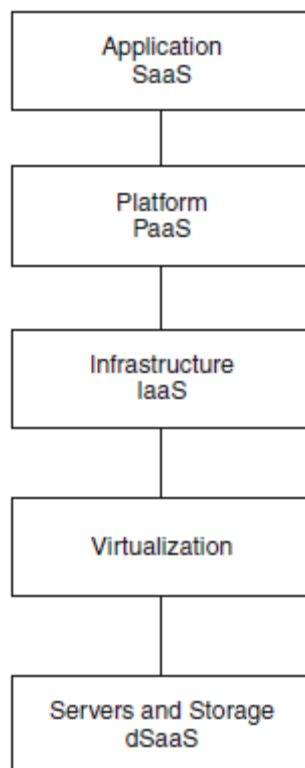
**Fig. 1.1** Six computing paradigms – from mainframe computing to Internet computing, to grid computing and cloud computing (adapted from Voas and Zhang (2009))

### 1.1.1 Layers of Cloud Computing

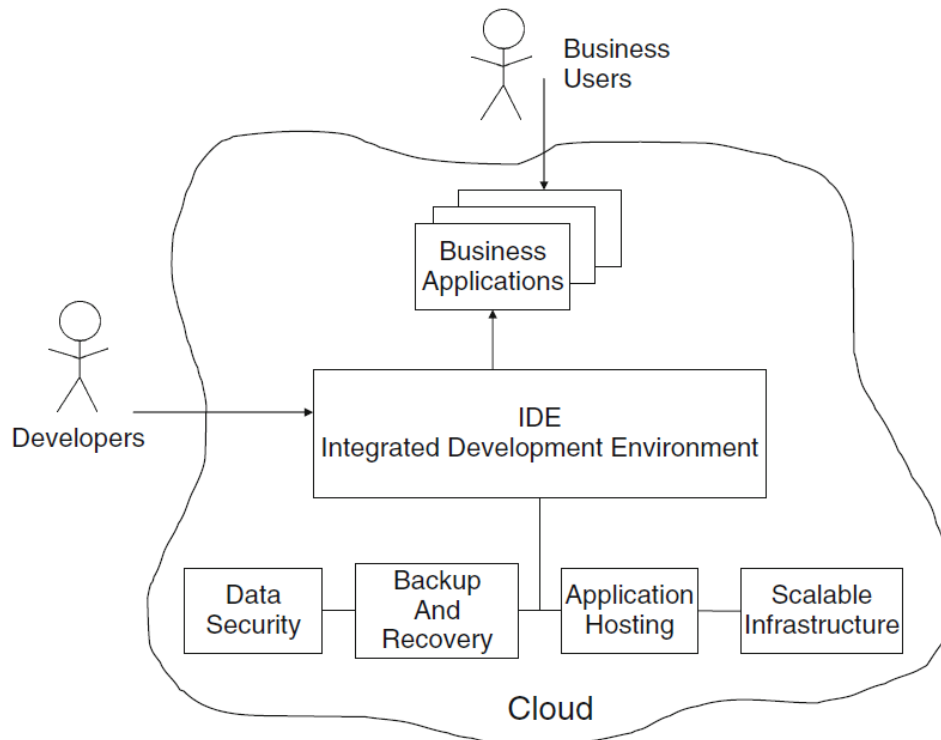
Cloud computing can be viewed as a collection of services, which can be presented as a layered cloud computing architecture, as shown in Fig. 1.2 [Jones]. The services offered through cloud computing usually include IT services referred as to SaaS (Software-as-a-Service), which is shown on top of the stack. SaaS allows users to run applications remotely from the cloud.

Infrastructure-as-a-service (IaaS) refers to computing resources as a service. This includes virtualized computers with guaranteed processing power and reserved bandwidth for storage and Internet access. Platform-as-a-Service (PaaS) is similar to IaaS, but also includes operating systems and required services for a particular application. In other words, PaaS is IaaS with a custom software stack for the given application. The data-Storage-as-a-Service (dSaaS) provides storage that the consumer is used including bandwidth requirements for the storage.

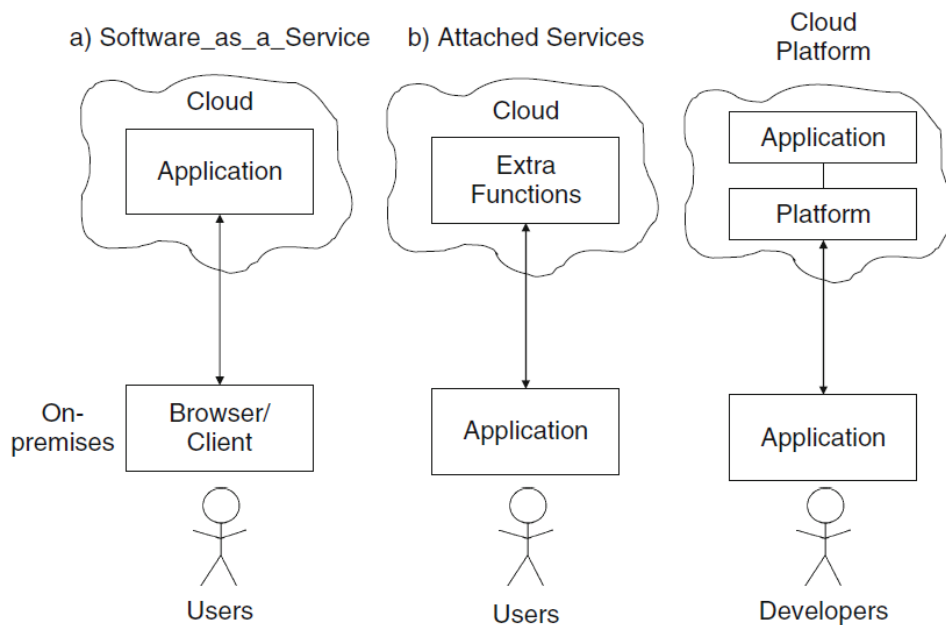
An example of Platform-as-a-Service (PaaS) cloud computing is shown in Fig. 1.3 [“Platform as a Service,” <http://www.zoho.com/creator/paas.html>]. The PaaS provides Integrated Development Environment (IDE) including data security, backup and recovery, application hosting, and scalable architecture. According to Chappell (2008) there are three categories of cloud services, as illustrated in Fig. 1.4. Fig. 1.4a shows the cloud service SaaS, where the entire application is running in the cloud. The client contains a simple browser to access the application. A well-known example of SaaS is salesforce.com. Fig. 1.4b illustrates another type of cloud services, where the application runs on the client; however it accesses useful functions and services provided in the cloud. An example of this type of cloud services on the desktop is Apple’s iTunes.



**Fig. 1.2** Layered architecture of Cloud Computing (adapted from Jones)



**Fig. 1.3** The concept of Platform-as-a-Service, Zoho Creator (adapted from “Platform as a Service,” <http://www.zoho.com/creator/paas.html>)



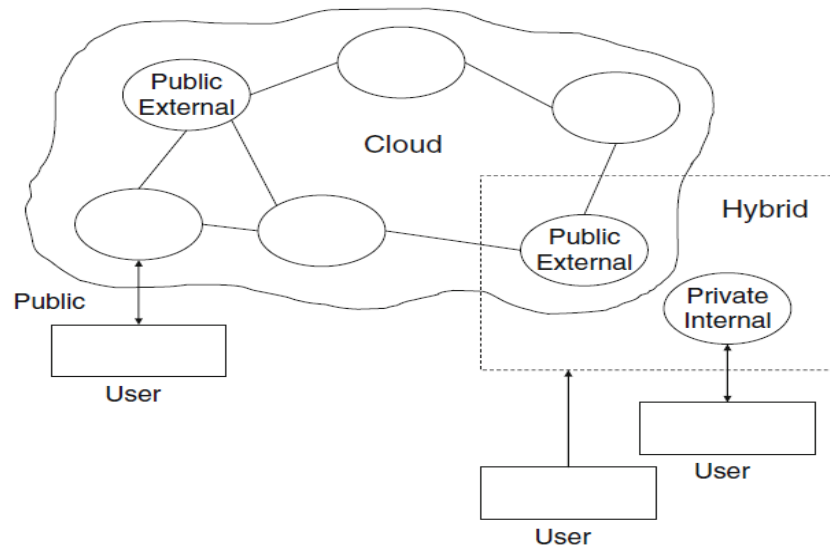
**Fig. 1.4** The categories of cloud services (adopted from Chappell (2008))

The desktop application plays music, while the cloud service is used to purchase a new audio and video content. An enterprise example of this cloud service is Microsoft Exchange Hosted Services. On-premises Exchange Server is using added services from the cloud including spam filtering, archiving, and other functions.

Finally, Fig. 1.4c shows a cloud platform for creating applications, which is used by developers. The application developers create a new SaaS application using the cloud platform.

### 1.1.2 Types of Cloud Computing

There are three types of cloud computing (a) public cloud, (b) private cloud, and (c) hybrid cloud, as illustrated in Fig. 1.5.



**Fig. 1.5** Three types of cloud computing

In the public cloud (or external cloud) computing resources are dynamically provisioned over the Internet via Web applications or Web services from an off-site third-party provider. Public clouds are run by third parties, and applications from different customers are likely to be mixed together on the cloud's servers, storage systems, and networks.

The email, calendar, photo, and music services used in the AV system analogy are all in the public cloud. Typically, a public cloud provider allows multiple organizations to provide multiple types of services (often referred to as *multitenancy*). The location for the underlying data center could be most anywhere in the world (often referred to as *location independence*). The underlying hardware is usually chosen by the cloud provider and not the users of the service (here you will likely find *virtualization* and *device independence*). The public cloud can also be described as an *external cloud* when viewed from within a given organization.

Private cloud (or internal cloud) refers to cloud computing on private networks. Private clouds are built for the exclusive use of one client, providing full control over data, security, and quality of service. Private clouds can be built and managed by a company's own IT organization or by a cloud provider.

A private cloud is restricted to one organization. Most often that organization is the single tenant—that is, unless the organization might want to host a private, multi-tenanted cloud for various internal segments or units of the organization. The data center for private clouds is managed by the organization. This can also be called an *internal cloud*.

A hybrid cloud environment combines multiple public and private cloud models. Hybrid clouds introduce the complexity of determining how to distribute applications across both a public and private cloud.

Nowadays there are few other types of cloud have been introduces. These are discussed below:

A community cloud is more restricted than a public cloud. The restriction is to a “community.” The restriction could be based on an industry segment, by general interest, or by whatever way a group might be defined. These clouds could be multi-tenanted. The underlying data center might be provided by a third party or by one member of the community.

A virtual private cloud involves some type of partitioning to ensure that the private cloud remains private. Typically, a virtual private cloud provider allows the definition of a network similar to a traditional network. Within such a network, it is possible to have systems such as database managements systems, business information (BI)/analytics systems, application servers, and so on.

### 1.1.3 Cloud Computing Versus Cloud Services

In this section we present two tables that show the differences and major attributes of cloud computing versus cloud services (Jens, 2008). Cloud computing is the IT foundation for cloud services and it consists of technologies that enable cloud services. The key attributes of cloud computing are shown in Table 1.1.

In Key attributes of cloud services are summarized in Table 1.2 (Jens, 2008).

**Table 1.1** Key Cloud Computing Attributes (adapted from Jens (2008))

<i>Attributes Description</i>	<i>Attributes Description</i>
Infrastructure systems	It includes servers, storage, and networks that can scale as per user demand.
Application software	It provides Web-based user interface, Web services APIs, and a rich variety of configurations.
Application development and deployment software	It supports the development and integration of cloud application software.
System and application management software	It supports rapid self-service provisioning and configuration and usage monitoring.
IP networks	They connect end users to the cloud and the infrastructure components.

**Table 1.2** Key Attributes of Cloud Services (adapted from Jens (2008))

Offsite. Third-party provider	In the cloud execution, it is assumed that third-party provides services. There is also a possibility of in-house cloud service delivery.
Accessed via the Internet	Services are accessed via standard-based, universal network access. It can also include security and quality-of-service options.
Minimal or no IT skill required	There is a simplified specification of requirements.
Provisioning	It includes self-service requesting, near real-time deployment, and dynamic and fine-grained scaling.
Pricing	Pricing is based on usage-based capability and it is fine-grained.
User interface	User interface include browsers for a variety of devices and with rich capabilities.
System interface	System interfaces are based on Web services APIs providing a standard framework for accessing and integrating among cloud services.
Shared resources	Resources are shared among cloud services users; however via configuration options with the service, there is the ability to customize.

### 1.1.4 Categories of Cloud Providers

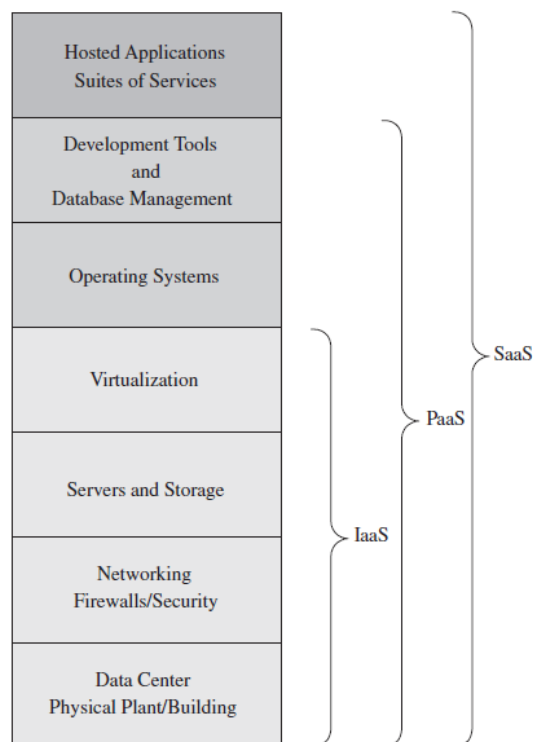
**Infrastructure as a Service (IaaS):** This contains the physical and virtual resources used to build the cloud. These cloud providers provision and manage the physical processing, storage, networking, and hosting environment. This is the data center or, in some cases, the data centers. Pricing is often based on resources used.

**Platform as a Service (PaaS):** This provides a complete computing platform. These cloud provider provision and manage cloud infrastructure as well as provide development, deployment, and administration tools. Here you will find the features that make a platform: operating systems, web servers, programming language, database management systems, and so on. This is where the provider might provide elasticity: the ability to scale up or scale down as needed. You will also find some level of reliability provided by the software platform. For example, some type of fault tolerance might be provided for the database management system. Pricing can be on many dimensions. For example, pricing could take into account the type of database management system, the level of activity, the amount of storage, and computational time/resources used.

**Software as a Service (SaaS):** This provides complete software systems. SaaS is a common way to provide applications such as email, calendars, CRM, social networks, content management, documentation management, and other office productivity applications. SaaS is also known as “on-demand software.” Pricing is often on per user basis, either monthly or yearly.

SaaS cloud providers are what most people mean when they refer to “the cloud.” They provide the services and related data that can be used directly or combined in some way with other SaaS providers or with your own unique data and services.

Fig. 1.6 illustrates the relationship of IaaS, PaaS, and SaaS in the cloud computing stack.



**Fig. 1.6** Cloud computing stack: IaaS, PaaS, and SaaS.

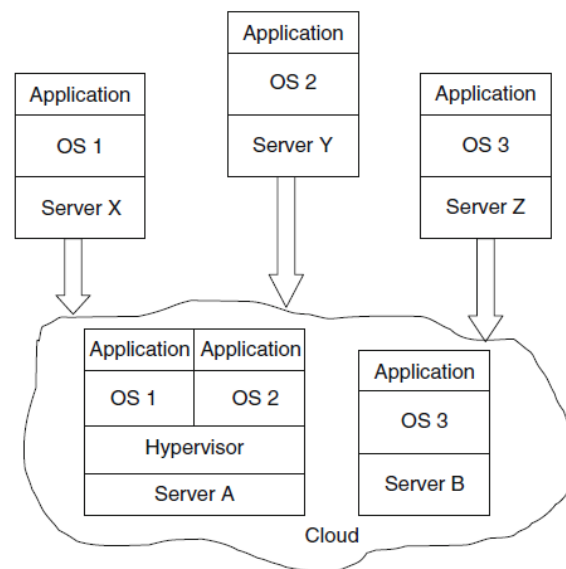


## 1.2 Enabling Technologies

Key technologies that enabled cloud computing are described in this section; they include virtualization, Web service and service-oriented architecture, service flows and workflows, and Web 2.0 and mashup.

### 1.2.1 Virtualization

The advantage of cloud computing is the ability to virtualize and share resources among different applications with the objective for better server utilization. Fig. 1.7 shows an example Jones]. In non-cloud computing three independent platforms exist for three different applications running on its own server. In the cloud, servers can be shared, or virtualized, for operating systems and applications resulting in fewer servers (in specific example two servers).



**Fig. 1.7** An example of virtualization: in non-cloud computing there is a need for three servers; in the cloud computing, two servers are used (adapted from Jones)

Virtualization technologies include virtual machine techniques such as VMware and Xen, and virtual networks, such as VPN. Virtual machines provide virtualized IT-infrastructures on-demand, while virtual networks support users with a customized network environment to access cloud resources.

### 1.2.2 Web Service and Service Oriented Architecture

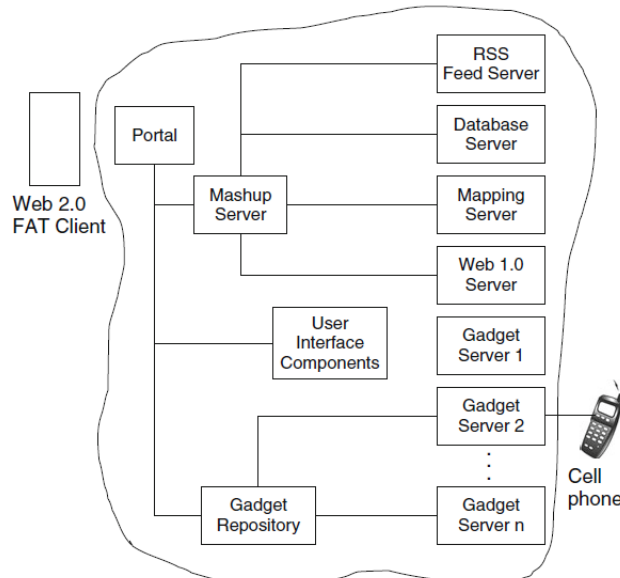
Web Services and Service Oriented Architecture (SOA) are not new concepts; however they represent the base technologies for cloud computing. Cloud services are typically designed as Web services, which follow industry standards including WSDL, SOAP, and UDDI. A Service Oriented Architecture organizes and manages Web services inside clouds (Vouk, 2008). A SOA also includes a set of cloud services, which are available on various distributed platforms.

### 1.2.3 Service Flow and Workflows

The concept of service flow and workflow refers to an integrated view of service based activities provided in clouds. Workflows have become one of the important areas of research in the field of database and information systems (Vouk, 2008).

### 1.2.4 Web 2.0 and Mashup

Web 2.0 is a new concept that refers to the use of Web technology and Web design to enhance creativity, information sharing, and collaboration among users (Wang, Tao, & Kunze, 2008). On the other hand, Mashup is a web application that combines data from more than one source into a single integrated storage tool. Both technologies are very beneficial for cloud computing.



**Fig. 1.8** Cloud computing architecture uses various components at different levels (adapted from Hutchinson and Ward (2009))

Fig. 1.8 shows a cloud computing architecture, adapted from Hutchinson and Ward (2009), in which an application reuses various components. The components in this architecture are dynamic in nature, operate in a SaaS model, and leverage SOA. The components closer to the user are smaller in nature and more reusable. The components in the center contain aggregate and extend services via mashup servers and portals. Data from one service (such as addresses in a database) can be mashed up with mapping information (such as Yahoo or Google maps) to produce an aggregated view of the information.

### 1.3 Cloud Computing Features

Cloud computing brings a number of new features compared to other computing paradigms (Wang et al., 2008; Grossman, 2009). There are briefly described in this section.

- Scalability and on-demand services  
Cloud computing provides resources and services for users on demand. The resources are scalable over several data centers.
- User-centric interface  
Cloud interfaces are location independent and can be accessed by well-established interfaces such as Web services and Internet browsers.
- Guaranteed Quality of Service (QoS)  
Cloud computed can guarantee QoS for users in terms of hardware/CPU performance, bandwidth, and memory capacity.
- Autonomous system

The cloud computing systems are autonomous systems managed transparently to users. However, software and data inside clouds can be automatically reconfigured and consolidated to a simple platform depending on user's needs.

- Pricing

Cloud computing does not require up-front investment. No capital expenditure is required. Users pay for services and capacity as they need them.

### 1.3.1 Cloud Computing Standards

Cloud computing standards have not been yet fully developed; however a number of existing typically lightweight, open standards have facilitated the growth of cloud computing ("Cloud Computing," Wikipedia, [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing)). Table 1.3 illustrates several of these open standards, which are currently used in cloud computing.

**Table 1.3** Cloud computing standards ("Cloud Computing," Wikipedia, [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing))

Applications	Communications: HTTP, XMPP Security: OAuth, OpenID, SSL/TLS Syndication: Atom
Client	Browsers: AJAX Offline: HTML5
Implementations	Virtualization: OVF
Platform	Solution stacks: LAMP
Service	Data: XML, JSON Web services: REST

Extensible Messaging and Presence Protocol (XMPP) is a communications protocol for message-oriented middleware based on XML (Extensible Markup Language). OAuth is an open standard to authorization. OAuth provides client applications a 'secure delegated access' to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. OpenID (OID) is an open standard and decentralized protocol by the non-profit OpenID Foundation that allows users to be authenticated by certain co-operating sites (known as Relying Parties or RP) using a third party service. SSL is short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL uses a cryptographic system that uses two keys to encrypt data – a public key known to everyone and a private or secret key known only to the recipient of the message.

Open Virtualization Format (OVF) is a packaging standard designed to address the portability and deployment of virtual appliances. LAMP is an acronym for an archetypal model of web service solution stacks, originally consisting of largely interchangeable components: Linux, the Apache HTTP Server, the MySQL relational database management system, and the PHP programming language. Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the W3C's XML 1.0 Specification and by several other related specifications, all of which are free open standards. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

## The Seven Standards of Cloud Computing Service Delivery

With the Force.com platform, salesforce.com has set the standard for cloud-computing service delivery. The Force.com platform adheres to the seven standards outlined below.

- ✓ **World-class security** – Provision world-class security at every level.
- ✓ **Trust and transparency** – Provide transparent, real-time, accurate service performance and availability information.
- ✓ **True multitenancy** – Deliver maximum scalability and performance to customers with a true multitenant architecture.
- ✓ **Proven scale** – Support millions of users with proven scalability.
- ✓ **High performance** – Deliver consistent, high-speed performance globally.
- ✓ **Complete disaster recovery** – Protect customer data by running the service on multiple, geographically dispersed data centers with extensive backup, data archive, and failover capabilities.
- ✓ **High availability** – Equip world-class facilities with proven high-availability infrastructure and application software.

Force.com is the *only* cloud-computing platform that adheres to all seven of these standards.

### 1.3.2 Cloud Computing Security

One of the critical issues in implementing cloud computing is taking virtual machines, which contain critical applications and sensitive data, to public and shared cloud environments. Therefore, potential cloud computing users are concerned about the following security issues (“Cloud Computing Security,” Third Brigade, [www.cloudreadysecurity.com](http://www.cloudreadysecurity.com)).

- Will the users still have the same security policy control over their applications and services?
  - Can it be proved to the organization that the system is still secure and meets SLAs?
  - Is the system complaint and can it be proved to company’s auditors?
- In traditional data centers, the common approaches to security include perimeter firewall, demilitarized zones, network segmentation, intrusion detection and prevention systems, and network monitoring tools.

The security requirements for cloud computing providers begins with the same techniques and tools as for traditional data centers, which includes the application of a strong network security perimeter. However, physical segmentation and hardware based security cannot protect against attacks between virtual machines on the same server. Cloud computing servers use the same operating systems, enterprise and Web applications as localized virtual machines and physical servers. Therefore, an attacker can remotely exploit vulnerabilities in these systems and applications. In addition, co-location of multiple virtual machines increases the attack surface and risk to MV-to-VM compromise. Intrusion detection and prevention systems need to be able to detect malicious activity in the VM level, regardless of the location of the VM within the virtualized cloud environment (“Cloud Computing Security,” Third Brigade, [www.cloudreadysecurity.com](http://www.cloudreadysecurity.com)).

In summary, the virtual environments that deploy the security mechanisms on virtual machines including firewalls, intrusion detection and prevention, integrity monitoring, and log inspection, will effectively make VM cloud secure and ready for deployment.

## 1.4 Cloud Computing Platforms

Cloud computing has great commercial potential. According to market research firm IDC, IT cloud services spending will grow from about \$16B in 2008 to about \$42B in 2012 and to increase its share of overall IT spending from 4.2% to 8.5%.

Table 1.4 presents key players in cloud computing platforms and their key offerings.

**Table 1.4** Key Players in Cloud Computing Platforms (adapted from Lakshmanan (2009))

Company	Cloud computing platform	Year of launch	Key offerings
Amazon.com	AWS (Amazon Web Services)	2006	Infrastructure as a service (Storage, Computing, Message queues, Datasets, Content distribution)
Microsoft	Azure	2009	Application platform as a service (.Net, SQL data services)
Google	Google App. Engine	2008	Web Application Platform as a service (Python run time environment)
IBM	Blue Cloud	2008	Virtualized Blue cloud data center
Salesforce.com	Force.com	2008	Proprietary 4GL Web application framework as an on Demand platform

## 1.5 Pricing

Pricing for cloud platforms and services is based on three key dimensions:

- i) storage,
- ii) bandwidth, and
- iii) compute.

*Storage* is typically measured as average daily amount of data stored in GB over a monthly period.

*Bandwidth* is measured by calculating the total amount of data transferred in and out of platform service through transaction and batch processing. Generally, data transfer between services within the same platform is free in many platforms.

*Compute* is measured as the time units needed to run an instance, or application, or machine to servicing requests. Table 6 compares pricing for three major cloud computing platforms.

**Table 1.5** Pricing comparison for major cloud computing platforms (adapted from “Which Cloud Platform is Right for You?,” [www.cumulux.com](http://www.cumulux.com).)

Resource	UNIT	Amazon	Google	Microsoft
Stored data	GB per month	\$0.10	\$0.15	\$0.15
Storage transaction	Per 10 K requests	\$0.10		\$0.10
Outgoing bandwidth	GB	\$0.10 – \$0.17	\$0.12	\$0.15
Incoming bandwidth	GB	\$0.10	\$0.10	\$0.10
Compute time	Instance Hours	\$0.10 – \$1.20	\$0.10	\$0.12

In summary, by analyzing the cost of cloud computing, depending on the application characteristics the cost of deploying an application could vary based on the selected platform. From Table 1.5, it seems that the unit pricing for three major platforms is quite similar. Besides unit pricing, it is important to translate it into monthly application development, deployments and maintenance costs.

### 1.6 Cloud Computing Components and Their Vendors

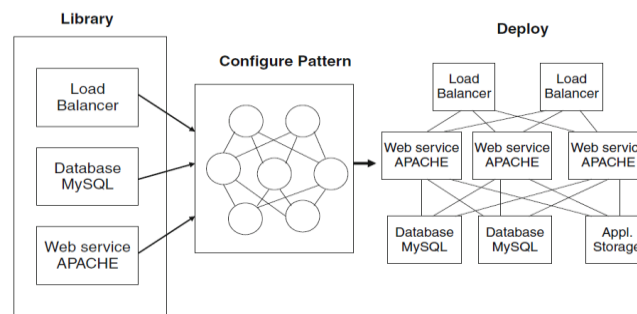
The main elements comprising cloud computing platforms include computer hardware, storage, infrastructure, computer software, operating systems, and platform virtualization. The leading vendors providing cloud computing components are shown in Table 1.6 (“Cloud Computing,” Wikipedia, [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing)).

**Table 1.6** The leading vendors of cloud computing components

Cloud computing components	Vendors
Computer hardware	Dell, HP, IBM, Sun
Storage	Sun, EMC, IBM
Infrastructure	Cisco, Juniper Networks, Brocade Communication
Computer Software	3tera, Eucalyptus, G-Eclipse, Hadoop
Operating Systems	Solaris, AIX, Linux (Red Hat, Ubuntu)
Platform virtualization	Citrix, VMWare, IBM, Xen, Linux KVM, Microsoft, Sun xVM

### 1.7 Example of Web Application Deployment

In this section we present an example how the combination of virtualization and on of self-service facilitate application deployment (Sun Microsystems, 2009). In this example we consider a two-tier Web application deployment using cloud, as illustrated in Fig. 1.9.



**Fig. 1.9** Example of the deployment of an application into a two-tier Web server architecture using cloud computing (adapted from Sun Microsystems (2009))

The following steps comprise the deployment of the application:

- The developer selects a load balancer, Web server, and database server appliances from a library of preconfigured virtual machine images.
- The developer configures each component to make a custom image. The load balancer is configured, the Web server is populated with its static content by uploading it to the storage cloud, and the database server appliances are populated with dynamic content for the site.
- The developer then layers custom code into the new architecture, in this way making the components meet specific application requirements.

- The developer chooses a pattern that takes the images for each layer and deploys them, handling networking, security, and scalability issues.

The secure, high-availability Web application is up and running. When the application needs to be updated, the virtual machine images can be updated, copied across the development chain, and the entire infrastructure can be redeployed.

In this example, a standard set of components can be used to quickly deploy an application. With this model, enterprise business needs can be met quickly, without the need for the time-consuming, manual purchase, installation, cabling, and configuration of servers, storage, and network infrastructure.

Small and medium enterprises were the early adopters to cloud computing. However, there are recently a number of examples of cloud computing adoptions in the large enterprises. Table 1.7 illustrates three examples of cloud computing use in the large enterprises (Lakshmanan, 2009).

**Table 1.7** Cloud computing examples in large enterprises

Enterprise	Scenario	Usage	Benefits
Eli Lilly	R&D Performance Computing High	Amazon server and storage cluster for drug discovery analysis and modeling.	Quick deployment time at a lower cost.
New York Times	Data Conversion	Conversion of archival articles (3 million) into new data formats using Amazon elastic compute services.	Rapid provisioning and higher elasticity on the infrastructure resources.
Pitney Bowes	B2B Application	Hosted model mail printing application for clients. Uses MS Azure.net and SQL services for the hosted model option (2009 Go live).	Flexibility at a lower cost and new business opportunity.

## 1.8 Cloud Computing Challenges

In summary, the new paradigm of cloud computing provides a number of benefits and advantages over the previous computing paradigms and many organizations are adopting it. However, there are still a number of challenges, which are currently addressed by researchers and practitioners in the field (Leavitt, 2009). They are briefly presented below.

### 1.8.1 Performance

The major issue in performance can be for some intensive transaction-oriented and other data-intensive applications, in which cloud computing may lack adequate performance. Also, users who are at a long distance from cloud providers may experience high latency and delays.

### 1.8.2 Security and Privacy

Companies are still concerned about security when using cloud computing. Customers are worried about the vulnerability to attacks, when information and critical IT resources are outside the firewall. The solution for security assumes that that cloud computing providers follow standard security practices, as described in Section 1.3.2.

### 1.8.3 Control

Some IT departments are concerned because cloud computing providers have a full control of the platforms. Cloud computing providers typically do not design platforms for specific companies and their business practices.

### 1.8.4 Bandwidth Costs

With cloud computing, companies can save money on hardware and software; however they could incur higher network bandwidth charges. Bandwidth cost may be low for smaller Internet-based applications, which are not data intensive, but could significantly grow for data-intensive applications.

### 1.8.5 Reliability

Cloud computing still does not always offer round-the-clock reliability. There were cases where cloud computing services suffered a few-hours outages. In the future, we can expect more cloud computing providers, richer services, established standards, and best practices. In the research arena, HP Labs, Intel, and Yahoo have launched the distributed Cloud Research Test Bed, with facilities in Asia, Europe, and North America, with the objective to develop innovations including cloud computing specific chips. IBM has launched the Research Computing Cloud, which is an on-demand, globally accessible set of computing resources that support business processes.

## 1.9 An Overview of Network Architectures for Clouds

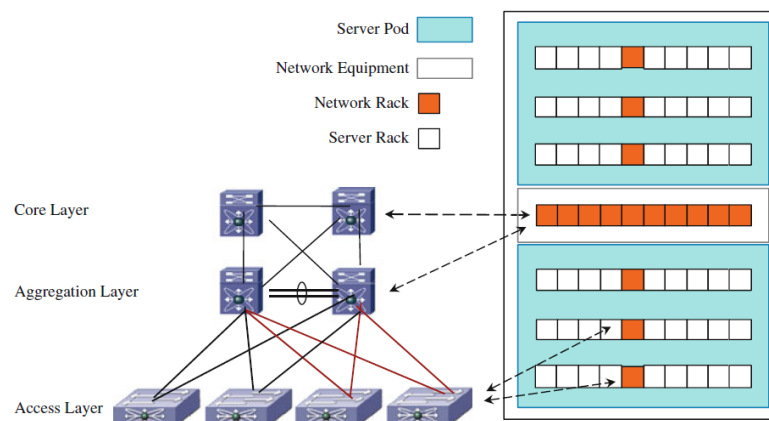
There are three principal areas in which the network architecture is of importance to cloud computing:

- (1) *Data center network*
- (2) *Data center interconnect network,*
- (3) *Pubic Internet*

We shall focus only on the first two areas in this chapter.

### 1.9.1 Data Center Network

Cloud providers offer scalable cloud services via massive data centers. In such massive-scale data centers, *Data Center Network* (DCN) is constructed to connect tens, sometimes hundreds, of thousands of serves to deliver massively scalable cloud services to the public. Hierarchical network design is the most common architecture used in data center networks. Fig. 1.10 show a conceptual view of a hierarchical data center network as well as an example of mapping the reference architecture to a physical data center deployment.



**Fig. 1.10** Data center network architecture



The *access layer* of a data center network provides connectivity for server resource pool residing in the data center. Design of the access layer is heavily influenced by the decision criteria such as server density, form factor, and server virtualization that can result in higher interface count requirements. The commonly used approaches for data center access layer connectivity are *end-of-row* (EoR) switch, *top-of-rack* (ToR) switch, and *integrated switch* (typically in the form of blade switches inside a modular blade server chassis). Another form of the integrated switch is the embedded *software switch* in a server end point. Each design approach has pros and cons, and is dictated by server hardware and application requirements. The *aggregation layer* of the data center provides a consolidation point where access layer switches are connected providing connectivity between servers for multi-tier applications, as well as connectivity across the core of the network to the clients residing within the campus, WAN, or Internet. The aggregation layer typically provides the boundary between Layer-3 routed links and Layer-2 Ethernet broadcast domains in the data center.

The primary function of the *core layer* in a data center network is to provide highly available, high performance Layer-3 switching for IP traffic between the data center and the Telco's Internet edge and backbone. In some situations, multiple geographically distributed data centers owned by a cloud service provider may be connected via a private WAN or a Metropolitan Area Network (MAN). For such environments, expanding Layer 2 networks across multiple data centers is a better architecture design. In other situations, the traffic has to be carried over the public Internet. The typical network topologies for this kind of geographically distributed data centers is Layer-3 Peering Routing between the data center core switches. By configuring all links connecting to the network core as point-to-point Layer-3 connections, rapid convergence around any link failure is provided, and the control plane of the core switches is not exposed to broadcast traffic from end node devices or required to participate in STP for Layer-2 network loop prevention.

The evolution of networking technology to support large-scale data centers is most evident at the access layer due to rapid increase of number of servers in a data center. A more prevalent approach is based on the concept of *switch virtualization* which allows the function of the logical Layer-2 access layer to span across multiple physical devices. There are several architectural variations in implementing switch virtualization at the access layer.

### 1.9.2 Data Center Interconnect Network

*Data center interconnect networks* (DCIN) are used to connect multiple data centers to support a seamless customer experience of cloud services. While a conventional, statically provisioned virtual private network can interconnect multiple data centers and offer secure communications, to meet the requirements of seamless user experience for cloud services (high-availability, dynamic server migration, application mobility), the DCIN for cloud services has emerged as a special class of networks based on the design principle of *Layer 2 network extension across multiple data centers* (Cisco Systems, 2009b). For example, in the case of server migration (either in a planned data center maintenance scenario or in an unplanned dynamic application workload balancing scenario) when only part of the server pool is moved at any given time, maintaining the Layer 2 adjacency of the entire server pool across multiple data centers as opposed to renumbering IP addresses of servers is a much better solution. The Layer 2 network extension approach, on one hand, is a must from business-continuity perspective; on the other hand, is cost effective from the operations perspective because it maintains the same server configuration and operations policies.

Among the chief technical requirements and use cases for data center interconnect networks are data center disaster avoidance (including data center maintenance without downtime), dynamic virtual server migration, high-availability clusters, and dynamic workload balancing and application mobility across multiple sites. These are critical requirements for cloud computing. Take the application mobility as an example. It provides the foundation necessary to enable compute elasticity – a key characteristics of cloud computing – by providing the flexibility to move virtual machines between different data centers. Fig. 4.3 shows a high level architecture for the data center interconnect network

based on the Layer 2 network extension approach. Since the conventional design principle for Layer 2 network is to reduce its diameter to increase performance and manageability (usually limiting it to the access layer, hence advocating consolidating servers to a single mega data center and limiting the Layer 2 connectivity to intra data center communications), there are many areas of improvement and further research needed to meet the needs of data center interconnect networks. Listed below are some of the key requirements for Layer 2 network extension across multiple data centers.

### End-to-End Loop Prevention

To improve the high availability of the Layer 2 VLAN when it extends between data centers, this interconnection must be duplicated. Therefore, an algorithm must be enabled to control any risk of a Layer 2 loop and to protect against any type of global disruptions that could be generated by a remote failure. An immediate option to consider is to leverage Spanning Tree Protocol (STP), but it must be isolated between the remote sites to mitigate the risk of propagating unwanted behaviors such as topology change or root bridge movement from one data center to another.

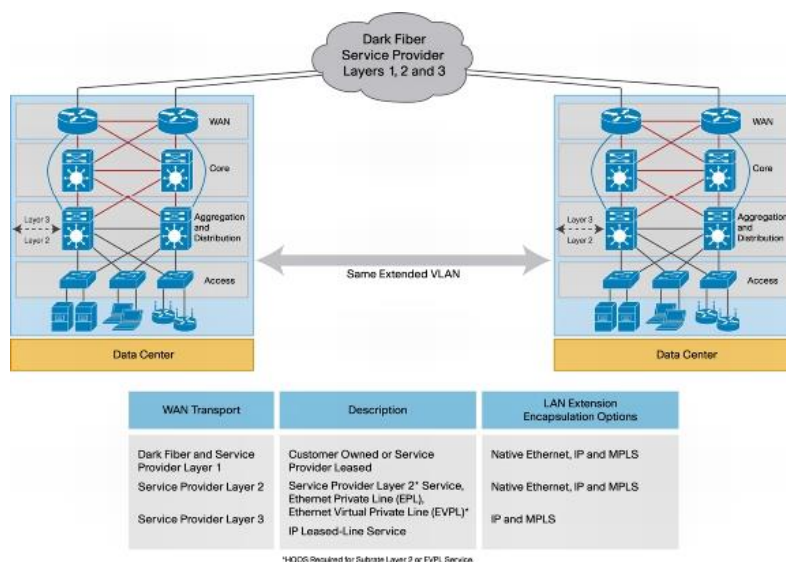


Fig. 1.11 Data center interconnect LAN extension encapsulation options

### WAN Load Balancing

Typically, WAN links are expensive, so the uplinks need to be fully utilized, with traffic load-balanced across all available uplinks. A mechanism to dynamically balance workloads at the virtual machine level is an area of research.

### Core Transparency

The LAN extension solution needs to be transparent to the existing enterprise core network, if available, to reduce any effect on operations. This is more common in the private cloud or hybrid cloud environments than in a public cloud.

### Encryption

The requirement for LAN extension cryptography is increasingly prevalent, for example, to meet the needs for cloud services and for federal and regulatory requirements.

### 1.9.3 Network Architecture for Hybrid Cloud Deployments

Hybrid clouds play a key role in the adoption of cloud computing as the new generation IT paradigm. While the IT industry and the research community are still in the early stage to understand the implementation technologies for hybrid clouds, a number of major functional components in the hybrid cloud network architecture have been identified. Fig. 1.12 shows a functional view of the network architecture for hybrid clouds.

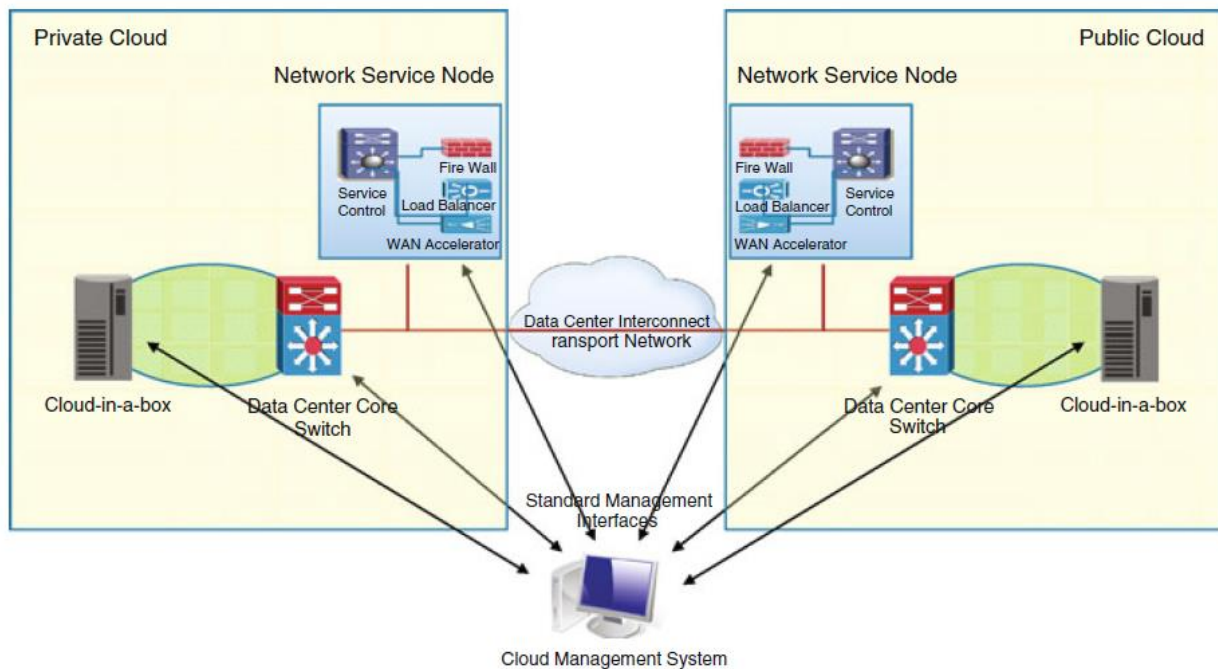


Fig. 1.12 A Functional view of network architecture for hybrid clouds

### 1.9.4 Cloud-in-a-Box

A new trend in the design and deployment of private and hybrid clouds is the concept of “*cloud-in-a-box*.” A cloud-in-a-box, sometimes also called a *cloud cell*, is a pre-integrated, prepackaged and self-contained service delivery platform that can be used easily and quickly to implement private cloud centers. Physically, it is typically delivered in a single chassis containing multiple blades; some blades are computing units, some switching units, and some storage units. They are interconnected by a combination of a common backplane and high-speed converged Ethernet connections. From the networking perspective, the switches that are pre-integrated into a cloud-in-a-box are typically the access layer switches.

Software wise, a common hypervisor environment typically expands across the computing units, the networking units, and storage units in a cloud-in-a-box device. From the networking perspective, this requires a virtual Ethernet switch to be embedded in the hypervisor. In the VMware environment, the VMware’s vNetwork Distributed Switch and Cisco’s Nexus 1000v virtual switch are the two well-known examples of hypervisor-embedded virtual Ethernet switches. On top of the common virtualization layer, a service management application is typically included to allow the management and automation of cloud services provisioning, accounting and billing, security, dynamic resource reallocation and workload mobility. Furthermore, some of today’s purpose-built cloud-in-a-box platforms also include a cloud service application to offer the specific cloud service. For example, a development-and-test oriented cloud-in-a-box platform may pre-integrate and prepackage a cloud-ready Integrated Development Environment (IDE) as part of the product. At the time of this chapter is written, there are a number of cloud-in-a-box products offered in the industry. See (VCEC, 2009; IBM Corporation, 2009) for further information.

## 1.10 Conclusions and Future Directions

In summary, cloud computing is definitely a type of computing paradigm/architecture that will remain for a long time to come. In the near future, cloud computing can emerge in various directions.

As the next paradigm shift for IT industry, cloud computing is still in the early stage. Just as the previous major IT paradigm shift – from centralized computing to distributed computing – has had tremendous impact on IP networking (and vice versa), we see a similar impact with regard to cloud computing and the next generation networks. In many ways, supporting cloud computing represents a natural evolution for the IP networking; we see the Layer 2 domain in the data center network becoming wider, flatter and virtualization aware; we see the data center interconnect network and Layer 4 network services becoming virtualization aware and self-adaptable to security, performance and SLA constraints; we see virtual machine mobility and cloud service elasticity not only within a single data center but also over metro networks or WAN across multiple data centers. As the IT industry creates and deploys more cloud services, more requirements will be put on to the networks and more intelligence will be implemented by the cloud-enabling network.

Our belief is that the hybrid cloud will emerge as the ideal cloud deployment model for most enterprises since it blends the best of private and public clouds. The networks play an extremely critical role in enabling hybrid cloud deployments. For core services with critical business data, the private network within the hybrid cloud can allow full control over network security, performance, management, etc. The public side of the hybrid cloud provides the ability to extend an enterprise's reach to Internet deployed applications and services which can then be integrated with its on-premise assets and business processes. We believe more cloud-enabling innovations will occur in both data center networks and data center interconnect networks. Furthermore, we believe the public Internet will embrace many of the capabilities exhibited in today's data center interconnect networks (and expand beyond). Somewhat contrary to today's loosely coupled IP networking architecture (with respect to other IT assets – servers, storage, and applications) which was the result from the distributed client server computing model, we believe the cloud computing model will drive a more tightly integrated network architecture with other IT assets Mell & Grance (October 2009).

## 1.11 References

1. Chappell, D. (August 2008). A short introduction to cloud platforms: An enterprise-oriented view. San Francisco, CA: Chappel and Associates.
2. Grossman, R. L. (March/April 2009). The case for cloud computing. *IEEE ITPro*, 23–27.
3. Hutchinson, C., & Ward, J. (March/April 2009). Navigation the next-generation application architecture. *IEEE ITPro*, 18–22.
4. Jens, F. (September 2008). *Defining cloud services and cloud computing*. <http://blogs.idc.com/ie/?p=190>.
5. Jones, M. T. *Cloud computing with linux*. [www.ibm.com/developerworks/linux/library/l-cloudcomputing](http://www.ibm.com/developerworks/linux/library/l-cloudcomputing).
6. Lakshmanan, G. (April 2009). *Cloud computing – Relevance to enterprise*. Infosys White Paper
7. Leavitt, N. (January 2009). Is cloud computing really ready for prime time? *IEEE Computer*, 15–20.
7. Sun Microsystems (June 2009). *Introduction to cloud computing architecture*. White Paper, Sun Microsystems.
8. Voas, J., & Zhang, J. (March/April 2009). Cloud computing: New wine or just a new bottle? *IEEE ITPro*, 15–17.
9. Vouk, M. A. (June 2008). Cloud computing – Issues, research and implementations. *Proceedings of the ITI 30th International Conference on Information Technology Interfaces, Cavtat, Croatia*, 31–40.
10. Wang, L., Tao, J., & Kunze, M. (2008). Scientific cloud computing: Early definition and experience. *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, Austin, TX*, 825–830.
11. Cisco Systems (May 2009a). *Security and virtualization in the data center*. [http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/white\\_paper\\_c11\\_493718.pdf](http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/white_paper_c11_493718.pdf).

12. Cisco Systems (July 2009b). *Data center interconnect: Layer 2 extension between remote data*.
13. [http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/white\\_paper\\_c11\\_493718.pdf](http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps708/white_paper_c11_493718.pdf).
14. Cisco Systems and VMware Inc. (August 2009c). *Virtual machine mobility with VMware vMotion and Cisco data center interconnect technologies*.  
[http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns836/white\\_paper\\_c11-557822.pdf](http://www.cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns836/white_paper_c11-557822.pdf).
15. Cisco Systems (October 2009d). *Data center design – IP network infrastructure*.
16. [http://www.cisco.com/en/US/docs/solutions/Enterprise/Data\\_Center/DC\\_3\\_0/DC-3\\_0\\_IPInfra.pdf](http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_3_0/DC-3_0_IPInfra.pdf).
17. Distributed Management Task Force (DMTFa). *DMTF open cloud standards incubator*.
18. <http://www.dmtf.org/about/cloud-incubator>. Accessed on February 2010.
19. Distributed Management Task Force (DMTFb) (February 2009). *Open virtualization format specification*.
20. [http://www.dmtf.org/standards/published\\_documents/DSP0243\\_1.0.0.pdf](http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf).
21. Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., & Sengupta, S. (August 2008). Towards a next generation data center architecture: Scalability and commoditization. *Proceedings of the ACM Workshop on Programmable Router and Extensible Services for Tomorrow (PRESTO)*, Seattle, WA, USA, 55–62.

[This chapter is a shortened version of the chapter 1: *Cloud Computing Fundamentals* and chapter 4: *The Role of Networks in Cloud Computing* of “*Handbook of Cloud Computing*, Editors: Borko Furht, Armando Escalante, Springer US, ISBN: 978-1-4419-6523-3 (Print) 978-1-4419-6524-0 (Online)”.]

**CHAPTER 2**  
**CASE STUDY: GOOGLE AS A CLOUD PROVIDER**

## 2. CASE STUDY: GOOGLE AS A CLOUD PROVIDER

### 2.1 Introduction

In this chapter we present a case study on the distributed systems infrastructure that underpins Google (hereafter referred to as the Google infrastructure). Google is one of the largest distributed systems in use today, and the Google infrastructure has successfully dealt with a variety of demanding requirements. The underlying architecture and choice of concepts are also very interesting. A study of the Google infrastructure therefore provides a perfect way to round off our study of distributed systems. Note that, as well as providing an example of how to support web search, the Google infrastructure has emerged as a leading example of cloud computing.

### 2.2 Introducing the case study: Google

Google is a US-based corporation with its headquarters in Mountain View, California (the Googleplex), offering Internet search and broader web applications and earning revenue largely from advertising associated with such services.

The name is a play on the word *googol*, the number  $10^{100}$  (or 1 followed by a hundred zeros), emphasizing the sheer scale of information available in the Internet today. Google's mission is to tame this huge body of information: 'to organize the world's information and make it universally accessible and useful'.

Google was born out of a research project at Stanford University, with the company launched in 1998. Since then, it has grown to have a dominant share of the Internet search market, largely due to the effectiveness of the underlying ranking algorithm used in its search. Significantly, Google has diversified, and as well as providing a search engine is now a major player in cloud computing.

Google provides a fascinating case study with extremely demanding requirements, particularly in terms of scalability, reliability, performance and openness. For example, in terms of search, it is noteworthy that the underlying system has successfully scaled with the growth of the company from its initial production system in 1998 to dealing with over 88 billion queries a month by the end of 2010, that the main search engine has never experienced an outage in all that time and that users can expect query results in around 0.2 seconds. Before proceeding to the case study, though, it is instructive to look in more detail at the search engine and also at Google as a cloud provider.

**The Google search engine:** The role of the Google search engine is, as for any web search engine, to take a given query and return an ordered list of the most relevant results that match that query by searching the content of the Web. The challenges stem from the size of the Web and its rate of change, as well as the requirement to provide the most relevant results from the perspective of its users. We provide a brief overview of the operation of Google search below; a fuller description of the operation of the Google search engine can be found in Langville and Meyer [2006]. As a running example, we consider how the search engine responds to the query 'Cloud Computing'. The underlying search engine consists of a set of services for crawling the Web and indexing and ranking the discovered pages, as discussed below.

**Crawling:** The task of the crawler is to locate and retrieve the contents of the Web and pass the contents onto the indexing subsystem. This is performed by a software service called Googlebot, which recursively reads a given web page, harvesting all the links from that web page and then scheduling further crawling operations for the harvested links (a technique known as *deep searching* that is highly effective in reaching practically all pages in the Web).

In the past, because of the size of the Web, crawling was generally performed once every few weeks. However, for certain web pages this was insufficient. For example, it is important for search engines to be able to report accurately on breaking news or changing share prices. Googlebot therefore took

note of the change history of web pages and revisited frequently changing pages with a period roughly proportional to how often the pages change. With the introduction of Caffeine in 2010 [googleblog.blogspot.comII], Google has moved from a batch approach to a more continuous process of crawling intended to offer more freshness in terms of search results. Caffeine is built using a new infrastructure service called Percolator that supports the incremental updating of large datasets [Peng and Dabek 2010].

**Indexing:** While crawling is an important function in terms of being aware of the content of the Web, it does not really help us with our search for occurrences of ‘Cloud Computing’. To understand how this is processed, we need to have a closer look at indexing. The role of indexing is to produce an index for the contents of the Web that is similar to an index at the back of a book, but on a much larger scale. More precisely, indexing produces what is known as an *inverted index* mapping words appearing in web pages and other textual web resources (including documents in *.pdf*, *.doc* and other formats) onto the positions where they occur in documents, including the precise position in the document and other relevant information such as the font size and capitalization (which is used to determine importance, as will be seen below). The index is also sorted to support efficient queries for words against locations.

As well as maintaining an index of words, the Google search engine also maintains an index of *links*, keeping track of which pages link to a given site. This is used by the PageRank algorithm, as discussed below. Let us return to our example query. This inverted index will allow us to discover web pages that include the search terms ‘Cloud’ and ‘Computing’ and, by careful analysis, we will be able to discover pages that include all of these terms. For example, the search engine will be able to identify that the two terms can all be found in amazon.com, www.cdk5.net and indeed many other web sites. Using the index, it is therefore possible to narrow down the set of candidate web pages from billions to perhaps tens of thousands, depending on the level of discrimination in the keywords chosen.

**Ranking:** The problem with indexing on its own is that it provides no information about the relative importance of the web pages containing a particular set of keywords – yet this is crucial in determining the potential relevance of a given page. All modern search engines therefore place significant emphasis on a system of ranking whereby a higher rank is an indication of the importance of a page and it is used to ensure that important pages are returned nearer to the top of the list of results than lower-ranked pages. As mentioned above, much of the success of Google can be traced back to the effectiveness of its ranking algorithm, PageRank [Langville and Meyer 2006].

**Anatomy of a search engine:** The founders of Google, Sergey Brin and Larry Page, wrote a seminal paper on the ‘anatomy’ of the Google search engine in 1998 [Brin and Page 1998], providing interesting insights into how their search engine was implemented. The overall architecture described in this paper is illustrated in Fig. 2.1, redrawn from the original. In this diagram, we distinguish between services directly supporting web search, drawn as ovals, and the underlying storage infrastructure components, illustrated as rectangles. While it is not the purpose of this chapter to present this architecture in detail, a brief overview will aid comparison with the more sophisticated Google infrastructure available today. The core function of crawling was described above. This takes as input lists of URLs to be fetched, provided by the URL server, with the resultant fetched pages placed into the store server. This data is then compressed and placed in the repository for further analysis, in particular creating the index for searching. The indexing function is performed in two stages. Firstly, the indexer uncompresses the data in the repository





<i>Application</i>	<i>Description</i>
Gmail	Mail system with messages hosted by Google but desktop-like message management.
Google Docs	Web-based office suite supporting shared editing of documents held on Google servers.
Google Sites	Wiki-like web sites with shared editing facilities.
Google Talk	Supports instant text messaging and Voice over IP.
Google Calendar	Web-based calendar with all data hosted on Google servers.
Google Wave	Collaboration tool integrating email, instant messaging, wikis and social networks.
Google News	Fully automated news aggregator site.
Google Maps	Scalable web-based world map including high-resolution imagery and unlimited user-generated overlays.
Google Earth	Scalable near-3D view of the globe with unlimited user-generated overlays.
Google App Engine	Google distributed infrastructure made available to outside parties as a service (platform as a service).

**Fig. 2.2** Example Google applications

Equally striking is that, as will become apparent below, the infrastructure has changed dramatically from the early attempts to identify an architecture for web search to the sophisticated distributed systems support provided today, both in terms of identifying more reusable building blocks for communication, storage and processing and in terms of generalizing the architecture beyond search.

**Google as a cloud provider** • Google has diversified significantly beyond search and now offers a wide range of web-based applications, including the set of applications promoted as Google Apps. More generally, Google is now a major player in the area of cloud computing. Cloud computing may be defined as ‘a set of Internet-based application, storage and computing services sufficient to support most users’ needs, thus enabling them to largely or totally dispense with local data storage and application software’. This is exactly what Google now strives to offer, in particular with significant offerings in the software as a service and platform as a service areas. We look at each area in turn below.

**Software as a service:** This area is concerned with offering application-level software over the Internet as web applications. A prime example is Google Apps, a set of web based applications including Gmail, Google Docs, Google Sites, Google Talk and Google Calendar. Google’s aim is to replace traditional office suites with applications supporting shared document preparation, online calendars, and a range of collaboration tools supporting email, wikis, Voice over IP and instant messaging.

**Platform as a service:** This area is concerned with offering distributed system APIs as services across the Internet, with these APIs used to support the development and hosting of web applications (note that the use of the term ‘platform’ in this context is unfortunately inconsistent with the way it is used elsewhere in this book, where it refers to the hardware and operating system level). With the launch of the Google App Engine, Google went beyond software as a service and now offers its distributed systems infrastructure as discussed throughout this chapter as a cloud service. More specifically, the Google business is already predicated on using this cloud infrastructure internally to support all its applications and services, including its web search engine. The Google App Engine now provides external access to a part of this infrastructure, allowing other organizations to run their own web applications on the Google platform.

We will see further details of the Google infrastructure as this chapter unfolds; refer to the Google web site for further details of the Google App Engine [code.google.com IV].

## 2.3 Overall architecture and design philosophy

This section looks at the overall architecture of the Google system, examining:

- ✓ Physical architecture adopted by Google;
- ✓ Associated system architecture that offers common services to the Internet search engine and the many web applications offered by Google.

### 2.3.1 Physical model

The key philosophy of Google in terms of physical infrastructure is to use very large numbers of commodity PCs to produce a cost-effective environment for distributed storage and computation. Purchasing decisions are based on obtaining the best performance per dollar rather than absolute performance with a typical spend on a single PC unit of around \$1,000. A given PC will typically have around 2 terabytes of disk storage and around 16 gigabytes of DRAM (dynamic random access memory) and run a cut-down version of the Linux kernel.

In electing to go down the route of commodity PCs, Google has recognized that parts of its infrastructure will fail and hence, as we will see below, has designed the infrastructure using a range of strategies to tolerate such failures. Hennessy and Patterson [2006] report the following failure characteristics for Google:

- By far the most common source of failure is due to software, with about 20 machines needing to be rebooted per day due to software failures. (Interestingly, the rebooting process is entirely manual.)
- Hardware failures represent about 1/10 of the failures due to software with around 2–3% of PCs failing per annum due to hardware faults. Of these, 95% are due to faults in disks or DRAM.

This vindicates the decision to procure commodity PCs; given that the vast majority of failures are due to software, it is not worthwhile to invest in more expensive, more reliable hardware. A further paper by Pinheiro *et al.* [2007] also reports on the failure characteristics of commodity disks as used in the Google physical infrastructure, providing an interesting insight into failure patterns of disk storage in large-scale deployments.

The physical architecture is constructed as follows [Hennessy and Patterson 2006]:

- Commodity PCs are organized in racks with between 40 and 80 PCs in a given rack. The racks are double-sided with half the PCs on each side. Each rack has an Ethernet switch that provides connectivity across the rack and also to the external world. This switch is modular, organized as a number of blades with each blade supporting either 8 100-Mbps network interfaces or a single 1-Gbps interface. For 40 PCs, five blades each containing eight network interfaces are sufficient to ensure connectivity within the rack. Two further blades, each supporting a 1-Gbps network interface, are used for connection to the outside world.
- Racks are organized into clusters, which are a key unit of management, determining for example the placement and replication of services. A cluster typically consists of 30 or more racks and two high-bandwidth switches providing connectivity to the outside world. Each rack is connected to both switches for redundancy; in addition, for further redundancy, each switch has redundant links to the outside world.
- Clusters are housed in Google data centres that are spread around the world. In 2000, Google relied on key data centres in Silicon Valley (two centres) and in Virginia. At the time of writing, the number of data centres has grown significantly and there are now centres in many geographical locations across the US and in Dublin (Ireland), Saint-Ghislain (Belgium), Zurich (Switzerland), Tokyo (Japan) and Beijing (China). (A map of known data centres as of 2008 can be found here [[royal.pingdom.com](http://royal.pingdom.com)].)

A simplified view of this overall organization is provided in Fig. 2.3. This physical infrastructure provides Google with enormous storage and computational capabilities, together with the necessary redundancy to build fault-tolerant, large-scale systems (note that, to avoid clutter, this figure only shows the Ethernet connections from one of the clusters to the external links).

**Storage capacity:** Let us consider the storage capacity available to Google. If each PC offers 2 terabytes of storage, then a rack of 80 PCs will provide 160 terabytes, with a cluster of 30 racks offering 4.8 petabytes. It is not known exactly how many machines Google has in total as the company maintains strict secrecy over this aspect of its business, but we can assume Google has on the order of 200 clusters, offering total storage capacity of 960 petabytes or just under 1 exabyte of storage (1018 bytes). This is likely to be a conservative figure, as Google VP Marissa Mayer is already talking about the data explosion taking us well into the exascale range [www.parc.com].

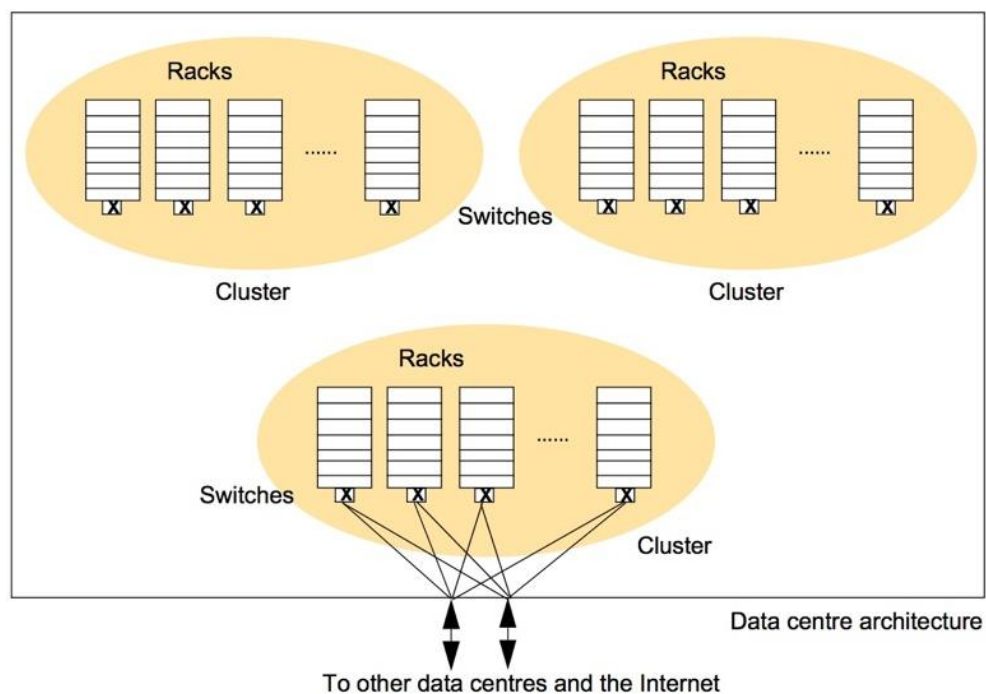


Fig. 2.3 Organization of the Google physical infrastructure

### 2.3.2 Overall system architecture

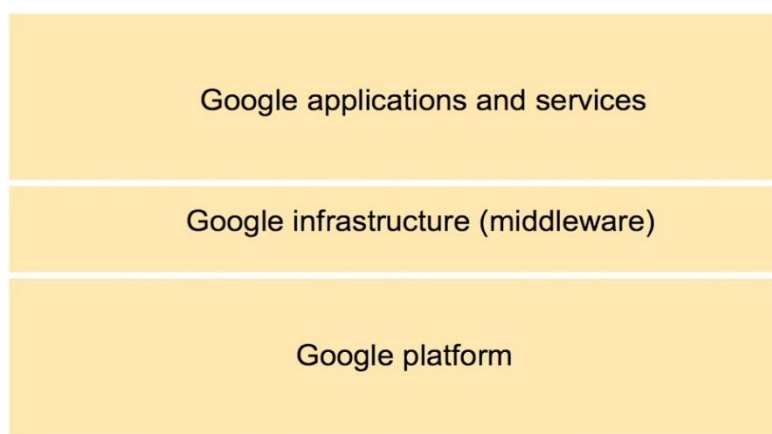
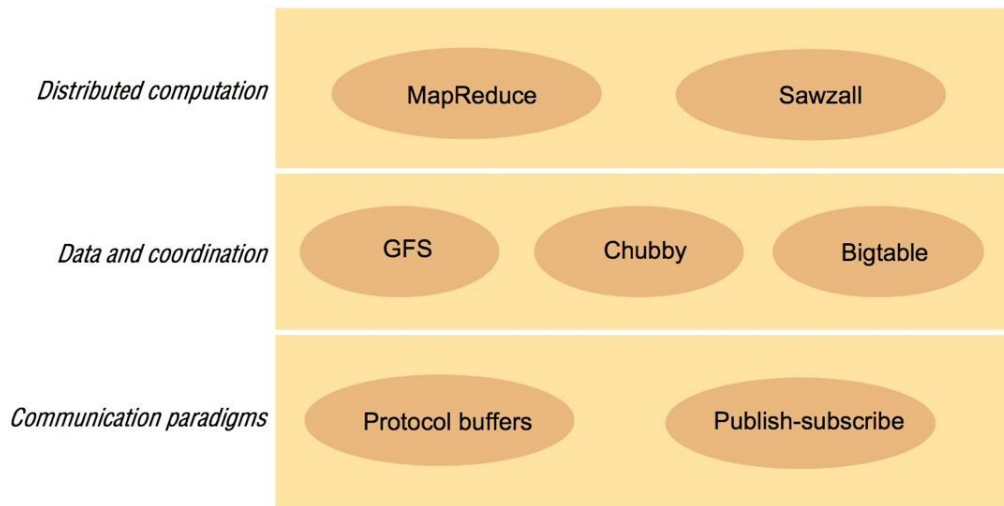


Fig. 2.4 The overall Google systems architecture



**Fig. 2.5** Google infrastructure

**Google infrastructure:** The system is constructed as a set of distributed services offering core functionality to developers (see Fig. 2.5). This set of services naturally partitions into the following subsets:

- the underlying communication paradigms, including services for both remote invocation and indirect communication:
  - the *protocol buffers* component offers a common serialization format for Google, including the serialization of requests and replies in remote invocation.
  - the Google *publish-subscribe* service supports the efficient dissemination of events to potentially large numbers of subscribers.
- data and coordination services providing unstructured and semi-structured abstractions for the storage of data coupled with services to support coordinated access to the data:
  - GFS offers a distributed file system optimized for the particular requirements of Google applications and services (including the storage of very large files).
  - Chubby supports coordination services and the ability to store small volumes of data.
  - Bigtable provides a distributed database offering access to semi-structured data.
- distributed computation services providing means for carrying out parallel and distributed computation over the physical infrastructure:
  - MapReduce supports distributed computation over potentially very large datasets (for example, stored in Bigtable).
  - Sawzall provides a higher-level language for the execution of such distributed computations.

With this background, we are now ready to examine the various constituent parts of the Google infrastructure, starting with the underlying communication paradigms. For each area, we present the overall design and highlight the key design decisions and associated trade-offs.

## 2.4 Underlying communication paradigms

It is clear that the choice of underlying communication paradigm(s) is crucial to the success of an overall system design. Options include:

- using an underlying interprocess communication service directly, such as that offered by socket abstractions.
- using a remote invocation service (such as a request-reply protocol, remote procedure calls or remote method invocation) offering support for client-server interactions;

- using an indirect communication paradigm such as group communication, distributed event-based approaches, tuple spaces or distributed shared memory approaches.

In keeping with the design principles identified in Section 2.3, Google adopts a simple, minimal and efficient remote invocation service that is a variant of a remote procedure call approach.

Readers will recall that remote procedure call communication requires a serialization component to convert the procedure invocation data (procedure name and parameters, possibly structured) from their internal binary representation to a *flattened* or *serialized* processor-neutral format ready for transmission to the remote partner. XML has emerged more recently as a ‘universal’ serialized data format, but its generality brings substantial overheads. Google has therefore developed a simplified and high-performance serialization component known as *protocol buffers* that is used for a substantial majority of interactions within the infrastructure. This can be used over any underlying communication mechanism to provide an RPC capability. An open source version of protocol buffers is available [code.google.com I].

A separate *publish-subscribe service* is also used, recognizing the key role that this paradigm can offer in many areas of distributed system design, including the efficient and real-time dissemination of events to multiple participants. In common with many other distributed system platforms, the Google infrastructure therefore offers a hybrid solution allowing developers to select the best communication paradigm for their requirements. Publish-subscribe is not an alternative to protocol buffers in the Google infrastructure, but rather an augmentation offering an added-value service for where it is most appropriate.

We examine the design of these two approaches below, with emphasis on protocol buffers (full details of the publish-subscribe protocol are not yet publicly available).

### 2.4.1 Remote invocation

Protocol buffers place emphasis on the description and subsequent serialization of data, and hence the concept is best compared to direct alternatives such as XML. The goal is to provide a language- and platform-neutral way to specify and serialize data in a manner that is simple, highly efficient and extensible; the serialized data can then be used for subsequent storage of data or transmission using an underlying communications protocol, or indeed for any other purpose that demands a serialization format for structured data. We will see later how this can be used as the basis for RPC style exchange.

In protocol buffers, a language is provided for the specification of *messages*. We introduce the key features of this (simple) language by example, with Fig. 2.7 showing how a book message might be specified.

As can be seen, the overall *Book* message consists of a series of uniquely numbered fields, each represented by a field name and the type of the associated value. The type can be one of:

- a *primitive data type* (including integer, floating-point, boolean, string or raw bytes);
- an *enumerated type*;
- a *nested message* allowing a hierarchical structuring of data.

We can see examples of each in Fig. 2.6.

Fields are annotated with one of three labels:

- *required* fields must be present in the message;
- *optional* fields may be present in the message;
- *repeated* fields can exist zero or more times in the message (the developers of protocol buffers view this as a type of dynamically sized array).

```
message Book {
  required string title = 1;
  repeated string author = 2;
  enum Status {
    IN_PRESS = 0;
    PUBLISHED = 1;
    OUT_OF_PRINT = 2;
  }
  message BookStats {
    required int32 sales = 1;
    optional int32 citations = 2;
    optional Status bookstatus = 3 [default = PUBLISHED];
  }
  optional BookStats statistics = 3;
  repeated string keyword = 4;
}
```

**Fig. 2.6** Protocol buffers example

Again, we can see uses of each annotation in the *Book* message format illustrated in Fig. 2.6. The unique number (=1, =2 and so on) represents the tag that a particular field has in the binary encoding of the message.

This specification is contained in a *.proto* file and compiled by a *protoc* tool. The output of this tool is generated code that allows programmers to manipulate the particular message type, in particular assigning/extracting values to/from messages. In more detail, the *protoc* tool generates a *builder* class that provides *getter* and *setter* methods for each field together with additional methods to *test* if a method has been set and to *clear* a field to the associated null value. For example, the following methods would be generated for the *title* field:

```
public boolean hasTitle();
public java.lang.String getTitle();
public Builder setTitle(String value);
public Builder clearTitle();
```

The importance of the builder class is that while messages are immutable in protocol buffers, builders are mutable and are used to construct and manipulate new messages. For repeated fields the generated code is slightly more complicated, with methods provided to return a *count* of the number of elements in the associated list, to *get* or *set* specific fields in the list, to *append* a new element to a list and to add a set of elements to a list (the *addAll* method). We illustrate this by example by listing the methods generated for the *keyword* field:

```
public List<string> getKeywordList();
public int getKeywordCount();
public string getKeyword(int index);
public Builder setKeyword(int index, string value);
public Builder addKeyword(string value);
public Builder addAllKeyword(Iterable<string> value);
public Builder clearKeyword();
```

The generated code also provides a range of other methods to manipulate messages, including methods such as *toString* to provide a readable representation of the message (often used for debugging for example) and also a series of methods to parse incoming messages.

As can be seen, this is a very simple format compared to XML (for example, contrast the specification above with equivalent specifications in XML), and one that its developers claim is 3 to 10 times smaller than XML equivalents and 10 to 100 times faster in operation. The associated programming interface providing access to the data is also considerably simpler than equivalents for XML.

Note that this is a somewhat unfair comparison, for two reasons. Firstly, the Google infrastructure is a relatively closed system and hence, unlike, XML, it does not address interoperability across open

systems. Secondly, XML is significantly richer in that it generates self-describing messages that contain the data and associated metadata describing the structure of the messages. Protocol buffers do not provide this facility directly (although it is possible to achieve this effect, as described in the relevant web pages in a section on techniques [code.google.com II]). In outline, this is achieved by asking the *protoc* compiler to generate a *FileDescriptorSet* that contains self-descriptions of messages, and then including this explicitly in message descriptions. The developers of protocol buffers, though, emphasize that this is not seen as a particularly useful feature and that it is rarely used in the Google infrastructure code.

**Supporting RPC:** As mentioned above, protocol buffers are a general mechanism that can be used for storage or communication. The most common use of protocol buffers, however, is to specify RPC exchanges across the network, and this is accommodated with extra syntax in the language. Again, we illustrate the syntax by example:

```
service SearchService {  
  rpc Search (RequestType) returns (ResponseType);  
}
```

This code fragment specifies a service interface called *SearchService* containing one remote operation, *Search*, which takes one parameter of type *RequestType* and returns one parameter of type *ResponseType*. For example, the types could correspond to a list of keywords and a list of *Books* matching this set of keywords, respectively. The *protoc* compiler takes this specification and produces both an abstract interface *SearchService* and a stub that supports type-safe RPC-style calls to the remote service using protocol buffers.

As well as being language- and platform-neutral, protocol buffers are also agnostic with respect to the underlying RPC protocol. In particular, the stub assumes that implementations exist for two abstract interfaces *RpcChannel* and *RpcController*, the former offering a common interface to underlying RPC implementations and the latter offering a common control interface, for example, to manipulate settings associated with that implementation. A programmer must provide implementations of these abstract interfaces, effectively selecting the desired RPC implementation. For example, this could pass serialized messages using HTTP or TCP or could map onto one of a number of third-party RPC implementations available and linked from the protocol buffers site [code.google.com III].

Note that a service interface can support multiple remote operations, but each operation must adhere to the pattern of taking a single parameter and returning a single result (with both being protocol buffer messages). This is unusual compared to the designs of RPC and RMI systems – remote invocations can have an arbitrary number of parameters, and in the case of RMI the parameters or results can be objects or indeed object references. The rationale for having one request and one reply is to support extensibility and software evolution; whereas the more general styles of interface may change significantly over time, this more constrained style of interface is likely to remain more constant. This approach also pushes the complexity towards the data in a manner that is reminiscent of the REST philosophy, with its constrained set of operations and emphasis on manipulating resources.

#### 2.4.2 Publish-subscribe

Protocol buffers are used extensively but not exclusively as the communication paradigm in the Google infrastructure. To complement protocol buffers, the infrastructure also supports a publish-subscribe system intended to be used where distributed events need to be disseminated in real time and with reliability guarantees to potentially large numbers of recipients. As mentioned above, the publish-subscribe service is an augmentation to protocol buffers and indeed uses protocol buffers for its underlying communication.

One key use for the publish-subscribe system, for example, is to underpin the Google Ads system, recognizing that advertisements in Google are world-wide and that advertisement serving systems anywhere in the network need to know in a fraction of a second the eligibility of certain advertisements that can be shown in response to a query.



The RPC system described above would clearly be inappropriate and highly inefficient for this style of interaction, especially given the potentially large numbers of subscribers and the guarantees required by the associated applications. In particular, the sender would need to know the identity of all the other advertisement serving systems, which could be very large. RPCs would need to be sent to all the individual serving systems, consuming many connections and a great deal of associated buffer space at the sender, not to mention the bandwidth requirements of sending the data across wide area network links. A publish-subscribe solution, in contrast, with its inherent time and space uncoupling, overcomes these difficulties and also offers natural support for failure and recovery of subscribers.

Google has not made details of the publish-subscribe system publicly available. We therefore restrict our discussion to some high-level features of the system.

Google adopts a *topic-based* publish-subscribe system, providing a number of *channels* for event streams with channels corresponding to particular topics. A topic based system was chosen for its ease of implementation and its relative predictability in terms of performance compared to content-based approaches – that is, the infrastructure can be set up and tailored to deliver events related to a given topic. The downside is a lack of expressive power in specifying events of interest. As a compromise, the Google publish-subscribe system allows enhanced subscriptions defined not just by selecting a channel but also by selecting subsets of events from within that channel. In particular, a given event consists of a header, an associated set of keywords and a payload, which is opaque to the programmer. Subscription requests specify the channel together with a filter defined over the set of keywords. Channels are intended to be used for relatively static and coarse-grained data streams requiring high throughputs of events (at least 1- Mbps), so the added capability for expressing refined subscriptions using filters helps greatly. For example, if a topic generates less than this volume of data, it will be subsumed within another topic but identifiable by keyword.

The publish-subscribe system is implemented as a *broker overlay* in the form of a set of trees, where each tree represents a topic. The root of the tree is the publisher and the leaf nodes represent subscribers. When filters are introduced, they are pushed as far back in the tree as possible to minimize unnecessary traffic.

Unlike the publish-subscribe systems, there is a strong emphasis on both reliable and timely delivery:

- In terms of reliability, the system maintains redundant trees; in particular, two separate tree overlays are maintained per logical channel (topic).
- In terms of timely delivery, the system implements a quality of service management technique to control message flows. In particular, a simple rate control scheme is introduced based on an imposed rate limit enforced on a per user/ per topic basis. This replaces a more complex approach and manages the anticipated resource usage across the tree in terms of memory, CPU and message and byte rates. Trees are initially constructed and constantly re-evaluated according to a shortest path algorithm.

## 2.5 Data storage and coordination services

We now present the three services that together provide data and coordination services to higher-level applications and services: the Google File System, Chubby and Bigtable. These are complementary services in the Google infrastructure:

- The Google File System is a distributed file system. It offers access to unstructured data in the form of files, but optimized to the styles of data and data access required by Google (very large files, for example).
- Chubby is a multi-faceted service supporting, for example, coarse-grained distributed locking for coordination in the distributed environment and the storage of very small quantities of data, complementing the large-scale storage offered by the Google File System.

- Bigtable offers access to more structured data in the form of tables that can be indexed in various ways including by row or column. Bigtable is therefore a style of distributed database, but unlike many databases it does not support full relational operators (these are viewed by Google as unnecessarily complex and unscalable).

These three services are also interdependent. For example, Bigtable uses the Google File System for storage and Chubby for coordination. We look at each service in detail below.

### 2.5.1 The Google File System (GFS)

The Google File System (GFS) is a distributed file system; it offers similar abstractions but is specialized for the very particular requirements that Google has in terms of storage and access to very large quantities of data [Ghemawat *et al.* 2003]. These requirements led to very different design decisions from those made in NFS and AFS (and indeed other distributed file systems), as we will see below. We start our discussion of GFS by examining the particular requirements identified by Google.

**GFS requirements:** The overall goal of GFS is to meet the demanding and rapidly growing needs of Google's search engine and the range of other web applications offered by the company. From an understanding of this particular domain of operation, Google identified the following requirements for GFS (see Ghemawat *et al.* [2003]):

- The first requirement is that GFS must run reliably on the physical architecture discussed in Section 2.3.1 – that is a very large system built from commodity hardware. The designers of GFS started with the assumption that components will fail (not just hardware components but also software components) and that the design must be sufficiently tolerant of such failures to enable application-level services to continue their operation in the face of any likely combination of failure conditions.
- GFS is optimized for the patterns of usage within Google, both in terms of the types of files stored and the patterns of access to those files. The number of files stored in GFS is not huge in comparison with other systems, but the files tend to be massive. For example, Ghemawat *et al.* [2003] report the need for perhaps one million files averaging 100 megabytes in size, but with some files in the gigabyte range. The patterns of access are also atypical of file systems in general. Accesses are dominated by sequential reads through large files and sequential writes that append data to files, and GFS is very much tailored towards this style of access. Small, random reads and writes do occur (the latter very rarely) and are supported, but the system is not optimized for such cases. These file patterns are influenced, for example, by the storage of many web pages sequentially in single files that are scanned by a variety of data analysis programs. The level of concurrent access is also high in Google, with large numbers of concurrent appends being particularly prevalent, often accompanied by concurrent reads.
- GFS must meet all the requirements for the Google infrastructure as a whole; that is, it must scale (particularly in terms of volume of data and number of clients), it must be reliable in spite of the assumption about failures noted above, it must perform well and it must be open in that it should support the development of new web applications. In terms of performance and given the types of data file stored, the system is optimized for high and sustained throughput in reading data, and this is prioritized over latency. This is not to say that latency is unimportant, rather, that this particular component (GFS) needs to be optimized for high-performance reading and appending of large volumes of data for the correct operation of the system as a whole.

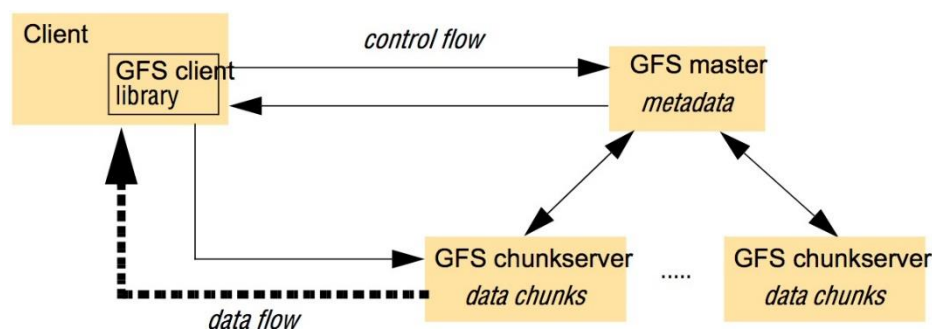
These requirements are markedly different from those for NFS and AFS (for example), which must store large numbers of often small files and where random reads and writes are commonplace. These distinctions lead to the very particular design decisions discussed below.

**GFS interface:** GFS provides a conventional file system interface offering a hierarchical namespace with individual files identified by pathnames. Although the file system does not provide full POSIX compatibility, many of the operations will be familiar to users of such file systems:

- create* – create a new instance of a file;
- delete* – delete an instance of a file;
- open* – open a named file and return a handle;
- close* – close a given file specified by a handle;
- read* – read data from a specified file;
- write* – write data to a specified file.

It can be seen that main GFS operations are very similar to those for the flat file service. We should assume that the GFS *read* and *write* operations take a parameter specifying a starting offset within the file, as is the case for the flat file service.

The API also offers two more specialized operations, *snapshot* and *record append*. The former operation provides an efficient mechanism to make a copy of a particular file or directory tree structure. The latter supports the common access pattern mentioned above whereby multiple clients carry out concurrent appends to a given file.



**Fig. 2.7** Overall architecture of GFS

**GFS architecture:** The most influential design choice in GFS is the storage of files in fixed-size *chunks*, where each chunk is 64 megabytes in size. This is quite large compared to other file system designs. At one level this simply reflects the size of the files stored in GFS. At another level, this decision is crucial to providing highly efficient sequential reads and appends of large amounts of data. We return to this point below, once we have discussed more details of the GFS architecture.

Given this design choice, the job of GFS is to provide a mapping from files to chunks and then to support standard operations on files, mapping down to operations on individual chunks. This is achieved with the architecture shown in Fig. 2.7, which shows an instance of a GFS file system as it maps onto a given physical cluster. Each GFS cluster has a single *master* and multiple *chunkservers* (typically on the order of hundreds), which together provide a file service to large numbers of clients concurrently accessing the data.

The role of the master is to manage metadata about the file system defining the namespace for files, access control information and the mapping of each particular file to the associated set of chunks. In addition, all chunks are replicated (by default on three independent chunkservers, but the level of replication can be specified by the programmer). The location of the replicas is maintained in the master. Replication is important in GFS to provide the necessary reliability in the event of (expected) hardware and software failures. This is in contrast to NFS and AFS, which do not provide replication with updates. The key metadata is stored persistently in an operation log that supports recovery in the event of crashes (again enhancing reliability). In particular, all the information mentioned above is logged apart from the location of replicas (the latter is recovered by polling chunk servers and asking them what replicas they currently store).

Although the master is centralized, and hence a single point of failure, the operations log is replicated on several remote machines, so the master can be readily restored on failure. The benefit of having such a single, centralized master is that it has a *global view* of the file system and hence it can make optimum management decisions, for example related to chunk placement. This scheme is also simpler to implement, allowing Google to develop GFS in a relatively short period of time. McKusick and Quinlan [2010] present the rationale for this rather unusual design choice.

When clients need to access data starting from a particular byte offset within a file, the GFS client library will first translate this to a file name and chunk index pair (easily computed given the fixed size of chunks). This is then sent to the master in the form of an RPC request (using protocol buffers). The master replies with the appropriate chunk identifier and location of the replicas, and this information is cached in the client and used subsequently to access the data by direct RPC invocation to one of the replicated chunk servers. In this way, the master is involved at the start and is then completely out of the loop, implementing a separation of control and data flows – a separation that is crucial to maintaining high performance of file accesses. Combined with the large chunk size, this implies that, once a chunk has been identified and located, the 64 megabytes can then be read as fast as the file server and network will allow without any other interactions with the master until another chunk needs to be accessed. Hence interactions with the master are minimized and throughput optimized. The same argument applies to sequential appends.

Note that one further repercussion of the large chunk size is that GFS maintains proportionally less metadata (if a chunk size of 64 kilobytes was adopted, for example, the volume of metadata would increase by a factor of 1,000). This in turn implies that GFS masters can generally maintain all their metadata in main memory (but see below), thus significantly decreasing the latency for control operations.

As the system has grown in usage, problems have emerged with the centralized master scheme:

- Despite the separation of control and data flow and the performance optimization of the master, it is emerging as a bottleneck in the design.
- Despite the reduced amount of metadata stemming from the large chunk size, the amount of metadata stored by each master is increasing to a level where it is difficult to actually keep all metadata in main memory.

For these reasons, Google is now working on a new design featuring a distributed master solution.

**Caching:** Caching often plays a crucial role in the performance and scalability of a file system. Interestingly, GFS does not make heavy use of caching. As mentioned above, information about the locations of chunks is cached at clients when first accessed, to minimize interactions with the master. Apart from that, no other client caching is used. In particular, GFS clients do not cache file data. Given the fact that most accesses involve sequential streaming, for example reading through web content to produce the required inverted index, such caches would contribute little to the performance of the system. Furthermore, by limiting caching at clients, GFS also avoids the need for cache coherency protocols.

GFS also does not provide any particular strategy for server-side caching (that is, chunk servers) rather relying on the buffer cache in Linux to maintain frequently accessed data in memory.

**Logging:** GFS is a key example of the use of *logging* in Google to support debugging and performance analysis. In particular, GFS servers all maintain extensive diagnostic logs that store significant server events and all RPC requests and replies. These logs are monitored continually and used in the event of system problems to identify the underlying causes.

**Managing consistency in GFS:** Given that chunks are replicated in GFS, it is important to maintain the consistency of replicas in the face of operations that alter the data – that is, the *write* and *record append* operations. GFS provides an approach for consistency management that:

- maintains the previously mentioned separation between control and data and hence allows high-performance updates to data with minimal involvement of masters;
- provides a relaxed form of consistency recognizing, for example, the particular semantics offered by *record append*.

The approach proceeds as follows.

When a mutation (i.e., a *write*, *append* or *delete* operation) is requested for a chunk, the master grants a chunk *lease* to one of the replicas, which is then designated as the *primary*. This primary is responsible for providing a serial order for all the currently pending concurrent mutations to that chunk. A global ordering is thus provided by the ordering of the chunk leases combined with the order determined by that primary. In particular, the lease permits the primary to make mutations on its local copies and to control the order of the mutations at the secondary copies; another primary will then be granted the lease, and so on.

The steps involved in mutations are therefore as follows (slightly simplified):

- On receiving a request from a client, the master grants a lease to one of the replicas (the primary) and returns the identity of the primary and other (secondary) replicas to the client.
- The client sends all data to the replicas, and this is stored temporarily in a buffer cache and not written until further instruction (again, maintaining a separation of control flow from data flow coupled with a lightweight control regime based on leases).
- Once all the replicas have acknowledged receipt of this data, the client sends a write request to the primary; the primary then determines a serial order for concurrent requests and applies updates in this order at the primary site. • The primary requests that the same mutations in the same order are carried out at secondary replicas and the secondary replicas send back an acknowledgement when the mutations have succeeded'
- If all acknowledgements are received, the primary reports success back to the client; otherwise, a failure is reported indicating that the mutation succeeded at the primary and at some but not all of the replicas. This is treated as a failure and leaves the replicas in an inconsistent state. GFS attempts to overcome this failure by retrying the failed mutations. In the worst case, this will not succeed and therefore consistency is not guaranteed by the approach.

GFS adopts a passive replication architecture with an important twist. In passive replication, updates are sent to the primary and the primary is then responsible for sending out subsequent updates to the backup servers and ensuring they are coordinated. In GFS, the client sends data to all the replicas but the request goes to the primary, which is then responsible for scheduling the actual mutations (the separation between data flow and control flow mentioned above). This allows the transmission of large quantities of data to be optimized independently of the control flow.

In mutations, there is an important distinction between *write* and *record append* operations. *writes* specify an offset at which mutations should occur, whereas *record appends* do not (the mutations are applied at the end of the file wherever this might be at a given point in time). In the former case the location is predetermined, whereas in the latter case the system decides. Concurrent *writes* to the same location are not serializable and may result in corrupted regions of the file. With *record append* operations, GFS guarantees the append will happen at least once and atomically (that is, as a contiguous sequence of bytes); the system does not guarantee, though, that all copies of the chunk will be identical (some may have duplicate data).

### 2.5.2 Chubby

Chubby [Burrows 2006] is a crucial service at the heart of the Google infrastructure offering storage and coordination services for other infrastructure services, including GFS and Bigtable. Chubby is a multi-faceted service offering four distinct capabilities:

- It provides coarse-grained distributed locks to synchronize distributed activities in what is a large-scale, asynchronous environment.
- It provides a file system offering the reliable storage of small files (complementing the service offered by GFS).
- It can be used to support the election of a primary in a set of replicas.
- It is used as a name service within Google.

At first sight, this might appear to contradict the overall design principle of simplicity (doing one thing and doing it well). As we unfold the design of Chubby, however, we will see that its heart is one core service that is offering a solution to *distributed consensus* and that the other facets emerge from this core service, which is optimized for the style of usage within Google.

We begin our study of Chubby by examining the interface it offers; then we look in detail at the architecture of a Chubby system and how this maps onto the physical architecture.

**Chubby interface:** Chubby provides an abstraction based on a file system, taking the view first promoted in Plan 9 [Pike *et al.* 1993] that every data object is a file. Files are organized into a hierarchical namespace using directory structures with names having the form of:

`/ls/chubby_cell/directory_name/.../file_name`

where `/ls` refers to the lock service, designating that this is part of the Chubby system, and `chubby_cell` is the name of a particular instance of a Chubby system (the term *cell* is used in Chubby to denote an instance of the system). This is followed by a series of directory names culminating in a *file\_name*. A special name, `/ls/local` will be resolved to the most local cell relative to the calling application or service.

Chubby started off life as a lock service, and the intention was that everything would be a lock in the system. However, it quickly became apparent that it would be useful to associate (typically small) quantities of data with Chubby entities – we see an example of this below when we look at how Chubby is used in primary elections. Thus entities in Chubby share the functionality of locks and files; they can be used solely for locking, to store small quantities of data or to associate small quantities of data (effectively metadata) with locking operations.

A slightly simplified version of the API offered by Chubby is shown in Fig. 2.8. *Open* and *Close* are standard operations, with *Open* taking a named file or directory and returning a Chubby *handle* to that entity. The client can specify various parameters associated with the *Open*, including declaring the intended usage (for example, for reading, writing or locking), and permissions checks are carried out at this stage using access control lists. *Close* simply relinquishes use of the handle. *Delete* is used to remove the file or directory (this operation fails if applied to a directory with children). In the role of a file system, Chubby offers a small set of operations for *whole-file* reading and writing; these are single operations that return the complete data of the file and write the complete data of the file. This whole-file approach is adopted to discourage the creation of large files, as this is not the intended use of Chubby. The first operation, *GetContentsAndStat*, returns both the contents of the file and any metadata associated with the file (an associated operation, *GetStat*, just returns the metadata; a *ReadDir* operation is also provided to read names and metadata associated with children of a directory. *SetContents* writes the contents of a file and *SetACL* provides a means to set access control list data. The reading and writing of whole files are *atomic* operations.

In the role of a lock-management tool, the main operations provided are *Acquire*, *TryAcquire* and *Release*. *Acquire* and *Release* correspond to the operations of the same name. *TryAcquire* is a non-blocking variant of *Acquire*. Note that although locks are *advisory* in Chubby an application or service must go through the proper protocol of acquiring and releasing locks. The developers of Chubby did consider an alternative of mandatory locks, whereby locked data is inaccessible to all other users and this is enforced by the system, but the extra flexibility and resilience of advisory locks was preferred, leaving the responsibility of checking for conflicts to the programmer [Burrows 2006].

Role	Operation	Effect
General	<i>Open</i>	Opens a given named file or directory and returns a handle
	<i>Close</i>	Closes the file associated with the handle
	<i>Delete</i>	Deletes the file or directory
File	<i>GetContentsAndStat</i>	Returns (atomically) the whole file contents and metadata associated with the file
	<i>GetStat</i>	Returns just the metadata
	<i>ReadDir</i>	Returns the contents of a directory – that is, the names and metadata of any children
	<i>SetContents</i>	Writes the whole contents of a file (atomically)
	<i>SetACL</i>	Writes new access control list information
Lock	<i>Acquire</i>	Acquires a lock on a file
	<i>TryAcquire</i>	Tries to acquire a lock on a file
	<i>Release</i>	Releases a lock

**Fig. 2.8** Chubby API

Should an application need to protect a file from concurrent access, it can use both the roles together, storing data in the file and acquiring locks before accessing this data.

Chubby can also be used to support a *primary election* in distributed systems – that is, the election of one replica as the primary in passive replication management. First, all candidate primaries attempt to acquire a lock associated with the election. Only one will succeed. This candidate becomes the primary, S with all other candidates then being secondaries. The primary records its victory by writing its identity to the associated file, and other processes can then determine the identity of the primary by reading this data. As mentioned above, this is a key example of combining the roles of lock and file together for a useful purpose in a distributed system. This also shows how primary election can be implemented on top of a consensus service as an alternative to the algorithms such as the ring-based approach or the bully algorithm.

Finally, Chubby supports a simple *event mechanism* enabling clients to register when opening a file to receive event messages concerning the file. More specifically, the client can subscribe to a range of events as an option in the *Open* call. The associated events are then delivered asynchronously via callbacks. Examples of events include that the file contents have been modified, a handle has become invalid, and so on.

Chubby is very much a cut-down file system programming interface compared to, for example, POSIX. Not only does Chubby require read and update operations to apply to whole files, but it does not support operations to move files between directories, nor does it support symbolic or hard links. Also Chubby maintains only limited metadata (related to access control, versioning and a checksum to protect data integrity).

**Chubby architecture:** As mentioned above, a single instance of a Chubby system is known as a cell; each cell consists of a relatively small number of replicas (typically five) with one designated as the master. Client applications access this set of replicas via a Chubby library, which communicates with

the remote servers using the RPC service described in Section 2.4.1. The replicas are placed at failure-independent sites to minimize the potential for correlated failures – for example, they will not be contained within the same rack. All replicas are typically contained within a given physical cluster, although this is not required for the correct operation of the protocol and experimental cells have been created that span Google data centres.

Each replica maintains a small database whose elements are entities in the Chubby namespace – that is, directories and files/locks. The consistency of the replicated database is achieved using an underlying consensus protocol (an implementation of Lamport’s Paxos algorithm [Lamport 1989, Lamport 1998]) that is based around maintaining *operation logs* (we look at the implementation of this protocol below). As logs can become very large over time, Chubby also supports the creation of *snapshots* – complete views of system state at a given point of time. Once a snapshot is taken, previous logs can be deleted with the consistent state of the system at any point determined by the previous snapshot together with the applications of the set of operations in the log. This overall structure is shown in Fig. 2.8.

A Chubby *session* is a relationship between a client and a Chubby cell. This is maintained using *KeepAlive* handshakes between the two entities. To improve performance, the Chubby library implements *client caching*, storing file data, metadata and information on open handles. In contrast to GFS (with its large, sequential reads and appends), client caching is effective in Chubby with its small files that are likely to be accessed repeatedly. Because of this caching, the system must maintain consistency between a file and a cache as well as between the different replicas of the file. The required *cache consistency* in Chubby is achieved as follows. Whenever a mutation is to occur, the associated operation (for example, *SetContents*) is blocked until all associated caches are invalidated (for efficiency, the invalidation requests are piggybacked onto *KeepAlive* replies from the master with the replies sent immediately when an invalidation occurs). Cached data is also never updated directly.

The end result is a very simple protocol for cache consistency that delivers deterministic semantics to Chubby clients. Contrast this with the client caching regime in NFS, for example, where mutations do not result in the immediate updating of cached copies, resulting in potentially different versions of files on different client nodes.

This determinism is important for many of the client applications and services that use Chubby to store access control lists. Big table requires consistent update of access control lists, across all replicas and in terms of cached copies. Note that it is this determinism that led to the use of Chubby as a name server within Google. The DNS allows naming data to become inconsistent. While this is tolerable in the Internet, the developers of the Google infrastructure preferred the more consistent view offered by Chubby, using Chubby files to maintain name to address mappings. Burrows [2006] discusses this use of Chubby as a name service in more detail.

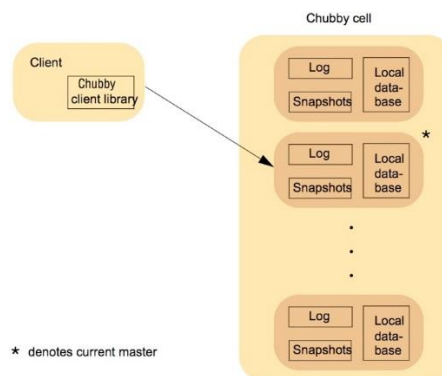


Fig. 2.8 Overall architecture of Chubby



## 2.6 Distributed computation services

To complement the storage and coordination services, it is also important to support high-performance distributed computation over the large datasets stored in GFS and Bigtable. The Google infrastructure supports distributed computation through the MapReduce service and also the higher-level Sawzall language. We look at MapReduce in detail and then briefly examine the key features of the Sawzall language.

### 2.6.1 MapReduce

Given the huge datasets in use at Google, it is a strong requirement to be able to carry out distributed computation by breaking up the data into smaller fragments and carrying out analyses of such fragments in parallel, making use of the computational resources offered by the physical architecture described in Section 2.3.1. Such analyses include common tasks such as sorting, searching and constructing inverted indexes (indexes that contain a mapping from words to locations in different files, this being key in implementing search functions). MapReduce [Dean and Ghemawat 2008] is a simple programming model to support the development of such applications, hiding underlying detail from the programmer including details related to the parallelization of the computation, monitoring and recovery from failure, data management and load balancing onto the underlying physical infrastructure.

We look at details of the programming model offered by MapReduce before examining how the system is implemented.

**MapReduce interface:** The key principle behind MapReduce is the recognition that many parallel computations share the same overall pattern – that is:

- break the input data into a number of chunks;
- carry out initial processing on these chunks of data to produce intermediary results;
- combine the intermediary results to produce the final output.

The specification of the associated algorithm can then be expressed in terms of two functions, one to carry out the initial processing and the second to produce the final results from the intermediary values. It is then possible to support multiple styles of computation by providing different implementations of these two functions. Crucially, by factoring out these two functions, the rest of the functionality can be shared across the different computations, thus achieving huge reductions in complexity in constructing such applications.

More specifically, MapReduce specifies a distributed computation in terms of two functions, *map* and *reduce* (an approach partially influenced by the design of functional programming languages such as Lisp, which provide functions of the same name, although in functional programming the motivation is not parallel computation):

- *map* takes a set of key-value pairs as input and produces a set of intermediary key value pairs as output.
- The intermediary pairs are then sorted by key value so that all intermediary results are ordered by intermediary key. This is broken up into groups and passed to *reduce* instances, which carry out their processing to produce a list of values for each group (for some computations, this could be a single value).

To illustrate the operation of MapReduce, let us consider a simple example. In section 2.2, we illustrated the various aspects of a web search for ‘*distributed systems book*’ Let us simplify this further by just searching for this complete string – that is, a search for the phrase ‘distributed systems book’ as it appears in a large body of content such as the crawled contents of the Web. In this example, the *map* and *reduce* functions would perform the following tasks:

- Assuming it is supplied with a web page name and its contents as input, the *map* function searches linearly through the contents, emitting a key-value pair consisting of (say) the phrase

followed by the name of the web document containing this phrase wherever it finds the strings 'distributed' followed by 'system' followed by 'book' (the example can be extended to also emit a position within the document).

- The *reduce* function in this case is trivial, simply emitting the intermediary results ready to be collated together into a complete index. The MapReduce implementation is responsible for breaking the data into chunks, creating multiple instances of the *map* and *reduce* functions, allocating and activating them on available machines in the physical infrastructure, monitoring the computations for any failures and implementing appropriate recovery strategies, dispatching intermediary results and ensuring optimal performance of the whole system.

With this approach, it is possible to make significant savings in terms of lines of code by reusing the underlying MapReduce framework. For example, Google reimplemented the main production indexing system in 2003 and reduced the number of lines of C++ code in MapReduce from 3,800 to 700 – a significant reduction, albeit in a relatively small system. This also results in other key benefits, including making it easier to update algorithms as there is a clean separation of concerns between what is effectively the application logic and the associated management of the distributed computation. In addition, improvements to the underlying MapReduce implementation immediately benefit all MapReduce applications. The downside is a more prescriptive framework, albeit one that can be customized by specifying the *map* and *reduce* and indeed other functions, as will become apparent below.

Function	Initial step	Map phase	Intermediate step	Reduce phase
Word count	Partition data into fixed-size chunks for processing	For each occurrence of word in data partition, emit $\langle \text{word}, 1 \rangle$	Merge/sort all key-value keys according to their intermediary key	For each word in the intermediary set, count the number of 1s
Grep		Output a line if it matches a given pattern		Null
Sort <i>N.B. This relies heavily on the intermediate step</i>		For each entry in the input data, output the key-value pairs to be sorted		Null
Inverted index		Parse the associated documents and output a $\langle \text{word}, \text{document ID} \rangle$ pair wherever that word exists		For each word, produce a list of (sorted) document IDs

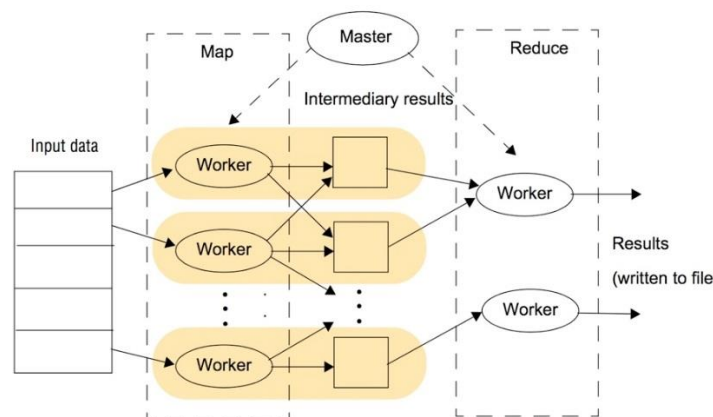
**Fig. 2.9** Examples of the use of MapReduce

To further illustrate the use of MapReduce, we provide in Fig. 2.9 a set of examples of common functions and how they would be implemented using *map* and *reduce* functions. The shared steps in the computation performed by the MapReduce framework are also shown for completeness. Further details of these examples can be found in Dean and Ghemawat [2004].

**MapReduce architecture:** MapReduce is implemented by a library that, as mentioned above, hides the details associated with parallelization and distribution and allows the programmer to focus on specifying the *map* and *reduce* functions. This library is built on top of other aspects of the Google infrastructure, in particular using RPC for communication and GFS for the storage of intermediary values. It is also common for MapReduce to take its input data from Bigtable and produce a table as a result, for example as with the Google Analytics example mentioned above.

The overall execution of a MapReduce program is captured in Fig. 2.10 which shows the key phases involved in execution:

- The first stage is to split the input file into  $M$  pieces, with each piece being typically 16–64 megabytes in size (therefore no bigger than a single chunk in GFS). The actual size is tunable by the programmer and therefore the programmer is able to optimize this for the particular parallel processing to follow. The key space associated with the intermediary results is also partitioned into  $R$  pieces using a (programmable) partition function. The overall computation therefore involves  $M$  *map* executions and  $R$  *reduce* executions.
- The MapReduce library then starts a set of worker machines (*workers*) from the pool available in the cluster, with one being designated as the *master* and others being used for executing *map* or *reduce* steps. The number of workers is normally much less than  $M+R$ . For example, Dean and Ghemawat [2008] report on typical figures of  $M=200,000$ ,  $R=5000$  with 2000 worker machines allocated to the task. The goal of the master is to monitor the state of workers and allocate idle workers to tasks, the execution of *map* or *reduce* functions. More precisely, the master keeps track of the status of *map* and *reduce* tasks in terms of being *idle*, *inprogress* or *completed* and also maintains information on the location of intermediary results for passing to workers allocated a *reduce* task.
- A worker that has been assigned a *map* task will first read the contents of the input file allocated to that *map* task, extract the key-value pairs and supply them as input to the *map* function. The output of the *map* function is a processed set of key/value pairs that are held in an intermediary buffer. As the input data is stored in GFS, the file will be replicated on (say) three machines. The master attempts to allocate a worker on one of these three machines to ensure locality and minimize the use of network bandwidth. If this is not possible, a machine near the data will be selected.



**Fig. 2.10** The overall execution of a MapReduce program

- The intermediary buffers are periodically written to a file local to the map computation. At this stage, the data are partitioned according to the partition function, resulting in  $R$  regions. This partition function, which is crucial to the operation of MapReduce, can be specified by the programmer, but the default is to perform a hash function on the key and then apply modulo  $R$  to the hashed value to produce  $R$  partitions, with the end result that intermediary results are grouped according to the hash value. Dean and Ghemawat [2004] provide the alternative example where keys are URLs and the programmer wants to group intermediary results by the associated host:  $hash(Hostname(key)) \bmod R$ . The master is notified when partitioning has completed and is then able to request the execution of associated *reduce* functions.

- When a worker is assigned to carry out a *reduce* function, it reads its corresponding partition from the local disk of the map workers using RPC. This data is sorted by the MapReduce library ready for processing by the *reduce* function. Once sorting is completed, the *reduce* worker steps through the key value pairs in the partition applying the *reduce* function to produce an accumulated result set, which is then written to an output file. This continues until all keys in the partition are processed.

**Achieving fault tolerance:** The MapReduce implementation provides a strong level of fault tolerance, in particular guaranteeing that if the *map* and *reduce* operations are *deterministic* with respect to their inputs (that is they always produce the same outputs for a given set of inputs), then the overall MapReduce task will produce the same output as a sequential execution of the program, even in the face of failures. To deal with failure, the master sends a *ping* message periodically to check that a worker is alive and carrying out its intended operation. If no response is received, it is assumed that the worker has failed and this is recorded by the master. The subsequent action then depends on whether the task executing was a *map* or a *reduce* task:

- If the worker was executing a *map* task, this task is marked as *idle*, implying it will then be rescheduled. This happens irrespective of whether the associated task is in progress or completed. Remember that results are stored on local disks, and hence if the machine has failed the results will be inaccessible.
- If the worker was executing a *reduce* task, this task is marked as *idle* only if it was still in progress; if it is completed, the results will be available as they are written to the global (and replicated) file system. Note that to achieve the desired semantics, it is important that the outputs from *map* and *reduce* tasks are written atomically, a property ensured by the MapReduce library in cooperation with the underlying file system. Details of how this is achieved can be found in Dean and Ghemawat [2008].

MapReduce also implements a strategy to deal with workers that may be taking a long time to complete (known as *stragglers*). Google has observed that it is relatively common for some workers to run slowly, for example because of a faulty disk that may perform badly due to a number of error-correction steps involved in data transfers. To deal with this, when a program execution is close to completion, the master routinely starts backup workers for all remaining *in-progress* tasks. The associated tasks are marked as *completed* when either the original or the new worker completes. This is reported as having a significant impact on completion times, again circumventing the problem of working with commodity machines that can and do fail. As mentioned above, MapReduce is designed to operate together with Bigtable in processing large volumes of structured data. Indeed, within Google it is common to find applications that use a mix of all the infrastructural elements. In the box on page 961, we describe the support provided to the Google Maps and Google Earth applications by MapReduce, Bigtable and GFS.

## 2.7 Summary

This case study examined the overall architecture of the system together with in-depth studies of the key underlying services – specifically, protocol buffers, the publish subscribe service, GFS, Chubby, Bigtable, and MapReduce - which all work together to support complex distributed applications and services including the core search engine. One key lesson to be taken from this case study is the importance of really understanding your application domain, deriving a core set of underlying design principles and applying them consistently. In the case of Google, this manifests itself in a strong advocacy of simplicity and low-overhead approaches coupled with an emphasis on testing, logging and tracing. The end result is an architecture that is highly scalable, reliable, high performance and open in terms of supporting new applications and services. The Google infrastructure is one of a number of middleware solutions for cloud computing that have emerged in recent years.

Other solutions include the Amazon Web Services (AWS) [aws.amazon.com], Microsoft's Azure [www.microsoft.com IV] and open source solutions including Hadoop (which includes an implementation of MapReduce) [hadoop.apache.org], Eucalyptus [open.eucalyptus.com], the Google App Engine (available externally and providing a window on some but not all of the functionality offered by the Google infrastructure) [code.google.com IV] and Sector/Sphere [sector.sourceforge.net]. OpenStreetMap [www.openstreetmap.org], an open alternative to Google Maps that operates in a similar manner using voluntarily developed software and non-commercial servers, has also been developed. Details of these implementations are generally available and the reader is encouraged to study a selection of these architectures, comparing the design choices with those presented in the above case study. Beyond that, there is a real paucity of published case studies related to distributed systems design, and this is a pity given the potential educational value of studying overall distributed systems architectures and their associated design principles. The main contribution of this chapter is therefore to provide a first in-depth case study illustrating the complexities of designing and implementing a complete distributed system solution.

## 2.8 References

1. Lanville, A.M. and Meyer, C.D. (2006). *Pagerank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ: Princeton University Press.
2. Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, Vol. 30, Issue 1-7, pp. 107-117.
3. Hennessy, J.L. and Patterson, D.A. (2006). *Computer Architecture: A Quantitative Approach*, 4th edn. San Francisco:CA: Morgan Kaufmann.
4. Ghemawat, S., Gobiuff, H. and Leung, S. (2003). The Google file system. *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, pp. 29-43.
5. Burrows, M. (2006). The Chubby lock service for loosely-coupled distributed systems. In *Proc. of the 7<sup>th</sup> Symposium on Operating Systems Design and Implementation*, Seattle, WA, pp. 335-350.
6. Pinheiro, E., Weber, W.D. and Barroso, L.A. (2007). Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, pp. 17-28.
7. Pike, R., Presotto, D., Thompson, K., Trickey, H. and Winterbottom, P. (1993). The use of name spaces in Plan 9. *Operating Systems Review*, Vol. 27, No. 2, pp. 72-76.
8. Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, Vol. 16, No. 2, pp. 133-69.
9. Lamport, L., Shostak, R. and Pease, M. (1982). Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, pp. 382-401.
10. Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Comms. ACM*, Vol. 51, No. 1, pp. 107-113.
11. Dean, J. and Ghemawat, S. (2004). MapReduce: simplified data processing on large clusters. In *Proc. of Operating System Design and Implementation, (OSDI'04)*, San Francisco, CA, pp. 137-150.

[This chapter is a shortened version of the chapter 21: Designing Distributed Systems: Google Case Study "DISTRIBUTED SYSTEMS Concepts and Design, Fifth Edition by George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, Addison Wesley".]

**CHAPTER 3**  
**TUTORIAL ON CLOUD SERVICES: VMWARE, EC2, APP ENGINE**

### 3. TUTORIAL ON CLOUD SERVICES: VMWARE, EC2, APP ENGINE

In this tutorial, we will show few demonstrations on different IaaS and PaaS services available. The hands on practice will clarify your confusions about cloud and you will learn about the usefulness of those cloud services.

#### 3.1 VMWare: a tool to create IaaS

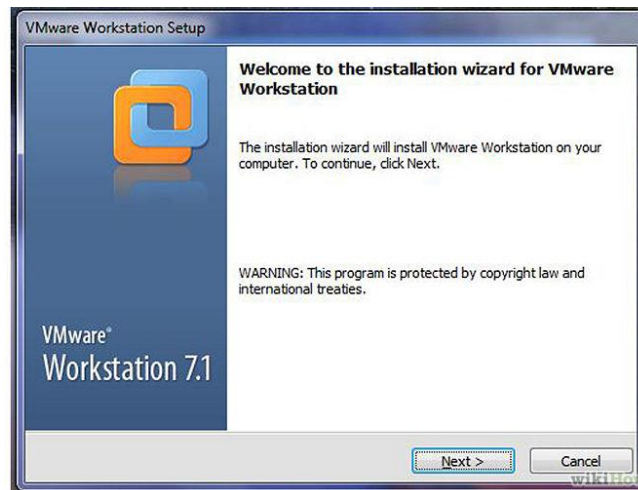
##### 3.1.1 How to Install VMware Workstation and Create a Virtual Machine on Your PC

This introduction is about how to install a virtual machine on your computer. A virtual machine (VM) is a software implementation of a machine (a computer) that executes programs like a physical machine. Most people have one computer. If you want to set up a local area network or a machine to be small-scale experiments, it is not enough. You also want to have both windows and Linux OS. Buy a computer is not worth it. Fortunately, Virtual machine software can virtual a lot of guests in a host computer.

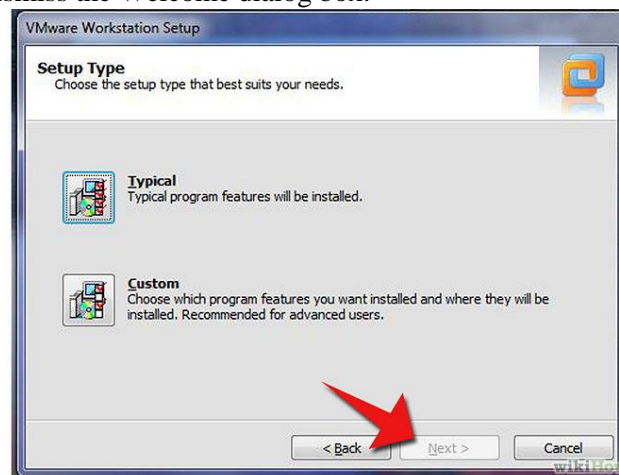
Steps

Step 1: Install the VMware Workstation a Windows host computers.

Note: To install Workstation on a Windows 7 host computer, you must log on as administrator. If you are installing from a CD, put your CD in your CD-ROM drive, it will begin automatically. If you are installing from a downloaded file, browse to the directory where you saved the downloaded installer file and run the installer. The file name is similar with this: VMware-workstation-full-7.1.3-324285.exe.



Step 2: Click Next to dismiss the Welcome dialog box.



Step 3: Choose the set up type you prefer. If you don't know it very well, choose typical. Then click next.



Step 4: Choose the directory in which to install VMware Workstation.

To install it in a directory other than the default, click Change and browse to your directory of choice. If the directory does not exist, the installer creates it for you. Click Next. Caution: Do not install VMware Workstation on a network drive.

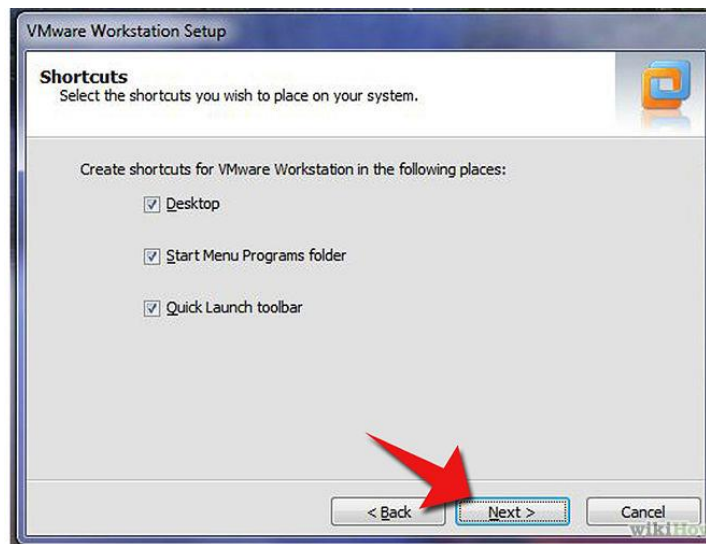


Step 5: Select if you want to check for product updates on startup. Deselect the check box if you do not want to check it.





Step 6: Select for if you like to feedback to VMware.  
Deselect the check box if you do not want to feedback. Click next.



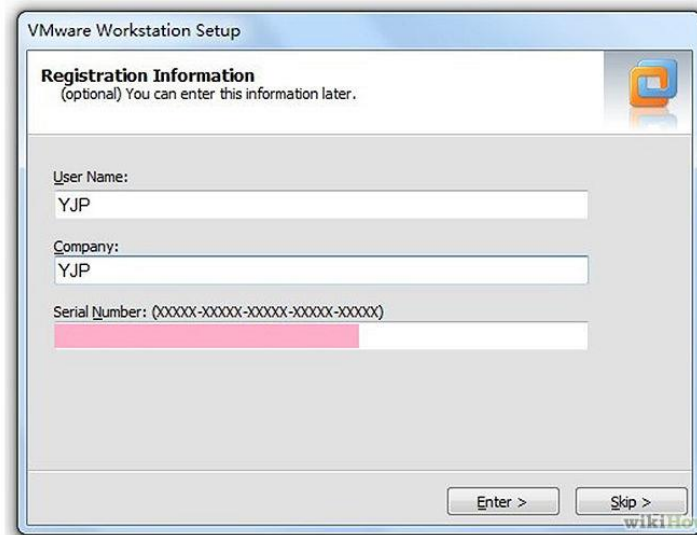
Step 7: Select the shortcuts that you want the installer to create.

Choices include Desktop, Start menu, and Quick Launch toolbar. Deselect any shortcuts you do not want the installer to create.



Step 8: The installer has gathered the necessary information and is ready to begin installing the software.

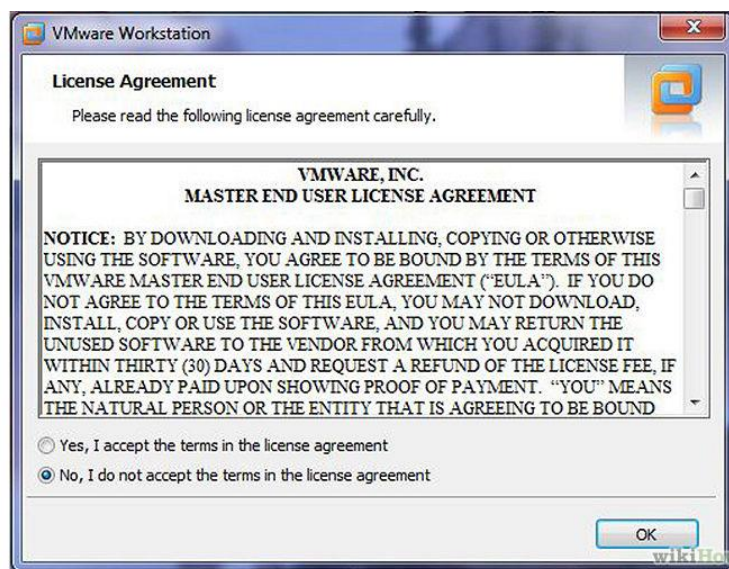
If you want to change any settings or information you provided, now is the time to make those changes. Click Back until you reach the dialog box containing the information you want to change. If you do not need to make any changes, click Continue. The installer begins copying files to your computer.



Step 9: Enter your serial number, your name(Optional), company name(Optional),then click Next.

Note: If you skip this step, you must enter your serial number later, before you can power on a virtual machine.

Step 10: Restart your computer, allow VMware Workstation to complete the installation, then double-click the VMware Workstation icon on your desktop.



Step 11: Select the Yes, I accept the terms in the license agreement option, then click Next.

Step 12: Start the New Virtual Machine Wizard.

Choose File > New > Virtual Machine to begin creating your virtual machine.

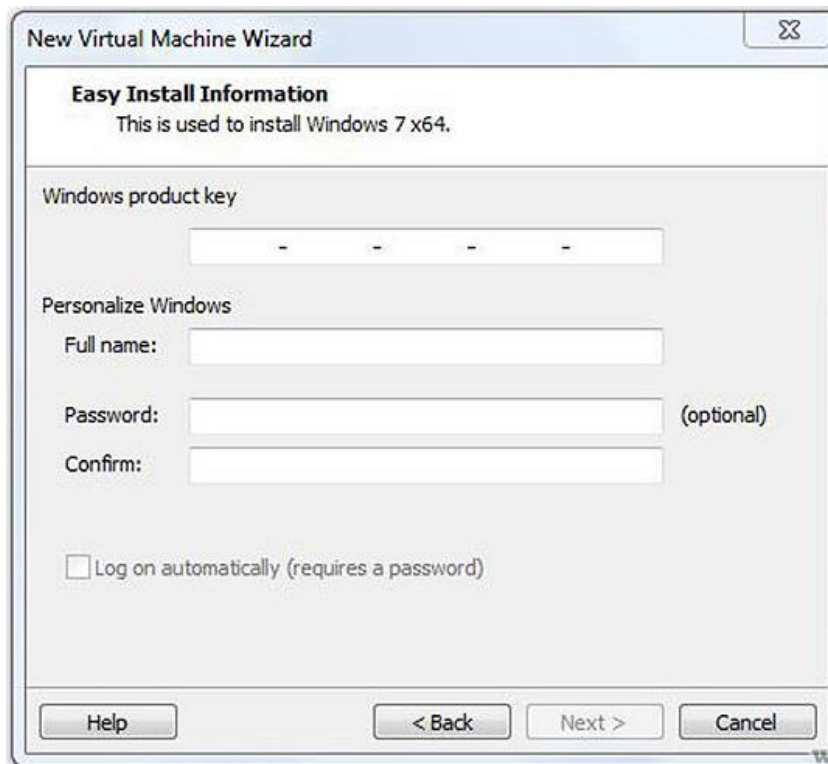


Step 13: Recommend you choose typical, then click next.



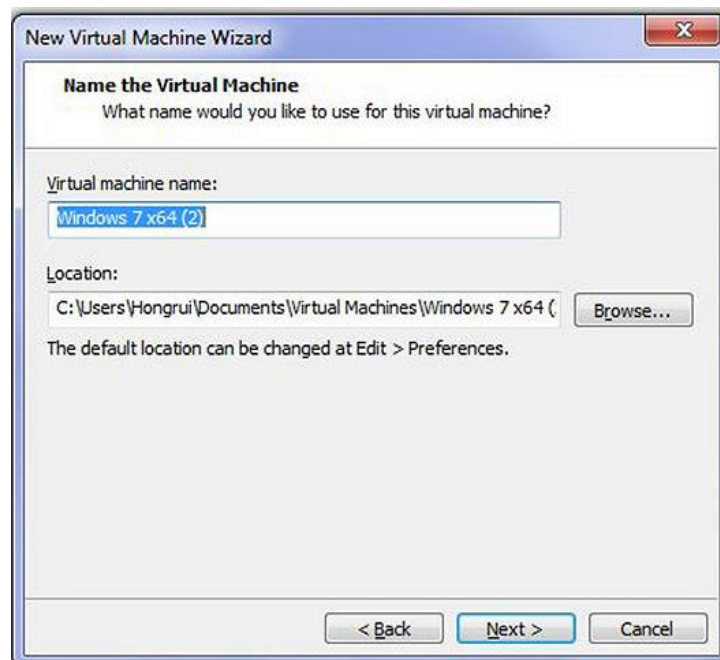
Step 14: Begin to install a guest operating system.

Choose how you will install the guest operating system, then click next.



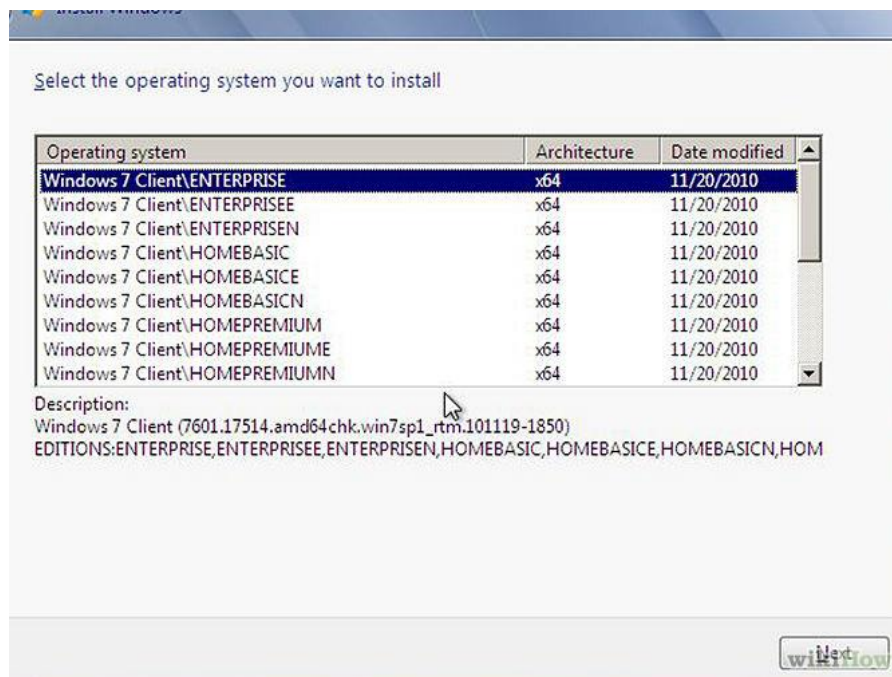
Step 15: Here is how to install a windows 7 as a guest operating system.

Enter the windows product key, full name, password(optional) and click next



Step 16: Name the virtual machine and choose the location for it.

Click Browse if you want to change the default location. Click next.



Step 17: Installing a guest operating system inside your VMware Workstation virtual machine is the same as installing it on a physical computer.

Step 18: Power on your virtual machine by clicking the Power On button. Follow the instructions provided by the operating system vendor.

Discussion. Installing any other operating system such as RedHat Linux or Debian on VMWare is similar to installing Windows 7 as a guest operating system. You will just have to follow the wizard to install the operating system (that is provided to install it on a physical machine). VMWare will many guest operating systems installed on it, provides a good demonstration of Infrastructure as a Service (IaaS) that happens in Amazon EC2.

### 3.1.2 Sources:

<http://www.wikihow.com/Install-VMware-Workstation-and-Create-a-Virtual-Machine-on-Your-PC>  
<http://blog.pluralsight.com/install-windows-7-in-vmware-workstation>

## 3.2 Amazon EC2: Infrastructure as a Service

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

### 3.2.1 Features of Amazon EC2

Amazon EC2 provides the following features:

- Virtual computing environments, known as *instances*
- Preconfigured templates for your instances, known as *Amazon Machine Images (AMIs)*, that package the bits you need for your server (including the operating system and additional software)
- Various configurations of CPU, memory, storage, and networking capacity for your instances, known as *instance types*
- Secure login information for your instances using *key pairs* (AWS stores the public key, and you store the private key in a secure place)
- Storage volumes for temporary data that's deleted when you stop or terminate your instance, known as *instance store volumes*
- Persistent storage volumes for your data using Amazon Elastic Block Store (Amazon EBS), known as *Amazon EBS volumes*
- Multiple physical locations for your resources, such as instances and Amazon EBS volumes, known as *regions* and *Availability Zones*
- A firewall that enables you to specify the protocols, ports, and source IP ranges that can reach your instances using *security groups*
- Static IP addresses for dynamic cloud computing, known as *Elastic IP addresses*
- Metadata, known as *tags*, that you can create and assign to your Amazon EC2 resources
- Virtual networks you can create that are logically isolated from the rest of the AWS cloud, and that you can optionally connect to your own network, known as *virtual private clouds (VPCs)*

### 3.2.2 Setting Up with Amazon EC2

If you've already signed up for Amazon Web Services (AWS), you can start using Amazon EC2 immediately. You can open the Amazon EC2 console, click **Launch Instance**, and follow the steps in the launch wizard to launch your first instance.

If you haven't signed up for AWS yet, or if you need assistance launching your first instance, complete the following tasks to get set up to use Amazon EC2:

1. Sign Up for AWS
2. Create an IAM User
3. Create a Key Pair
4. Create a Virtual Private Cloud (VPC)
5. Create a Security Group

### 3.2.3 Sign Up for AWS

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all services in AWS, including Amazon EC2. You are charged only for the services that you use.

With Amazon EC2, you pay only for what you use. If you are a new AWS customer, you can get started with Amazon EC2 for free.

If you have an AWS account already, skip to the next task. If you don't have an AWS account, use the following procedure to create one.

### 3.3 To create an AWS account

1. Open <http://aws.amazon.com>, and then click **Sign Up**.



2. Put your email address and check "I am a new user" radio button.



3. Put your name and password, press create account button.

#### Login Credentials

Use the form below to create login credentials that can be used for AWS as well as Amazon.com.

My name is:

My e-mail address is:

Type it again:

note: this is the e-mail address that we will use to contact you about your account

Enter a new password:

Type it again:

4. Provide address details, check the agreement and press the continue button.

**Contact Information**

\* Required Fields

Full Name\* Kamrul Hasan

Company Name IUT

Country\* Bangladesh


Address\* Asst Prof, CSE, IUT  
Apartment, suite, unit, building, floor, etc.

City\* Gazipur

State / Province or Region\* Dhaka

Postal Code\* 1704

Phone Number\* 8801195239734

Security Check 

Refresh Image

Please type the characters as shown above


CXWNEB

**AWS Customer Agreement**

Check here to indicate that you have read and agree to the terms of the [AWS Customer Agreement](#)

Create Account and Continue

5. This will take you to the payment information page. Put your credit card number and continue.

 Amazon Web Services Sign Up

Contact Information   Payment Information   Identity Verification   Support Plan   Confirmation

**Payment Information**

Please enter your payment information below. You will be able to try a broad set of AWS products for free via the Free Usage Tier. We will only bill your credit card for usage that is not covered by our Free Usage Tier.

AWS Free Usage Tier	Compute Amazon EC2	Storage Amazon S3	Database Amazon RDS
free for 1 year	750hrs/month*	5GB	750hrs/month*

[View all offers & details >](#)

Credit Card Number   Expiration Date

Cardholder's Name

Choose Your Billing Address  
Select the address associated with your credit card.

Use my contact address  
(Asst Prof, CSE, IUT Gazipur Dhaka 1704 BD)

Use a new address

Continue

6. The next step takes you to phone verification step. Amazon will place an automated call to verify you by asking a PIN number (displayed on the screen) to be typed on the phone keypad. Press call me now to initiate the call.



Identity Verification

You will be called immediately by an automated system and prompted to enter the PIN number provided.

**1. Provide a telephone number**

Please enter your information below and click the "Call Me Now" button.

Country Code	Phone Number	Ext
<input type="text" value="Bangladesh (+880)"/>	<input type="text" value="8801195239734"/>	<input type="text"/>

**2. Call in progress**

Identity Verification

You will be called immediately by an automated system and prompted to enter the PIN number provided.

**1. Provide a telephone number ✓**

**2. Call in progress**

Please follow the instructions on the telephone and key in the following Personal Identification Number (PIN) on your telephone when prompted.

If you have not yet received a call at the number indicated above please wait. This page will automatically update with what you need to do next.

**3. Identity verification complete**

7. Once you provide the PIN number your account will be created.

### 3.4 Create an IAM User

Services in AWS, such as Amazon EC2, require that you provide credentials when you access them, so that the service can determine whether you have permission to access its resources. The console requires your password. You can create access keys for your AWS account to access the command line interface or API. However, we don't recommend that you access AWS using the credentials for your AWS account; we recommend that you use AWS Identity and Access Management (IAM) instead. Create an IAM user, and then add the user to an IAM group with administrative permissions or and grant this user administrative permissions. You can then access AWS using a special URL and the credentials for the IAM user.

If you signed up for AWS but have not created an IAM user for yourself, you can create one using the IAM console.

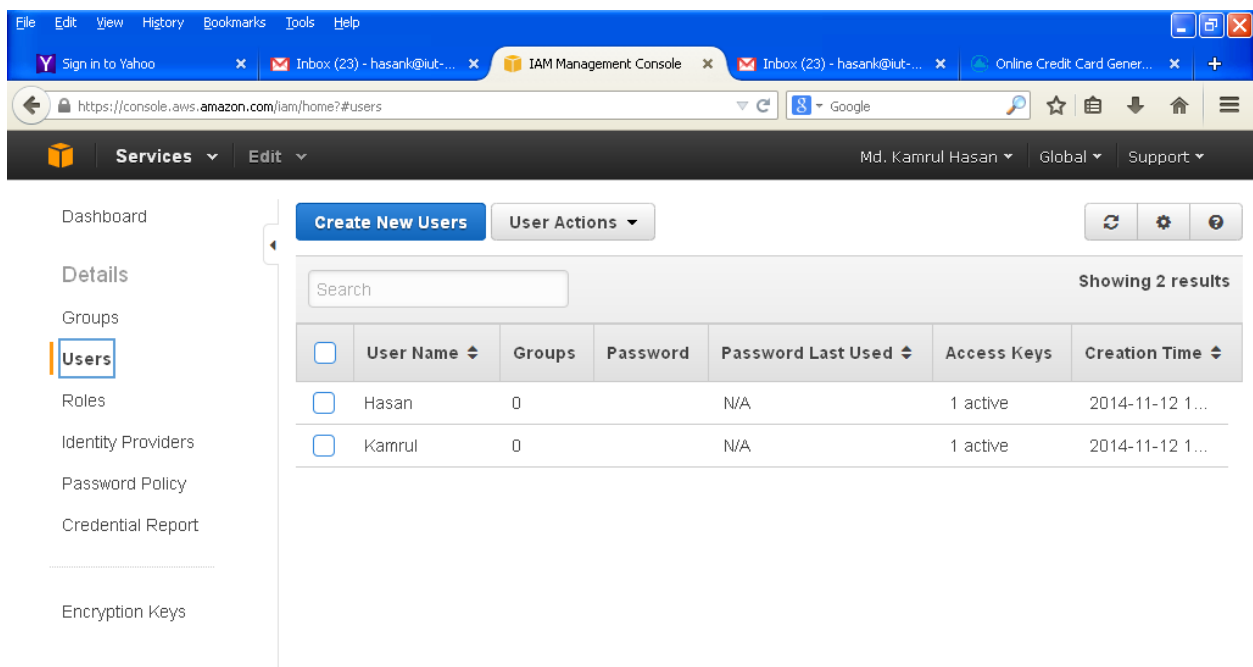
### 3.4.1 To create the Administrators group

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane, click **Groups** and then click **Create New Group**.
3. In the **Group Name** box, type **Administrators** and then click **Next Step**.
4. In the **Select Policy Template** section, click **Select** next to the **Administrator Access** policy template.
5. Click **Next Step** and then click **Create Group**.

Your new group is listed under **Group Name**.

### 3.4.2 To create the IAM user, add the user to the Administrators group, and create a password for the user

1. In the navigation pane, click **Users** and then click **Create New Users**.
2. In box **1**, type a user name and then click **Create**.
3. Click **Download Credentials** and save your access key in a secure place. You will need your access key for programmatic access to AWS using the AWS CLI, the AWS SDKs, or the HTTP APIs.



#### Note

You cannot retrieve the secret access key after you complete this step; if you misplace it you must create a new one.

After you have downloaded your access key, click Close.

4. In the content pane, under **User Name**, click the name of the user you just created. (You might need to scroll down to find the user in the list.)
5. In the content pane, in the **Groups** section, click **Add User to Groups**.
6. Select the **Administrators** group and then click **Add to Groups**.

7. In the content pane, in the **Security Credentials** section (you might need to scroll down to find this section), under **Sign-In Credentials**, click **Manage Password**.
8. Select **Assign a custom password** and then type and confirm a password. When you are finished, click **Apply**.

To sign in as this new IAM user, sign out of the AWS console, then use the following URL, where *your\_aws\_account\_id* is your AWS account number without the hyphens (for example, if your AWS account number is 1234-5678-9012, your AWS account ID is 123456789012):

[https://your\\_aws\\_account\\_id.signin.aws.amazon.com/console/](https://your_aws_account_id.signin.aws.amazon.com/console/)

Enter the IAM user name and password that you just created. When you're signed in, the navigation bar displays "*your\_user\_name @ your\_aws\_account\_id*".

If you don't want the URL for your sign-in page to contain your AWS account ID, you can create an account alias. From the IAM dashboard, click **Create Account Alias** and enter an alias, such as your company name. To sign in after you create an account alias, use the following URL:

[https://your\\_account\\_alias.signin.aws.amazon.com/console/](https://your_account_alias.signin.aws.amazon.com/console/)

To verify the sign-in link for IAM users for your account, open the IAM console and check under **IAM users sign-in link** on the dashboard.

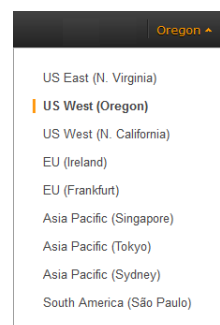
### 3.4.3 Create a Key Pair

AWS uses public-key cryptography to secure the login information for your instance. A Linux instance has no password; you use a key pair to log in to your instance securely. You specify the name of the key pair when you launch your instance, then provide the private key when you log in using SSH.

If you haven't created a key pair already, you can create one using the Amazon EC2 console. Note that if you plan to launch instances in multiple regions, you'll need to create a key pair in each region.

#### 3.4.3.1 To create a key pair

1. Open the Amazon EC2 console.
2. From the navigation bar, select a region for the key pair. You can select any region that's available to you, regardless of your location. However, key pairs are specific to a region; for example, if you plan to launch an instance in the US West (Oregon) region, you must create a key pair for the instance in the US West (Oregon) region.
3. Click **Key Pairs** in the navigation pane.
4. Click **Create Key Pair**.
5. Enter a name for the new key pair in the **Key pair name** field of the **Create Key Pair** dialog box, and then click **Create**. Choose a name that is easy for you to remember, such as your IAM user name, followed by -key-pair, plus the region name. For example, *me-key-pair-uswest2*.
6. The private key file is automatically downloaded by your browser. The base file name is the name you specified as the name of your key pair, and the file name extension is .pem. Save the private key file in a safe place.



### Important

This is the only chance for you to save the private key file. You'll need to provide the name of your key pair when you launch an instance and the corresponding private key each time you connect to the instance.

If you will use an SSH client on a Mac or Linux computer to connect to your Linux instance, use the following command to set the permissions of your private key file so that only you can read it.

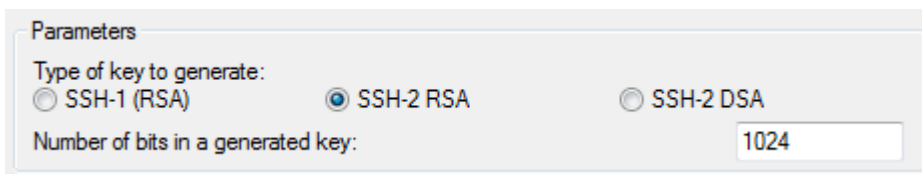
```
$ chmod 400 your_user_name-key-pair-region_name.pem
```

#### 3.4.3.2 To connect to your instance using your key pair

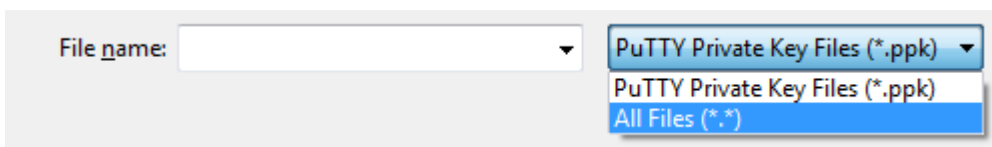
To your Linux instance from a computer running Mac or Linux, you'll specify the .pem file to your SSH client with the -i option and the path to your private key. To connect to your Linux instance from a computer running Windows, you can use either MindTerm or PuTTY. If you plan to use PuTTY, you'll need to install it and use the following procedure to convert the .pemfile to a .ppk file.

#### 3.4.3.3 (Optional) To prepare to connect to a Linux instance from Windows using PuTTY

1. Download and install PuTTY from <http://www.chiark.greenend.org.uk/~sgtatham/putty/>. Be sure to install the entire suite.
2. Start PuTTYgen (for example, from the **Start** menu, click **All Programs >PuTTY>PuTTYgen**).
3. Under **Type of key to generate**, select **SSH-2 RSA**.



4. Click **Load**. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, select the option to display files of all types.



5. Select the private key file that you created in the previous procedure and click **Open**. Click **OK** to dismiss the confirmation dialog box.
6. Click **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Click **Yes**.
7. Specify the same name for the key that you used for the key pair. PuTTY automatically adds the .ppk file extension.

#### 3.4.4 Create a Virtual Private Cloud (VPC)

Amazon VPC enables you to launch AWS resources into a virtual network that you've defined. If you have a default VPC, you can skip this section and move to the next task, Create a Security Group. To determine whether you have a default VPC, see Supported Platforms in the Amazon EC2 Console. Otherwise, you can create a nondefault VPC in your account using the steps below.

### Important

If your account supports EC2-Classic in a region, then you do not have a default VPC in that region. T2 instances must be launched into a VPC.

#### 3.4.4.1 To create a nondefault VPC

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. From the navigation bar, select a region for the VPC. VPCs are specific to a region, so you should select the same region in which you created your key pair.
3. On the VPC dashboard, click **Start VPC Wizard**.
4. On the **Step 1: Select a VPC Configuration** page, ensure that **VPC with a Single Public Subnet** is selected, and click **Select**.
5. On the **Step 2: VPC with a Single Public Subnet** page, enter a friendly name for your VPC in the **VPC name** field. Leave the other default configuration settings, and click **Create VPC**. On the confirmation page, click **OK**.

#### 3.4.5 Create a Security Group

Security groups act as a firewall for associated instances, controlling both inbound and outbound traffic at the instance level. You must add rules to a security group that enable you to connect to your instance from your IP address using SSH. You can also add rules that allow inbound and outbound HTTP and HTTPS access from anywhere.

Note that if you plan to launch instances in multiple regions, you'll need to create a security group in each region

##### 3.4.5.1 To create a security group with least privilege

1. Open the Amazon EC2 console.
2. From the navigation bar, select a region for the security group. Security groups are specific to a region, so you should select the same region in which you created your key pair.
3. Click **Security Groups** in the navigation pane.
4. Click **Create Security Group**.
5. Enter a name for the new security group and a description. Choose a name that is easy for you to remember, such as your IAM user name, followed by `_SG_`, plus the region name. For example, `me_SG_uswest2`.
6. In the **VPC** list, ensure that your default VPC is selected; it's marked with an asterisk (\*).



#### Note

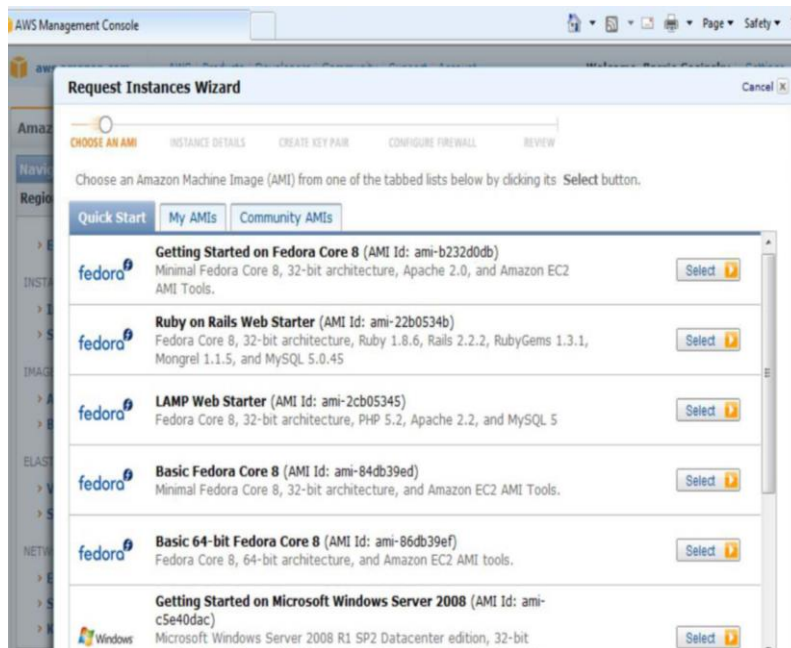
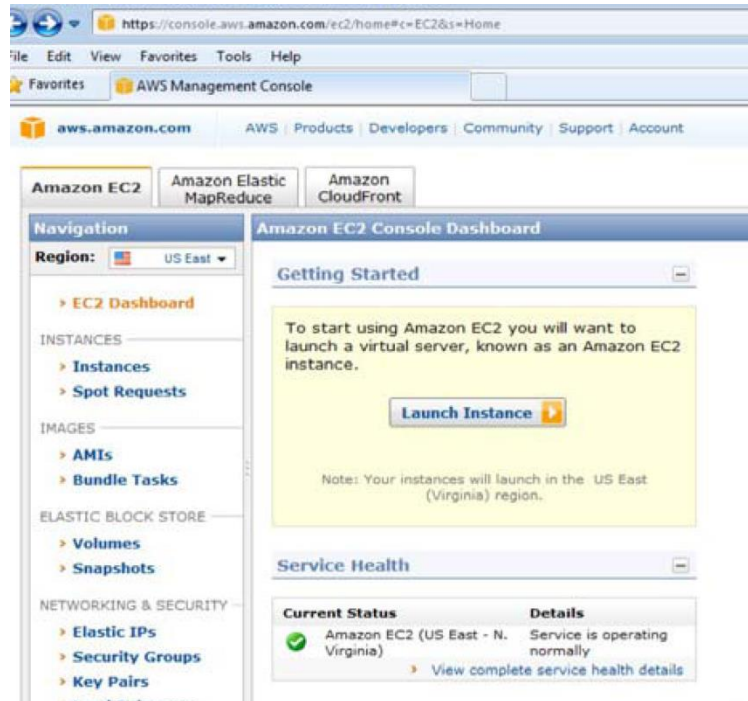
If your account supports EC2-Classic, select the VPC that you created in the previous task.

7. On the **Inbound** tab, create the following rules (click **Add Rule** for each new rule), and then click **Create**:
  - Select **HTTP** from the **Type** list, and make sure that **Source** is set to **Anywhere** (0.0.0.0/0).
  - Select **HTTPS** from the **Type** list, and make sure that **Source** is set to **Anywhere** (0.0.0.0/0).

- Select **SSH** from the **Type** list. In the **Source** box, ensure **Custom IP** is selected, and specify the public IP address of your computer or network in CIDR notation. To specify an individual IP address in CIDR notation, add the routing prefix /32. For example, if your IP address is 203.0.113.25, specify 203.0.113.25/32. If your company allocates addresses from a range, specify the entire range, such as 203.0.113.0/24.

### 3.5 Launching an Amazon Machine Instance (AMI)

1. Click the Launch Instance button in the Getting Started Section to launch the Request Instances in the AWS Management Console , <https://console.aws.amazon.com>



2. Scroll the list to find the type of system image you want, and click Select.

3. Specify the Number of Instance(s) desired, the Availability Zone where the instance(s) should be

The screenshot shows the 'Request Instances Wizard' in the 'INSTANCE DETAILS' step. The progress bar at the top indicates the current step. Below the progress bar, there is a text prompt: 'Provide the details for your instance(s). You may also decide whether you want to launch your instances as "on-demand" or "spot" instances.' The form includes the following fields: 'Number of Instances' with a text input containing '1', 'Availability Zone' with a dropdown menu set to 'No Preference', and 'Instance Type' with a dropdown menu set to 'Small (m1.small, 1.7 GB)'. Below these fields, there are two radio button options: 'Launch Instances' (which is selected) and 'Request Spot Instances'. A description for 'Launch Instances' states: 'EC2 Instances let you pay for compute capacity by the hour with no long term commitments. This transforms what are commonly large fixed costs into much smaller variable costs.' At the bottom of the wizard, there are 'Back' and 'Continue' buttons.

4. Put advanced instance options

The screenshot shows the 'Request Instances Wizard' in the 'Advanced Instance Options' step. The progress bar at the top indicates the current step. Below the progress bar, there is a text prompt: 'Here you can choose a specific kernel or RAM disk to use with your instances. You can also choose to enable CloudWatch Monitoring or enter data that will be available from your instances once they launch.' The form includes the following fields: 'Kernel ID' with a dropdown menu set to 'Use Default', 'RAM Disk ID' with a dropdown menu set to 'Use Default', 'Monitoring' with a checkbox labeled 'Enable CloudWatch Monitoring for this instance' (which is checked) and a note '(additional charges will apply)', and 'User Data' with a text area and a checkbox labeled 'base64 encoded' (which is checked). At the bottom of the wizard, there are 'Back' and 'Continue' buttons.

5. Choose or create the key pair.

**Request Instances Wizard** Cancel X

CHOOSE AN AMI   
  INSTANCE DETAILS   
  **CREATE KEY PAIR**   
 CONFIGURE FIREWALL   
 REVIEW

Public/private key pairs allow you to securely connect to your instance after it launches. To create a key pair, enter a name and click **Create & Download your Key Pair**. You will then be prompted to save the private key to your computer. Note, you only need to generate a key pair once - not each time you want to deploy an Amazon EC2 instance.

**Choose from your existing Key Pairs**

Your existing Key Pairs\*:

Create a new Key Pair

Proceed without a Key Pair

6. Configure the firewall

**Request Instances Wizard** Cancel X

CHOOSE AN AMI   
  INSTANCE DETAILS   
  CREATE KEY PAIR   
  **CONFIGURE FIREWALL**   
 REVIEW

Security groups determine whether a network port is open or blocked on your instances. You may use an existing security group, or we can help you create a new security group to allow access to your instances using the suggested ports below. Add additional ports now or update your security group anytime using the Security Groups page. All changes take effect immediately.

Choose one or more of your existing Security Groups

**Create a new Security Group**

1. Name your Security Group

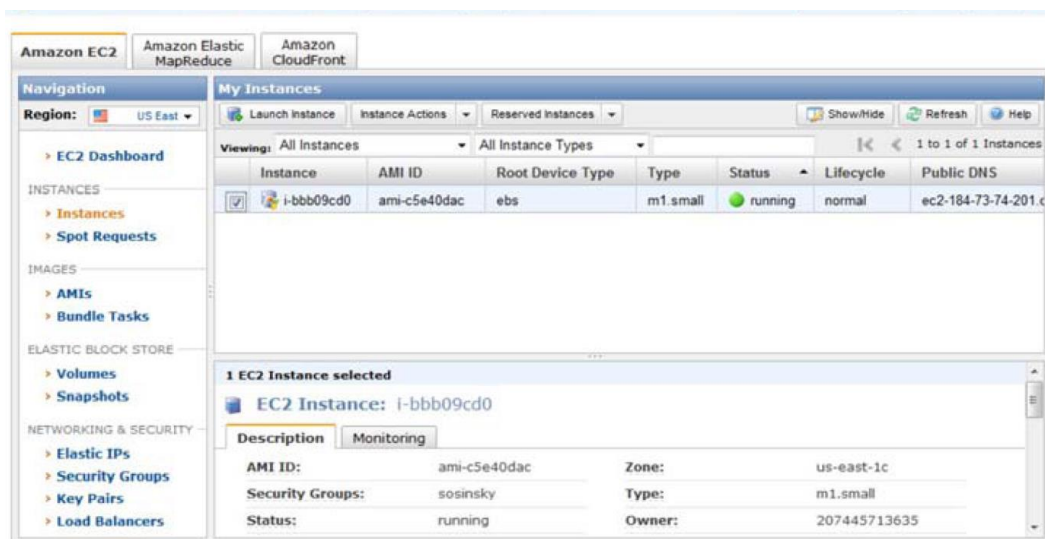
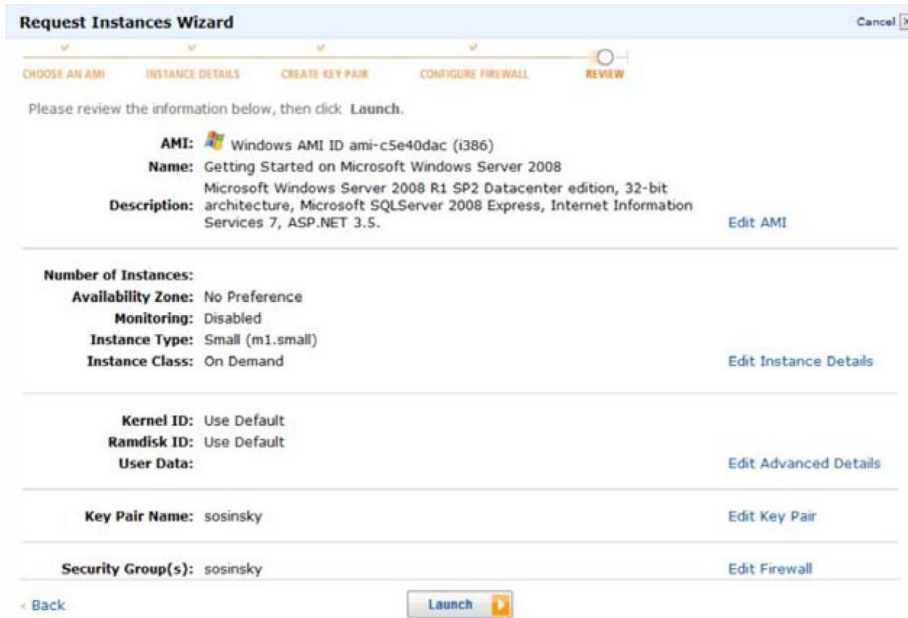
2. Describe your Security Group

3. Define allowed Connections

Application	Transport	Port	Source Network (IPv4 CIDR)	Actions
RDP	tcp	3389	All Internet	<input type="button" value="Remove"/>
MS SQL Server	tcp	1433	All Internet	<input type="button" value="Remove"/>
HTTP	tcp	80	All Internet	<input type="button" value="Remove"/>
Select...	-	-	All Internet Change	<input type="button" value=""/>

7. You can now start your instance.

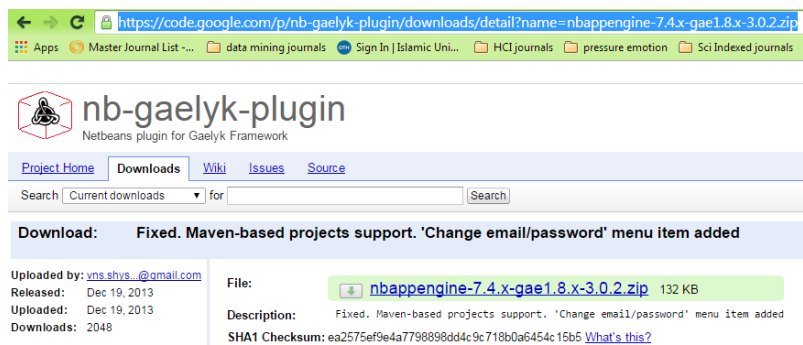




Google App Engine: Platform as a Service  
 Google App Engine on Netbeans 8

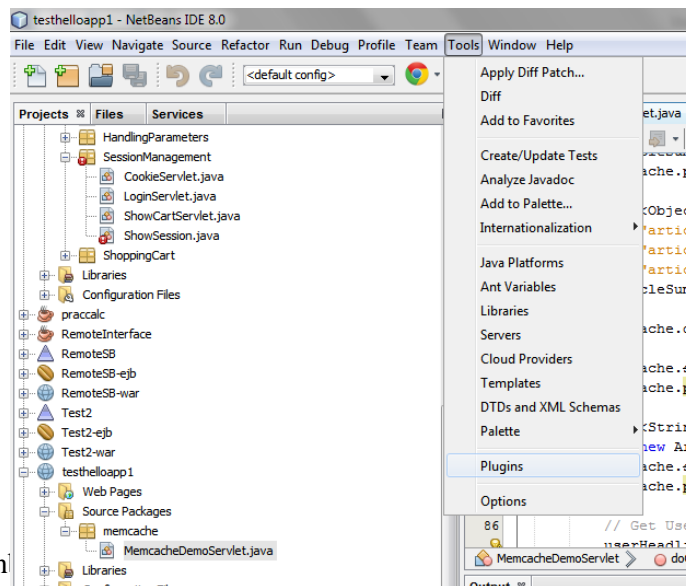
### 3.5.1 Adding plugins to Netbeans 8.0:

1. Download the zipped file for Netbeans 7.4 from the link, <https://code.google.com/p/nb-gaelyk-plugin/downloads/detail?name=nbappengine-7.4.x-gae1.8.x-3.0.2.zip>



2. Extract the zip file into some directory.

3. In Netbeans 8.0, follow to menu “Tools”, “Plugins”. And within the dialog, to “Downloaded” tab. Click “Add Plugin” and browse to the folder.



4. Select all .n
5. Watch out, don't close the dialog! Instead, click on the small “Install” button at the left bottom corner (marked in red circle).
6. Go through the Wizard, confirm license, confirm “plugins not signed” warning and go on.
7. Now you may close the dialog.
8. Restart Netbeans.

### 3.5.2 Configuring GoogleAppEngine SDK:

1. Download the Google AppEngine SDK from <https://cloud.google.com/appengine/downloads>
2. Unzip the downloaded zip file into any directory.
3. Open the “Services” window in NetBeans and right click on the Servers option. Add Server option will pop-up.
4. Choose the Google App Engine from the Server list. Click Next.
5. Choose the server location where you have unzipped the sdk.
6. Click on Next, confirm ports and DataNucleus. (take whatever given, no need to modify)

Sources:

1. <http://techtavern.wordpress.com/2014/05/13/google-app-engine-on-netbeans-8/>

### 3.6 Developing web applications using GoogleAppEngine:

1. To create a web application using GoogleAppEngine you need to have any google account which is free. For example, [someone@gmail.com](mailto:someone@gmail.com).
2. Login to your google account in [www.appengine.google.com](http://www.appengine.google.com) with email id and password. It will take you to the login page.
3. After login you will be redirected to Google App Engine home page to create a web application. Click on Create Application.
4. Give an application identifier and check for the availability. Give an Application title and check in “I accept these terms”, click on “Create Application”.

5. Then you will see a message like below:
6. Now open the netbeans and click on File → New Project. Then select web application project under java web categories and click next.
7. Give project name as the same name you have created for your google app engine web application which is “helloiutworkshop”. Select the project location and folder then click next.
8. In the server and settings option select Google App Engine as a server, select JavaEE 5 as Java EE version and keep the context path as it is. Click next.
9. No need to select any frameworks. Click on finish.
10. A project named “helloiutworkshop” will appear in the project pane. Right click the project and click on run.
11. You will see an output “Hello World!” in the default web browser. This project was run locally in your machine for testing.
12. To deploy the project in your google app engine account, right click the project and click on “Deploy to Google App Engine”. You may be asked for your app engine account information (email and password).
13. After deploying you can check the output in the link, <http://1.helloiutworkshop.appspot.com>

**CHAPTER 4**  
**GREEN CLOUD COMPUTING**

## 4. GREEN CLOUD COMPUTING

### 4.1 Introduction

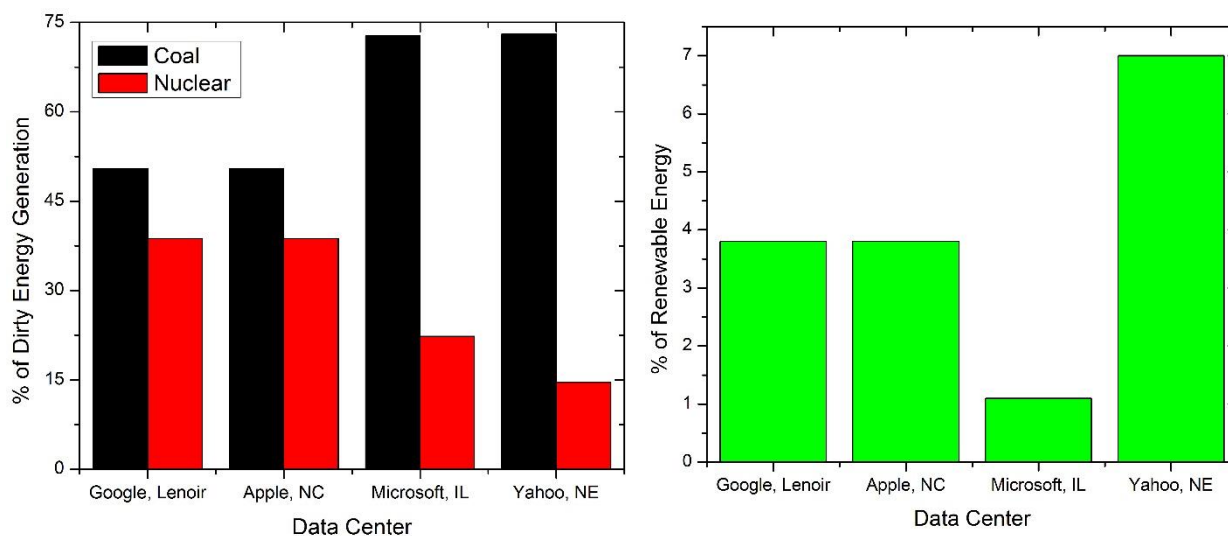
Cloud computing is an Internet-based computing system, whereby shared resources, software and information are provided to computers and other client devices on-demand, like a public utility. Strongly promoted by the leading industrial companies (e.g., Microsoft, Google, IBM, Amazon, etc.), cloud computing is quickly becoming popular in recent years. The characteristics of Clouds include on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. Traditionally, business organizations used to invest huge amount of capital and time in acquisition and maintenance of computational resources. The emergence of Cloud computing is rapidly changing this ownership-based approach to subscription-oriented approach by providing access to scalable infrastructure and services on-demand, as depicted in Fig. 4.1.

According to IDC (International Data Corporation) report [1], the global IT Cloud services spending has been increased from \$16 billion in 2008 to \$42 billion in 2012, representing a compound annual growth rate (CAGR) of 27%. Attracted by this growth prospects, Web-based companies (Amazon, eBay, Salesforce.com), hardware vendors (HP, IBM, Cisco), telecom providers (AT&T, Verizon), software firms (EMC/VMware, Oracle/Sun, Microsoft) and others are all investing huge amount of capital in establishing Cloud datacenters. According to Google's earnings reports, the company has spent \$US1.9 billion on datacenters in 2006, and \$US2.4 billion in 2007 [2].



Fig. 4.1: Cloud Computing Architecture

Some studies [26] observe that the Cloud phenomenon may aggravate the problem of carbon emissions and global warming. The reason given is that the collective demand for computing resources is expected to further increase dramatically in the next few years. Even the most efficiently built datacenter with the highest utilization rates will only mitigate, rather than eliminate, harmful CO<sub>2</sub> emissions. The reason given is that Cloud providers are more interested in electricity cost reduction rather than carbon emission. Comparison of significant cloud datacenters at different locations and their energy usage can be compared using the energy usage graphs, as shown in Fig. 4.2.

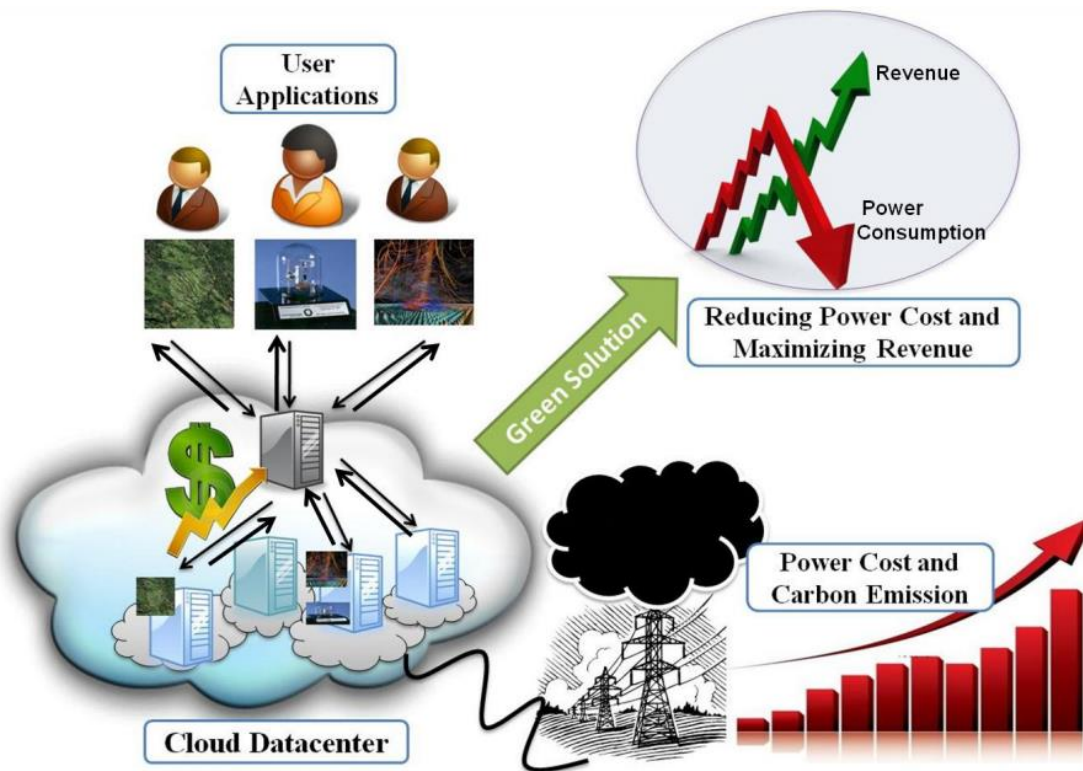
**Fig. 4.2: Comparison of Significant Cloud Datacenters**

Due to large number of equipment, datacenters can consume massive energy consumption and emit large amount of carbon. Servers and storage systems are not the only infrastructure that consumes energy in the datacenter. In reality, the cooling equipment consume equivalent amount of energy as the IT systems themselves. Ranganathan [3] suggests that for every dollar spent on electricity costs in large-scale datacenters another dollar is spent on cooling. Table 4.1 summarizes the percentage of energy consumed by each datacenter devices.

<b>Datacenter Component</b>	<b>Power Consumption (in percentage)</b>
Cooling device (Chiller, Computer Room Air Conditioning (CRAC))	33%+9%
IT Equipment	30%
Electrical Equipment (UPS, Power Distribution Units (PDUs), lighting)	28%

Table 4.1: Energy usage profile of data center components

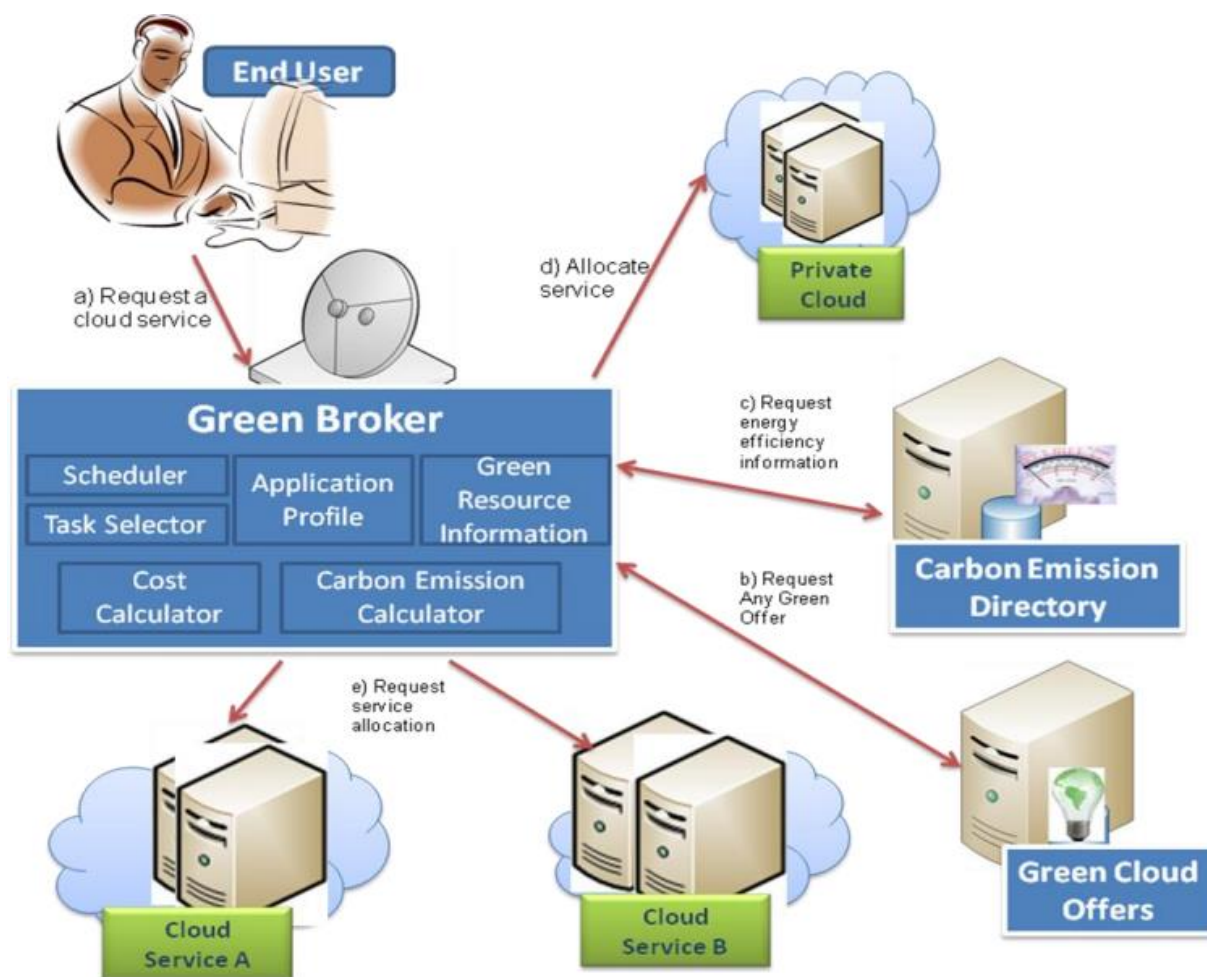
Clouds are essentially virtualized datacenters and applications offered as services on a subscription basis. They require high energy usage for its operation [4]. Today, a typical datacenter with 1000 racks need 10 Megawatt of power to operate [5], which results in higher operational cost. Thus, for a datacenter, the energy cost is significant component of its operating and up-front costs. In addition, in April 2007, Gartner found that the Information and Communication Technologies (ICT) industry generates about 2% of the total global CO<sub>2</sub> emissions, which is equal to the aviation industry [5]. According to a report published by the European Union, a decrease in emission volume of 15% - 30% is required before year 2020 to keep the global temperature increase below 2°C. Thus, energy consumption and carbon emission by Cloud infrastructures has become a key environmental concern. Green Cloud Computing (GCC) is the emerging cloud computing framework for reducing its carbon footprint in wholesome manner without sacrificing the quality of service (performance, responsiveness and availability) offered by the multiple Cloud providers.



**Fig. 4.3: Cloud and Environmental Sustainability**

#### 4.2 Green Cloud Architecture

In the Green Cloud architecture [25], users submit their Cloud service requests through a new middleware Green Broker that manages the selection of the greenest Cloud provider to serve the user's request. The Cloud providers can register their services in the form of "green offers" to a public directory which is accessed by Green Broker. The green offers consist of green services, pricing and time when it should be accessed for least carbon emission. Green Broker gets the current status of energy parameters for using various Cloud services from Carbon Emission Directory. The Carbon Emission Directory maintains all the data related to energy efficiency of Cloud service. This data may include Power Usage Effectiveness (PUE) and cooling efficiency of Cloud datacenter which is providing the service, the network cost and carbon emission rate of electricity. Green Broker calculates the carbon emission of all the Cloud providers who are offering the requested Cloud service. Then, it selects the set of services that will result in least carbon emission and buy these services on behalf users.



**Fig. 4.4: Green Cloud Architecture**

The Green Cloud framework is designed such that it keeps track of overall energy usage of serving a user request. It relies on two main components, Carbon Emission Directory and Green Cloud offers, which keep track of energy efficiency of each Cloud provider and also give incentive to Cloud providers to make their service “Green”. From user side, the Green Broker plays a crucial role in monitoring and selecting the Cloud services based on the user QoS requirements, and ensuring minimum carbon emission for serving a user. In general, a user can use Cloud to access any of these three types of services (SaaS, PaaS, and IaaS), and therefore process of serving them should also be energy efficient.

### 4.3 Green Cloud Technologies

To meet the growing demand for their services and ensure minimal costs, Cloud providers must implement power-efficient management of physical resources. Furthermore, as many applications require deadline constraints, power consumption in data centers must be minimized without violating the SLAs. According to Accenture Report [6], there are following four key factors that have enabled the Cloud computing to lower energy usage and carbon emissions from ICT. Due to these Cloud features, organizations can reduce carbon emissions by at least 30% per user by moving their applications to the Cloud.

- i. **Dynamic Provisioning:** In traditional setting, datacenters and private infrastructure used to be maintained to fulfill worst case demand. Thus, IT companies end up deploying far more infrastructure than needed. The virtual machines in a Cloud infrastructure can be live migrated to another host in case user application requires more resources. Cloud providers monitor and



- predict the demand and thus allocate resources according to demand. Those applications that require less number of resources can be consolidated on the same server. Thus, datacenters always maintain the active servers according to current demand, which results in low energy consumption than the conservative approach of over-provisioning.
- ii. **Multi-tenancy:** Using multi-tenancy approach, Cloud computing infrastructure reduces overall energy usage and associated carbon emissions. The SaaS providers serve multiple companies on same infrastructure and software. This approach is obviously more energy-efficient than multiple copies of software installed on different infrastructure. Furthermore, businesses have highly variable demand patterns in general, and hence multi-tenancy on the same server allows the flattening of the overall peak demand which can minimize the need for extra infrastructure.
  - iii. **Server Utilization:** In general, on-premise infrastructure run with very low utilization, sometimes it goes down up to 5 to 10 percent of average utilization. Using virtualization technologies, multiple applications can be hosted and executed on the same server in isolation, thus lead to utilization levels up to 70%. Thus, it dramatically reduces the number of active servers.
  - iv. **Datacenter Efficiency:** The power efficiency of datacenters has major impact on the total energy usage of Cloud computing. By using the most energy efficient technologies, Cloud providers can significantly improve the PUE [7] of their datacenters. Today's state-of-the-art datacenter designs for large Cloud service providers can achieve PUE levels as low as 1.1 to 1.2, which is about 40% more power efficiency than the traditional datacenters. The server design in the form of modular containers, water or air based cooling, or advanced power management through power supply optimization, are all approaches that have significantly improved PUE in datacenters.
  - v. A number of researchers proposed different mechanism focusing different aspects of cloud services to deliver GCC facilities. Key contributing works are summarized below:

**Efficient Resource Provisioning in Compute Clouds via VM Multiplexing[8]:** In both static and dynamic provisioning, VM sizing is perhaps the most vital step. VM sizing refers to the estimation of the amount of resources that should be allocated to a VM. The objective of VM sizing is to ensure that VM capacity is commensurate with the workload.

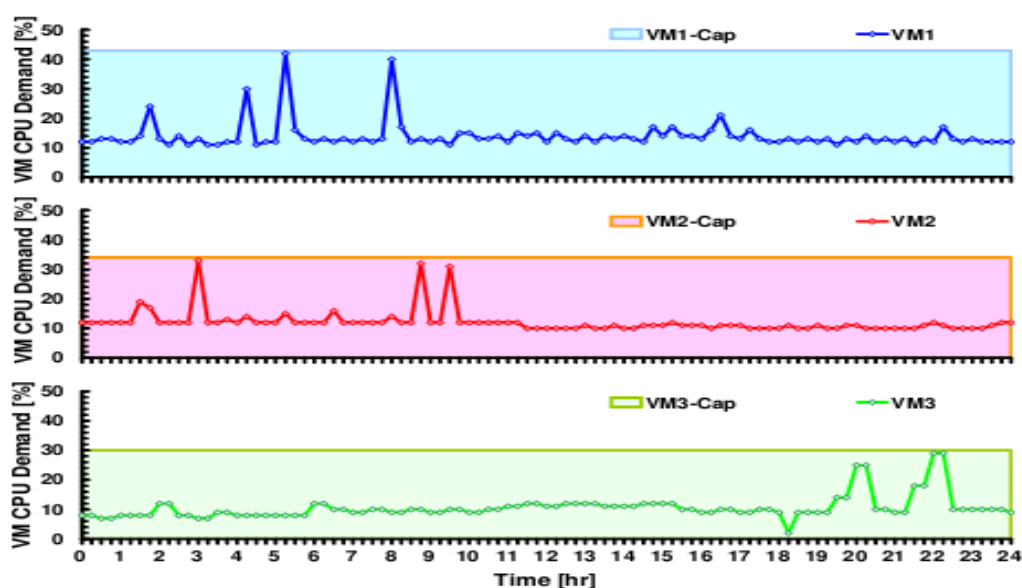


Fig. 4.5: Individual VM capacity requirements

While over-provisioning wastes costly resources, under-provisioning degrades application performance and may lose customers. In this paper, the authors advocate a joint VM provisioning approach in which multiple VMs are consolidated and provisioned based on an estimate of their

aggregate capacity needs. Joint-VM provisioning exploits statistical multiplexing among the dynamic VM demand characteristics, i.e., the peaks and valleys in one VM's demand do not necessarily coincide with the other VMs.

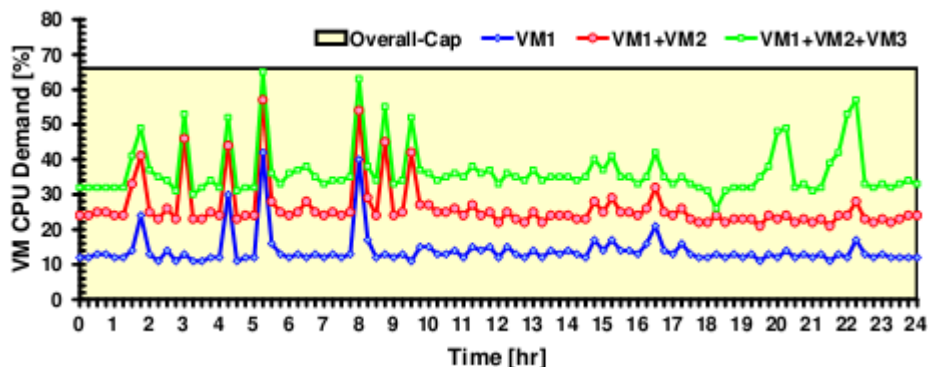


Fig. 4.6: Aggregate capacity requirement for multiplexed VMs.

While there is no direct resource management mechanism to specify joint reservations, the authors used a commonly overlooked abstraction of virtualized clusters, i.e., resource pools, to this effect. Resource pools are defined within individual hosts or a cluster to provide a virtual abstraction layer that divides resources into multiple subsets. At the cluster level, one can consider the entire resource capacity of the cluster as a single monolithic root resource pool. Then, additional child resource pools can divide the cluster capacity into exclusive partitions [9, 10]. Resource control mechanisms such as reservations and limits can also be defined at the pool level.

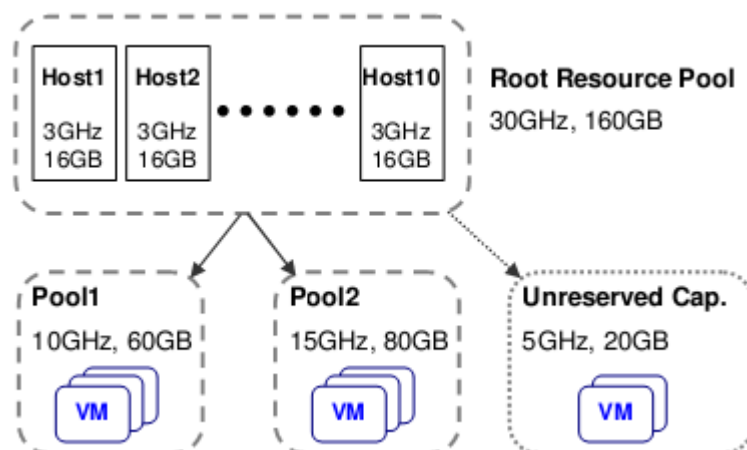


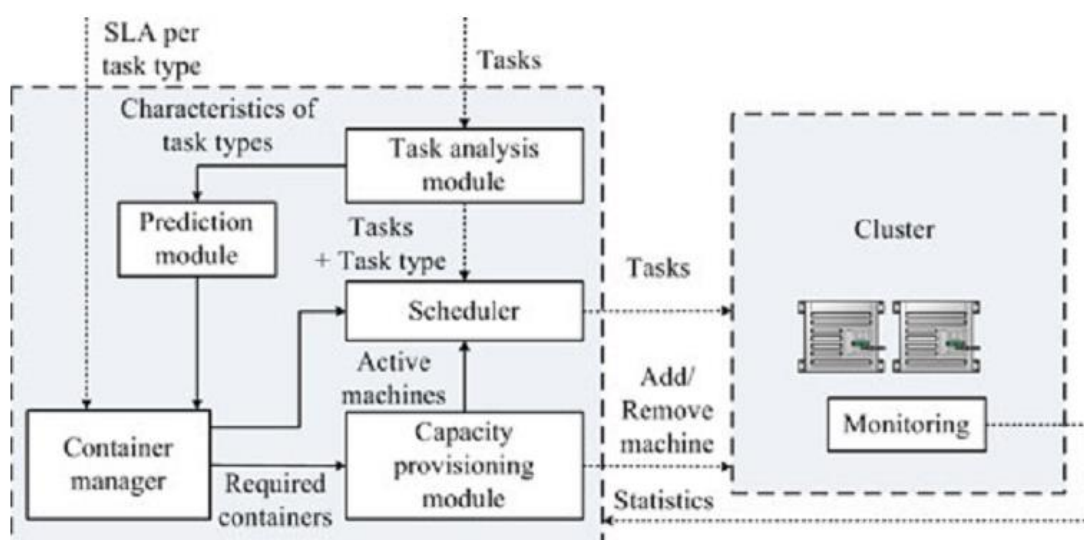
Fig. 4.7: Cluster-level resource pools

**Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud (HARMONY) [11]:** In recent years, there has been extensive research on improving data center energy efficiency [12], [13] through dynamic capacity provisioning (DCP). The goal of this technique is to dynamically adjust the number of active machines in a data center in order to reduce energy consumption while meeting the service level objectives (SLOs) of workloads. A metric of particular importance is scheduling delay [14], which is the time a request waits in the scheduling queue before it is scheduled on a machine. There is often a tradeoff between energy savings and scheduling delay. Even though turning off a large number of machines can achieve high energy savings, at the same time, it reduces service capacity and hence leads to high scheduling delay.

A key challenge that often has been overlooked or considered difficult to address in DCP is heterogeneity, which is prevalent in production cloud data centers [15]. We summarize the types of heterogeneity found in production environments as follows:

- **Machine Heterogeneity:** Production data centers often comprise several types of machines from multiple generations [16]. They have heterogeneous processor architectures and speeds, hardware features, memory and disk capacities.
- **Workload Heterogeneity:** Production data centers receive a vast number of heterogeneous resource requests with diverse resource demands, durations, priorities and performance objectives [16].

It has been reported that the differences in resource demand and duration can span several orders of magnitude [17]. Given a rise of workload requests, a heterogeneity-oblivious DCP scheme can turn on wrong types of machines which are not capable of handling these requests (e.g., due to insufficient capacity), resulting in both resource wastage and high scheduling delays.



**Fig. 4.8: Dynamic Resource Provisioning System Architecture**

**HARMONY** is a DCP framework that considers both task and machine heterogeneity. This requires:

1. An accurate characterization of both workload and machines,
2. Effectively capture the dynamic workload composition at runtime, and
3. Using the captured information to control the number of machines in the compute cluster to achieve a balance between energy savings and scheduling delay.

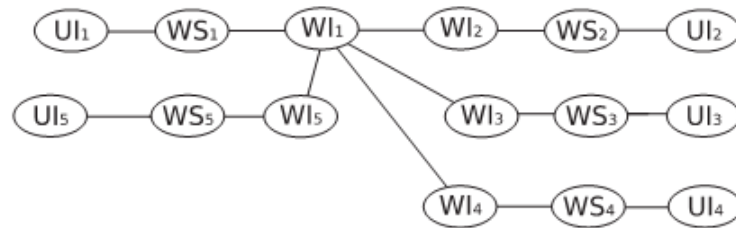
**A Tabu Search Algorithm for the Location of Data Centers and Software Components in Green Cloud Computing Networks [18]:** The authors solved the problem of designing a cloud computing network by answering the following questions:

- What potential data centers should be used in the network?
- Which data center should host each of the software components of the cloud applications?
- How many servers will be hosted at each data center?
- How will the information be routed through the network?
- What are the link capacities required to carry that information?

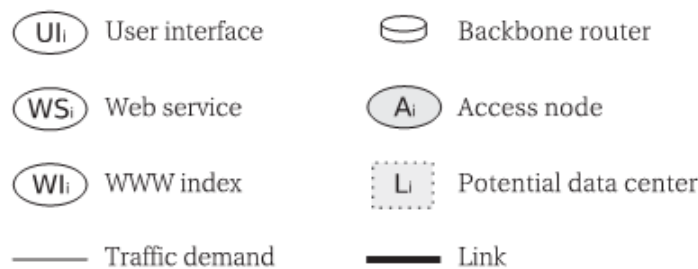
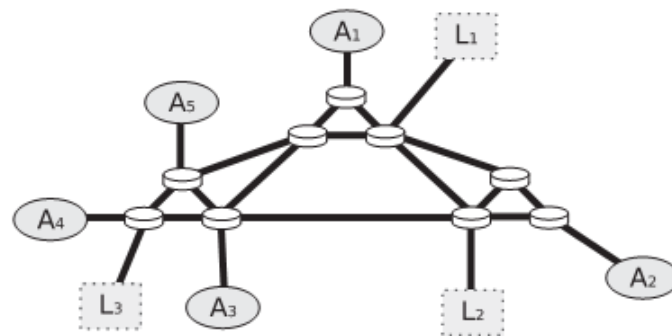
The authors have designed a multi-objective function that allows planners to weight each attribute according to their priorities. The objective function is composed of the following metrics: traffic delay, energy consumption, CO<sub>2</sub> emissions, traffic cost, server cost, data center capital expenditures (CAPEX), and data center operational expenditures (OPEX). The problem is to define the location of data centers and software components, the number of servers in each data center, the information

routing, and the capacity of each link in the network. The design objective is to minimize the network delay, the cost, the energy consumption, and the CO<sub>2</sub> emissions. The proposed problem is formalized as a mixed-integer linear programming (MILP) model and solved with a very efficient tabu search heuristic.

### $G^A$ : Application layer

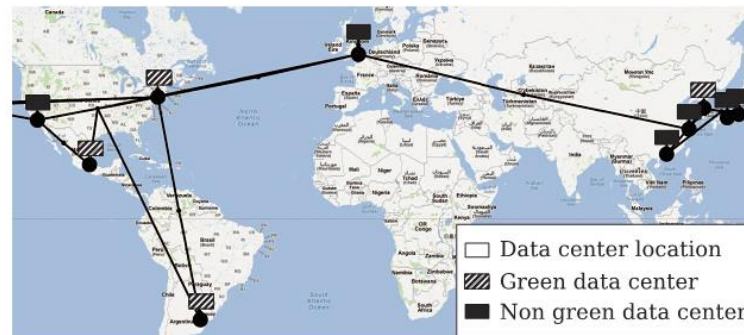


### $G^N$ : Network layer

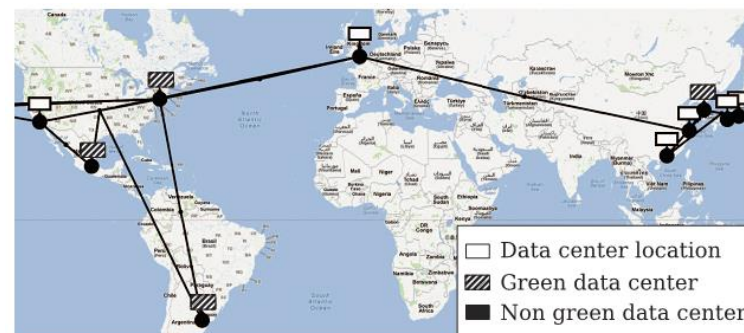


**Fig. 4.9: Network and application layers**

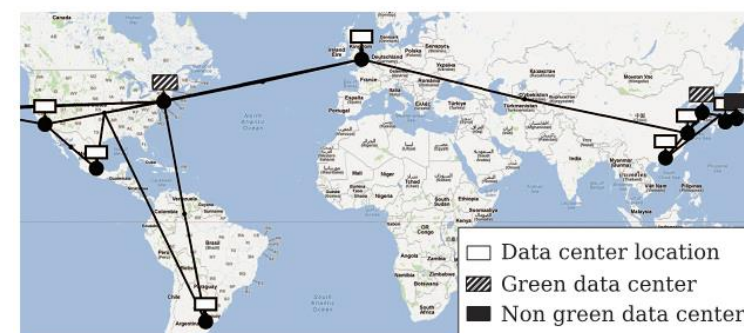
The authors evaluated the performance of Tabu Search in a case of 10 cities to analyze how the different aspects of the model interact. This case also demonstrates how planners can use the framework with multiple objectives. The network has 10 access nodes and 10 potential datacenter locations. The access nodes are shown as black circles and the potential data center locations are shown as white boxes. The application graph is a web search engine with a user interface, a web service, and a web index for each access node, and the topology of traffic demands. The case was solved in three different contexts, depending on the optimization priorities: A) delay oriented, B) pollution oriented, and C) cost oriented.



Solution A. Delay minimization.

**Fig. 4.10: Delay, Pollution and Cost Minimization in a Network**

Solution B. Pollution minimization.



Solution C. Cost minimization.

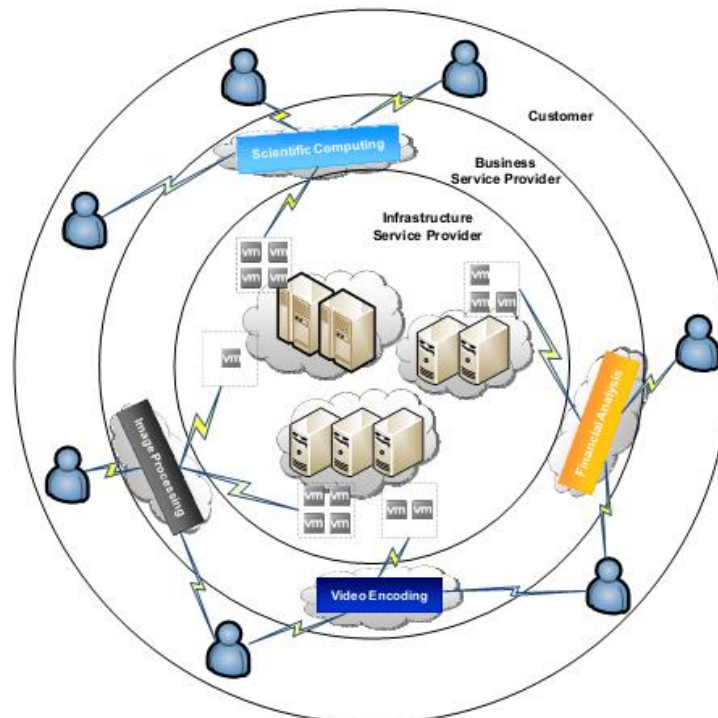
**Tradeoffs between Profit and Customer Satisfaction for Service Provisioning in the Cloud [19]:** Compared with other markets, the cloud computing market faces challenges on providing certainty to the commodities exchanged among buyers and sellers. Performance variation of cloud computing nodes is high and some nodes may have order of magnitude worse performance than other nodes [20]. It becomes a significant issue for an infrastructure service provider and a business service provider if they intend to offer viable service level agreements (SLAs) to their customers. Lack of such agreements may cause valuable applications to move away from the cloud and will jeopardize sustainable growth of cloud computing in the future.

To a certain degree, the problem arises due to the fact that each party in the cloud has its own interest. An infrastructure service provider intends to optimize its physical resource utilization for profit, which may affect the performance of VM instances running on it. On the other hand, a business service provider intends to satisfy its customers with minimal cost of renting VM instances.

The utility model in economics is commonly used to represent the need of a customer and the obligation of a provider. Under the utility model, the prices of resources provided by the infrastructure service dynamically reflect the supply and demand of these resources. The authors modeled the customer satisfaction based on the utility theory [21]. As shown in Equation 1, the authors defined the satisfaction, or utility of using a service as a function of the service price  $p$  and the response time  $t$ .

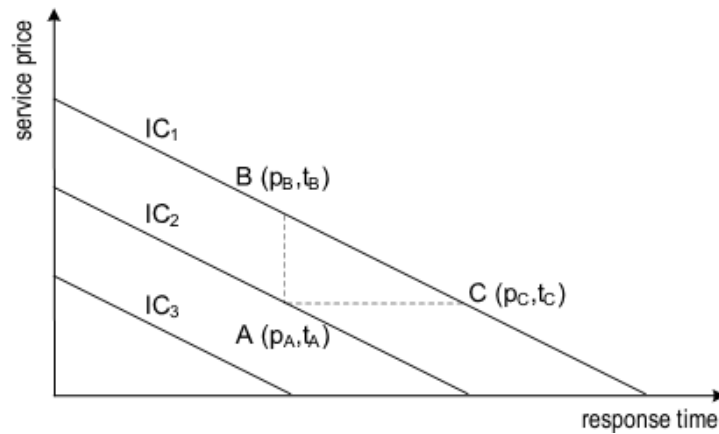
$U(p,t) = U_o - \alpha p - \beta t$  -----(1), in which,  $U_o$  is the maximum utility that the service delivers to the customer;  $\alpha$  and  $\beta$  are coefficients.  $U_o$  is proportional to the size of the service request. It is natural that serving a request of large size means high utility for a customer.

The  $\alpha/\beta$  ( $\beta/\alpha$ ) is known as marginal rate of substitution in economics, denoting the rate at which the customer is willing to give up response time (or service price) in exchange for service price (or response time) without any satisfaction change. A customer may have different levels of satisfaction, as shown in the Fig 11.



**Fig. 4.11: The three-tier cloud computing market**

With a set of SLAs between a service provider and a number of customers, the service provider is capable of making efficient decisions regarding VM instance renting to satisfy both its own profit target and customers' needs. The flexibility in price and response time can also relieve the performance variation issue in the cloud. Specially, a service provider can do the following to optimize its resource use:

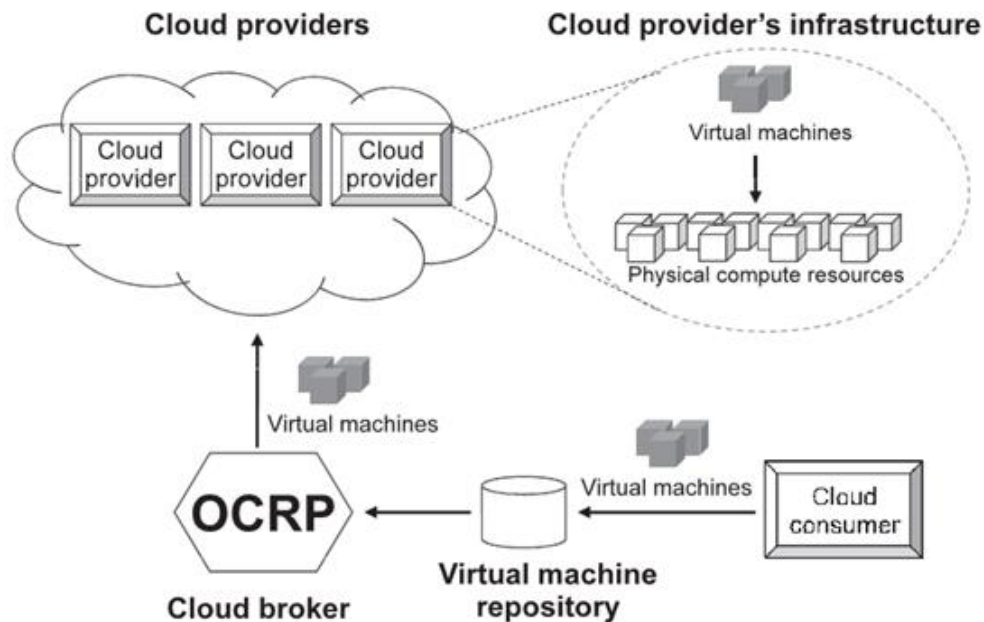


**Fig. 4.12: An example of indifference map.**

1. While maintaining a certain level of customer satisfaction, the service provider is enabled to optimize profit by reducing the response time and charging a higher service price, which means moving the point  $(p, t)$  upleft along the indifference curve, e.g., from point C to point B.
2. While keeping a profit target, the service provider can improve customer satisfaction by reducing the response time, which means moving point  $(p, t)$  left horizontally from an indifference curve to another indifference curve with higher satisfaction level, e.g., from point C to A.
3. For delayed services caused by performance variation of rented VM instances, the service provider can maintain the customer satisfaction by charging a lower service price, which means moving the point  $(p, t)$  downright along the indifference curve, e.g., from point B to point C.

**Optimization of Resource Provisioning Cost in Cloud Computing [22]:** In cloud computing, a resource provisioning mechanism is required to supply cloud consumers a set of computing resources for processing the jobs and storing the data. Cloud providers can offer cloud consumers two resource provisioning plans, namely short-term on-demand and long-term reservation plans. Amazon EC2 [23] and GoGrid [24] are, for instances, cloud providers which offer IaaS services with both plans. Pricing in on-demand plan is charged by pay-per-use basis.

With the reservation plan, the cloud consumers proactively reserve the resources in advance. As a result, the under-provisioning problem can occur when the reserved resources are unable to fully meet the demand due to its uncertainty. On the other hand, the over-provisioning problem can occur if the reserved resources are more than the actual demand in which part of a resource pool will be underutilized. It is important for the cloud consumer to minimize the total cost of resource provisioning by reducing the on-demand cost and over-subscribed cost of under-provisioning and over-provisioning.



**Fig. 4.13: System Model of Cloud Computing Environment**

The system model of cloud computing environment consists of four main components, namely cloud consumer, virtual machine (VM) repository, cloud providers, and cloud broker. The cloud consumer has demand to execute jobs. Before the jobs are executed, computing resources has to be provisioned from cloudproviders. To obtain such resources, the consumer firstly creates VMs integrated with software required by thejobs. The created VMs are stored in the VM repository. Then, the VMs can be hosted on cloud providers' infrastructures whose resources can be utilized by theVMs. The cloud broker is located in the cloud consumer's site and is responsible on behalf of the cloud consumer for provision resources for hosting the VMs. In addition, the broker can allocate the VMs originally stored in the VM repository to appropriate cloud providers.

#### 4.4 Open Research Challenges in GCC

Cloud computing is expanding to increasingly more users, applications, and devices. By simply improving the efficiency of equipment, Cloud computing cannot be claimed to be Green. What is important is to make its usage more carbon efficient both from user and provider's perspective. Cloud providers need to reduce the electricity demand of Clouds and take major steps in using renewable energy sources rather than just looking for cost minimization. To make Cloud computing Green, there are still many technological solutions required to make it a reality [25]:

- First efforts are required in designing software at various levels (OS, compiler, algorithm and application) that facilitates system wide energy efficiency. Although SaaS providers may still use already implemented software, they need to analyze the runtime behavior of applications. The compiler and operating systems need to be designed in such a way that resources can be allocated to application based on the required level of performance, and thus performance versus energy consumption tradeoff can be managed.
- To enable the green Cloud datacenters, the Cloud providers need to understand and measure existing datacenter power and cooling designs, power consumptions of servers and their cooling requirements, and equipment resource utilization to achieve maximum efficiency. In addition, modeling tools are required to measure the energy usage of all the components and services of Cloud, from user PC to datacenter where Cloud services are hosted.
- For designing the holistic solutions in the scheduling and resource provisioning of applications within the datacenter, all the factors such as cooling, network, memory, and CPU should be considered. For instance, consolidation of VMs even though effective technique to minimize



overall power usage of datacenter, also raises the issue related to necessary redundancy and placement geo-diversity required to be maintained to fulfill SLAs with users.

- Last but not the least, the responsibility also goes to both providers and customers to make sure that emerging technologies do not bring irreversible changes which can bring threat to the health of human society. The way end users interact with the application also has a very real cost and impact. For example, purging of unsolicited emails can eliminate energy wasted in storage and network. Similarly, if Cloud providers want to provide a truly green and renewable Cloud, they must deploy their datacenters near renewable energy sources and maximize the Green energy usage in their already established datacenters. Before adding new technologies such as virtualization, proper analysis of overhead should be done real benefit in terms of energy efficiency.

#### 4.5 Conclusion

Cloud computing business potential and contribution to already aggravating carbon emission from ICT, has led to a series of discussion whether Cloud computing is really green. It is forecasted that the environmental footprint from data centers will triple between 2002 and 2020, which is currently 7.8 billion tons of CO<sub>2</sub> per year [25]. There are reports on Green IT analysis of Clouds and datacenters that show that Cloud computing is “Green”, while others show that it will lead to alarming increase in Carbon emission.

In this report, first the benefits offered by Cloud computing by studying its fundamental definitions and benefits, the services it offers to end users are analyzed. Then, the components of Clouds that contribute to carbon emission and the features of Clouds that make it “Green” are discussed. Several research efforts and technologies that increase the energy efficiency of various aspects of Clouds are also discussed. For this study, several unexplored areas that can help in maximizing the energy efficiency of Clouds from a holistic perspective are identified.

Along with the quality of service and the costs, the energy consumption and the CO<sub>2</sub> emissions are fundamental considerations in regard to planning cloud computing networks. What is important is to make its usage more carbon efficient both from user and provider’s perspective. Cloud Providers need to reduce the electricity demand of Clouds and take major steps in using renewable energy sources rather than just looking for cost minimization.

#### 4.6 References

1. Gleeson, E. 2009. Computing industry set for a shocking change. Retrieved January 10,2010 from <http://www.moneyweek.com/investment-advice/computing-industry-set-for-a-shocking-change-43226.aspx>
2. New Datacenter Locations. 2008. <http://royal.pingdom.com/2008/04/11/map-of-all-google-data-center-locations/>
3. Ranganathan P, 2010, Recipe for efficiency: principles of power-aware computing.Communication. ACM, 53(4):60–67.
4. Bianchini, R., and Rajamony, R. 2004, Power and energy management for server systems, Computer, 37 (11) 68-74.
5. Rivoire, S., Shah, M. A., Ranganathan, P., and Kozyrakis, C. 2007. Joulesort: a balanced energy-efficiency benchmark, Proc. of the 2007 ACM SIGMOD InternationalConference on Management of Data, NY, USA.
6. Accenture Microsoft Report. 2010. Cloud computing and Sustainability: The Environmental Benefits of Moving to the Cloud, Nov 2010.
7. Rawson, A., Pflueger, J., and Cader, T., 2008. Green Grid Data Center Power EfficiencyMetrics. Consortium Green Grid.
8. Meng, X., Isci, C., Kephart, J., Zhang, L., Bouillet, E., Pendarakis, D. 2010. Efficient Resource Provisioning in Compute Cloudsvia VM Multiplexing, Proc. ACM Int’l Conf.Autonomic Computing (ICAC),June 7–11, 2010, Washington, DC, USA.

9. Rolia, J., Cherkasova, L., Arlit, M., and Andrzejak, A. Capacity management service for resource pools. In *International Workshop on Software and Performance*, 2005.
10. VMware Inc. *Resource Management with VMware DRS*. Whitepaper, VMware Inc., 2006.
11. Zhang, Q., Zhani, M. F., Boutaba, R., Hellerstein, J. L. Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud, *IEEE Transactions on Cloud Computing*, Vol. 2, No. 1, January-March 2014.
12. Ren, S. et al. Provably-Efficient Job Scheduling for Energy and Fairness in Geographically Distributed Data Centers, *Proc. IEEE 32nd Int'l Conf. Distributed Computing Systems (ICDCS)*, 2012.
13. Zhang, Q., Zhani, M. F., Zhu, Q., Zhang, S., Boutaba, R., and Hellerstein, J. L. Dynamic Energy-Aware Capacity Provisioning for Cloud Computing Environments, *Proc. ACM Int'l Conf. Autonomic Computing (ICAC)*, 2012.
14. Mishra, A. K., Hellerstein, J. L., Cirne, W., Das, C. R. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters, *ACM SIGMETRICS Performance Evaluation Rev.*, vol. 37, pp. 34-41, Mar. 2010.
15. Reiss, C., Tumanov, A., Ganger, G., Katz, R., Kozuch, M. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis, *Proc. ACM Symp. Cloud Computing*, 2012.
16. Sharma, B., Chudnovsky, V., Hellerstein, J. L., Rifaat, R., Das, C. R. Modeling and Synthesizing Task Placement Constraints in Google Compute Clusters, *Proc. Second ACM Symp. Cloud Computing (SOCC)*, 2011.
17. Boutaba, R., Cheng, L., Zhang, Q. On Cloud Computational Models and the Heterogeneity Challenge, *J. Internet Services and Applications*, vol. 3, pp. 77-86, 2012.
18. Larumbe, F., Sanso, B., A Tabu Search Algorithm for the Location of Data Centers and Software Components in Green Cloud Computing Networks, *IEEE Transactions On Cloud Computing*, Vol. 1, No. 1, January-June 2013.
19. Chen, J., Wang, C., Zhou, B. B., Sun, L., Lee, Y. C., Zomaya, A. Y. Tradeoffs between Profit and Customer Satisfaction for Service Provisioning in the Cloud, *High-Performance Parallel and Distributed Computing (HPDC)*, June 8–11, 2011, San Jose, California, USA
20. Armbrust, M., Fox, A., Grith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M. Above the clouds: a Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California at Berkeley, February 2009.
21. Mankiw, G. *Principles of economics*. South-Western Pub, 2008.
22. Lee, B. S., Niyato, D. Optimization of Resource Provisioning Cost in Cloud Computing, *IEEE Transactions On Services Computing*, Vol. 5, No. 2, April-June 2012.
23. Amazon EC2, <http://aws.amazon.com/ec2>, 2012.
24. GoGrid, <http://www.gogrid.com>, 2012.
25. Green Cloud computing and Environmental Sustainability, <http://www.cloudbus.org/papers/Cloud-EnvSustainability2011.pdf>
26. Greenpeace International. 2010. Make IT Green <http://www.greenpeace.org/international/en/publications/reports/make-it-green-Cloudcomputing/>

”

**CHAPTER 5**  
**CLOUD TECHNOLOGY**

## 5. CLOUD TECHNOLOGY

### 5.1 Overview

Cloud technologies are built on existing technologies such as virtualization and clustering to virtualize hardware, software, storage, and networking resources into flexible units that are quickly allocated to meet demand. So rather than the old static model of dedicated hardware servers for various tasks, and static network and storage configurations, all of those formerly specialized devices are assimilated into a common resource pool. It's a more efficient use of hardware, and very fast to scale up or down according to demand. You can even configure self-service for users so they can grab whatever they need when they need it.

Private clouds are hosted on your own premises, and there are public clouds like Amazon's EC2 and the Rackspace Cloud. You can combine private and public clouds in many useful ways. For example, keep your sensitive data locked away in your private cloud, and use a public cloud for sharing, testing, and extra non-sensitive storage.

All computing resources are shareable in a cloud, and there are three basic service models:

- SaaS, software as a service
- PaaS, platform as a service
- IaaS, infrastructure as a service

SaaS is centrally-hosted application software accessed by client software, with data typically kept on the server for access from any networked computer. Yes, just like in the olden client-server days, but the modern twist is to stuff everything through a Web browser. Using a Web browser as the client has its down sides, starting with HTTP, which was never designed for complex computing tasks, but by gosh we're making it haul water, chop wood, and dig ditches, and it's doing it cross-platform. SaaS is popular with software vendors because it reduces their support costs, gives them more control, and at long last supports that coveted grail of the monthly subscription model. It's nice for customers as well, because they don't have to hassle with installation and maintenance.

PaaS is a nice option for customers who want more control of their datacenter, but not all the headaches of system and network administration. An example of this is managed cloud Web hosting where the host takes care of hardware, operating systems, networking, load balancing, backups, and updates and patches. The customer manages the development and configuration of whatever software they want to use. It's like sitting down to a fully-configured datacenter and getting right to work.

IaaS can be thought of as virtual bare hardware that the customer manages like a physical server, with control of all the software and configuration. You could also call it HaaS, hardware as a service.

### 5.2 Research Trends in Cloud Computing

#### 5.2.1 Software Engineering of Large-Scale systems

Large-scale systems are inherently difficult to manage. The level of complexity and rate of change in such systems presents a constantly moving target, which requires substantial talent and manpower to manage. Debugging large-scale cloud applications is exceptionally difficult, due to the scale of a deployment, the abstraction from the underlying hardware, and the nature of a remote deployment.

Research should focus on a tool chain which allows engineers to collect, visualize and inspect the state of jobs running across many thousands of cores. Research into this area can be built on top of existing cloud platforms and wouldn't necessarily require low-level access to large-scale test-beds. In academia, however, this area has received insufficient attention hitherto the means to develop and debug large-scale applications are under researched.

There remains only a small body of research in this crucial area which limits the ability of academics to translate prototypes into real world software. A lack of knowledge forces current research to be discarded or significantly rewritten in order to be applicable to large scale systems. Through further investigation into the software engineering of large-scale systems, academics can produce superior cloud research which supports the scale of systems that are now becoming commonplace.

### **5.2.2 Cloud Service Platform**

Platform as a Service (PaaS) clouds aim to offer a higher level of abstraction, away from bare-bones infrastructure services. At this level of the cloud stack the focus is on automating certain tasks, which if done by manually writing programs can be a time consuming task. In other words an abstract environment, which incorporates elasticity and on-demand service features offered through an easy to use interface.

The research focus of PaaS needs to be placed on building environments, which make use of cloud infrastructure provided by multiple vendors, and at the same time providing a high-level interface that will allow scientists to quickly and easily access the resources. Such research can reap maximum benefit if based on user driven requirements and directed towards developing stand-alone and independent environments, which make use of the variety of PaaS features offered by the providers.

### **5.2.3 Cloud Scalability**

The most significant benefits of cloud computing are obtained by applications that scale over a large number of resources. Traditionally, in grid and cluster environments the resources requested for an application had to be provisioned before its execution. Cloud computing, however, facilitates dynamic resource provisioning and provides a high level abstract interface for doing this. Within this area, we believe that there are numerous challenges, which can be addressed for driving academic cloud research forward.

The first challenge within elasticity is related to resource provisioning, an area which overlaps with scalability. Through the Infrastructure as a Service layer large pools of abstracted resources can be quickly provisioned. This facilitates a wide range of research opportunities. Notably the issues of efficient provisioning are not yet fully addressed. There is a need for research which investigates how to avoid under and over provisioning, how to provision for performance and cost, and how to efficiently compute requirements.

The second challenge is related to workload management the second challenge is related to workload manage. How can frameworks use workload information for provisioning resources, or is it possible to anticipate the workload which can be used to provision resources thereby reducing the effects of variable times in provisioning resources.

### **5.2.4 Fault tolerance of cloud environment**

Fault tolerance of cloud is another area where the academy can contribute significantly through its research. This challenge is also related to modeling elasticity. An attractive feature of the cloud is its abstract view of elasticity and how easily resources can be provisioned. One important question that arises is can the abstract view also present different views of the system.

For example, what level of performance can be obtained by provisioning  $n$  resources when the workload increases by  $m\%$ , what cost models can best capture these scenarios, what additional costs will be incurred when  $n$  resources are added.

How to use abstracted redundancy of cloud environments to add new vm-instances when a module crashes in a Service Oriented Architecture can be an exciting area of future research interest.

### 5.3 Inside OpenStack

OpenStack is a complex beast containing multiple components. The core components are OpenStack Compute, OpenStack Glance, OpenStack Identity Service, and OpenStack Object Store.

OpenStack Compute is the virtual machine provisioning and management module. Its development name is Nova, so when you read about Nova it's the same thing. It supports multiple hypervisors including KVM, QEMU, LXC, and XenServer. Compute is the mighty tool that controls the whole works: networking, CPU, storage, memory, creating, controlling, and removing virtual machine instances, security, and access control. You control all this from the command-line or from a graphical Web-based dashboard.

OpenStack Glance, the OpenStack Image Service, manages virtual disk images. Glance supports Raw, Hyper-V (VHD), VirtualBox (VDI), Qemu/KVM (qcow2), and VMWare (VMDK, OVF) virtual machine images, and it also supports Amazon Machine Images (AMI). You can do all kinds of cool things with Glance: stream virtual disk images, configure public and private images and control access to them, and of course create and destroy them.

OpenStack Object Store, as the name suggests, manages storage. It is a distributed storage system for managing all types of storage: archives, user data, virtual machine images, and the hardware they're stored on. There are multiple layers of redundancy and automatic replication, so a failure in a node doesn't result in data loss, and recovery is automatic.

The Identity Service manages users and projects.

The OpenStack project is an open source cloud computing platform that supports all types of cloud environments. The project aims for simple implementation, massive scalability, and a rich set of features. Cloud computing experts from around the world contribute to the project.

OpenStack provides an Infrastructure-as-a-Service (IaaS) solution through a variety of complementary services. Each service offers an application programming interface (API) that facilitates this integration. The following table provides a list of OpenStack services:

Service	Project name	Description
Dashboard	Horizon	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.
Networking	Neutron	Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a RESTful, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.

Service	Project name	Description
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Shared services		
Identity service	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	Glance	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.

#### 5.4 Before you begin

For a functional environment, OpenStack doesn't require a significant amount of resources. We recommend that your environment meets or exceeds the following minimum requirements:

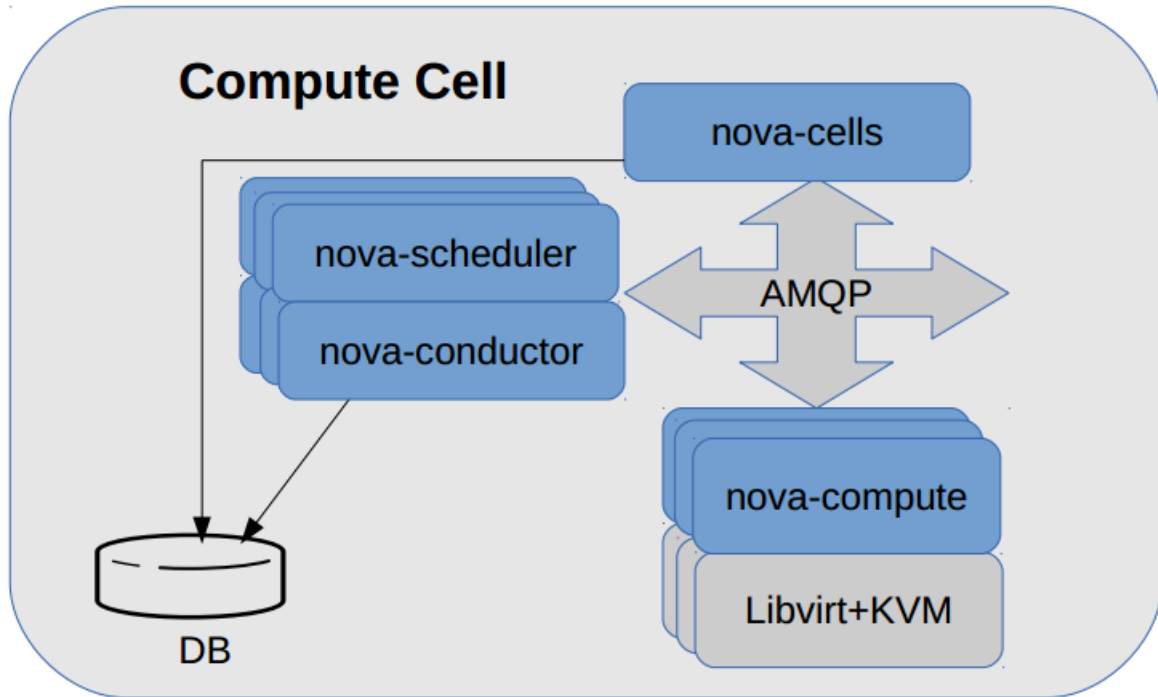
- Controller Node: 1 processor, 2 GB memory, and 5 GB storage
- Network Node: 1 processor, 512 MB memory, and 5 GB storage
- Compute Node: 1 processor, 2 GB memory, and 10 GB storage

To minimize clutter and provide more resources for OpenStack, we recommend a minimal installation of your Linux distribution. Also, we strongly recommend that you install a 64-bit version of your distribution on at least the compute node. If you install a 32-bit version of your distribution on the compute node, attempting to start an instance using a 64-bit image will fail.

In this guide I explain how to install Openstack in a single physical node. I install the nova controller and a compute node in this node. The aim of this article is to get you started with Openstack IaaS with minimum effort in a short period of time.

The CLOUD's primary purpose consists in providing services (including resources, applications tools etc.) under the conditions of the modern IT environments (see above). In the CLOUD case this means in particular a focus on increasing availability of services, data, or even just the infrastructure – with the main intention to reduce cost for resource utilisation.

Historically CLOUD systems arise from the need to ensure that this availability and related quality criteria can be met, even if the load is highly dynamic, i.e. the number of requests, the data size, the workload of the task etc. all vary over time. CLOUDs are the new form of utility computing. Most current CLOUD offerings realise elasticity through dynamic replication of the service instance, respectively an accordion image within the infrastructure. CLOUD systems build up on the internet of service principle to expose the services in a highly accessible fashion, i.e. with minimal configuration and device requirements. The fact is that CLOUDs can reduce the overhead for managing and administering resources through automation and outsourcing, and should reduce the overhead for creating highly available and reliable services.



## 5.5 What you need

The steps below can be followed using one physical node. The node should possess two network interfaces. One of them could be a virtual one. I have tested this on Ubuntu 12.04 LTS 64 bit server. The memory and storage requirements of the node depend on how many virtual machines you run on Openstack once it is ready. For example, if you plan to run 10 virtual machines with 256Mb memory and 5Gb HD each, then you need at least 3G memory and 60Gb hard disk for the node. You also need an internet connection to download the necessary Openstack software.

Note the installation described in this document is in no way production ready. You may need to do a lot of enhancements, feature additions to make it such.

## 5.6 Installation Steps

Step 1: Install Ubuntu server

Install Ubuntu server as you do any normal installation. Please refer to good Ubuntu documentation for this. During the installation steps, do the following.

- Create a user account on the host machine (say nova).
- Install openssh.
- Assign hostname (say openstack). Assign domain name (say demo.com)
- Assign static ip (say 192.168.16.20)
- Give gateway to access internet (say 192.168.16.1). I assume here you have a wired connection to the internet. Instead, if you have a wireless connection, you can let it connect to the internet using dhcp.

You can do the above steps once the Ubuntu installation is finished, as well like below

- Create user account (say nova)
- ```
$ sudo /usr/sbin/adduser nova
```
- Install openssh
- ```
$ sudo apt-get install openssh-server (to ssh into instance)
```
- Assign static ip by editing /etc/network/interfaces file
- ```
auto eth0
iface eth0 inet static
```



```
address 192.168.16.20
netmask 255.255.252.0
gateway 192.168.16.1
auto eth1
iface eth1 inet manual
up ifconfig eth1 up
Then
$ sudo ifup eth0
$ sudo ifup eth1
- Assign hostname and domain name by putting an entry in /etc/hosts file as in
192.168.16.20 openstack.demo.com openstack
```

Step2:

Log in using nova account you created.

```
$ sudo apt-get update
```

Step3:

Checkout the Installation Scripts

```
$ sudo apt-get -y install git
```

```
$ git clone https://github.com/damitha23/openstack.git
```

```
$ cd openstack
```

```
$ unzip OpenStackInstaller.zip
```

Note that content of OpenStackInstaller folder has scripts I took from <https://github.com/uksysadmin/OpenStackInstaller.git> maintained by Kevin Jackson <[kevin@linuxservices.co.uk](mailto:kevin@linuxservices.co.uk)> <https://twitter.com/#!/itarchitectkevirc.freenode.org>: uksysadmin

Step4: Installing Openstack

```
$ cd /home/nova/OpenStackInstaller
```

Modify oscontrollerinstall.sh as per your requirements and execute. It will take couple of minutes to install Openstack.

Also modify the OSinstall.sh to add following configuration that would go into nova.conf

```
--rpc_response_timeout=<new timeout in seconds>
```

Give a sufficient response timeout to avoid timeout errors.

Example oscontrollerinstall.sh

```
./OSinstall.sh -T all -C openstack.demo.com -F 192.168.16.128/25 -f 192.168.17.128/25 -s 126 -P eth0
```

```
-p eth1 -t demo -v kvm
```

Important: The virtualization type here I used is kvm.

Note that I use -T all options since I install in this server both controller and a compute node.

With -C parameter we give the hostname of the node. You should have an entry in the /etc/hosts file for this as following.

```
192.168.16.20 openstack.demo.com openstack
```

If your node ip regularly change it is good idea to have following kind of entry in /etc/rc.local file so that it will automatically add that entry when node bootup

```
ip=`/sbin/ifconfig eth0 | grep 'inet addr:' | cut -d: -f2 | awk '{ print $1 }'`
```

```
echo $ip openstack.demo.com openstack >> /etc/hosts
```

Note that here ip is taken from eth0 interface. You may need adjustments.

With -F parameter we give the floating ip range for the project.

With -f parameter we give the fixed ip range for the project.

With -s parameter we give number of nodes in the private network.

I use eth1 as private interface. eth0 as public interface. For the public ips(floating ips) we should give an valid range from the network where the host machine took IP. So a valid floating ip subnet would be 192.168.16.128/25. You can calculate such an range from the subnet calculator in link [1] or [2]

A valid fixed ip subnet would be 192.168.17.128/25. Note that if the floating ip's are exhausted, then there will be errors and instance would not be created. To avoid this situation, make sure that you allocate as many as floating ips, at least, as the fixed ips. Now you can access Openstack UI from <http://openstack.demo.com> using

Username:admin

Password:openstack

You may need to add an host entry in the node where your browser reside when giving the above url as in

```
192.168.16.20 openstack.demo.com openstack
```

Now you can manage your Openstack environment from the UI interface.

If one of your interface is a virtual interface(This could be the case when are installing on a laptop) your install command could be like following

```
./OSinstall.sh -T all -C openstack.demo.com -F 192.168.16.128/25 -f 192.168.17.128/25 -s 126 -P eth0
```

```
-p eth0:0 -t demo -v kvm
```

Make sure eth0:0 is defined as following

```
auto eth0:0
```

```
iface eth0:0 inet manual
```

And make sure it is up by using

```
$ ifup eth0:0
```

Step5: Upload an Image

From this step on you can execute the commands as normal user. I upload an ubuntu image to glance.

For kvm virtual machine download a base ubuntu image precise-server-cloudimg-amd64-disk1.img from <http://cloud-images.ubuntu.com/precise/current/> and create a folder called /home/nova/upload folder and copy the image into it.

Modify /home/nova/OpenStackInstaller/uploadimage.sh and execute to upload the image.

An example uploadimage.sh would be

```
./imageupload.sh -a admin -p openstack -t demo -C openstack.demo.com -x amd64 -y ubuntu -w 12.04
```

```
-z /root/upload/precise-server-cloudimg-amd64-disk1.img -n cloudimg-ubuntu-12.04
```

Here openstack.demo.com is the hostname of the openstack controller.

Execute

```
$ cd OpenStackInstaller
```

```
$ source ./demorc
```

```
$ nova image-list
```

command to see whether your newly uploaded image appear in the image list.

Step6: Testing the Controller

```
$ cd OpenStackInstaller
```

```
$ source ./demorc
```

Now add a keypair. It is highly recommended that you use your own keypair when creating

instances. For example suppose you create an instance as normal user, using a keypair owned by root user. You may succeed in creating your instance. But you will get permission denied exception when trying to ssh to that instance.

```
$ nova keypair-add wso2 > wso2.pem
```

Set permission for the private key

```
$ chmod 0600 wso2.pem
```

You can see the created key listed

```
$ nova keypair-list
```

Allow needed ports for the default security group.

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

```
$ nova secgroup-add-rule default tcp 80 80 0.0.0.0/0
```

```
$ nova secgroup-add-rule default tcp 443 443 0.0.0.0/0
```

```
$ nova secgroup-add-rule default tcp 3306 3306 0.0.0.0/0
```

```
$ nova secgroup-add-rule default tcp 8080 8080 0.0.0.0/0
```

Now list the images and select an image id to create an instance from it

```
$ nova image-list
```

```
$ nova boot --key_name=nova-key --flavor=1 --image=<image id> <instance name>
```

Instead of the above command you can use the following command if you need to pass some user data into the instance you want to create.

```
$ nova boot --key_name=nova-key --flavor=1 --image=<image id> --user_data=/root/client/payload.zip <instance name>
```

Now see whether your instance is up and running. Look for the running instances ip.

```
$ nova list
```

```
$ ssh -i wso2.pem ubuntu@ipaddress
```

If you can access the virtual machine instance then you have successfully created a controller with a compute node in it. Log into the nova mysql database running in the controller machine and observe that there is a compute node entry in the compute\_nodes table.

```
$ mysql -uroot -popenstack
```

Note that mysql password is defined in the OpenStackInstaller/OSinstall.sh file.

```
mysql>use nova
```

```
mysql>select id, created_at from compute_nodes;
```

You should see one compute node entry in the table. Now from your Openstack node you can start playing with creating/deleting your new instances. You can monitor the /var/log/nova/nova-compute.log to see the status of creating the nodes. You can create more and more instances and verify that in both compute nodes until you see a short, un-descriptive message that basically say your quota has exceeded.

## 5.7 Troubleshooting

### 5.7.1 Cannot ssh to the instance:

Make sure you have enabled tcp port

Using the nova command-line tool:

```
$ nova secgroup-add-rule default tcp 22 22 -s 0.0.0.0/0
```

If you still cannot ping or SSH your instances after issuing the nova secgroup-add-rule commands, look at the number of dnsmasq processes that are running. If you have a running instance, check to see that TWO dnsmasq processes are running. If not, perform the following

as root:

```
$ sudo killall dnsmasq
```

```
$ sudo service nova-network restart
```

### 5.7.2 When installing nova essex into a new box dpkg error occur and then mysql configuration take a long time and fail:

This happen when you forget to do an apt-get update before starting to install nova essex. This could not be corrected until doing a fresh installation again.

Your applications deployed in instances cannot be accessed

Make sure you have enabled your application port.

Using the nova command-line tool:

```
$ nova secgroup-add-rule default tcp 8080 8080 -s 0.0.0.0/0
```

Note that you need to replace 8080 above with the port your application is running.

### 5.7.3 Cannot shutdown the instance:

Sometimes even after terminate command is executed on an instance it is not terminated but go to shutoff state. At such moments try restarting nova services.

### 5.7.4 Error returned when creating the very first instance

Make sure that you public and private interfaces are up

Eg: sudo ifconfig eth1 up

### 5.7.5 Timeout: Timeout while waiting on RPC response

Sometimes when creating instances you get the response timeout error. The default request timeout for nova is 60 seconds. To increase this add following entry to /etc/nova.conf and restart nova services

```
--rpc_response_timeout=<new timeout in seconds>
```

### 5.7.6 Successfully added compute node but cannot create instances in that node

When instances are created in that node the instance state is in ERROR. In the compute node log we have

```
libvirtError: Unable to read from monitor: Connection reset by peer
```

To avoid this make sure that you have commented out the following three entries in the compute nodes /etc/nova.conf

```
##--novncproxy_base_url=http://192.168.16.20:6080/vnc_auto.html
```

```
##--vncserver_proxyclient_address=192.168.16.20
```

```
##--vncserver_listen=192.168.16.20
```

If not comment them out and restart nova services in the compute node.

Instances are not created

- Check whether both interfaces of the controller is up and all compute node interfaces are up. If not make them up and then restart nova services.

### 5.7.7 Disaster Recovery

Nova instances can be rebooted using

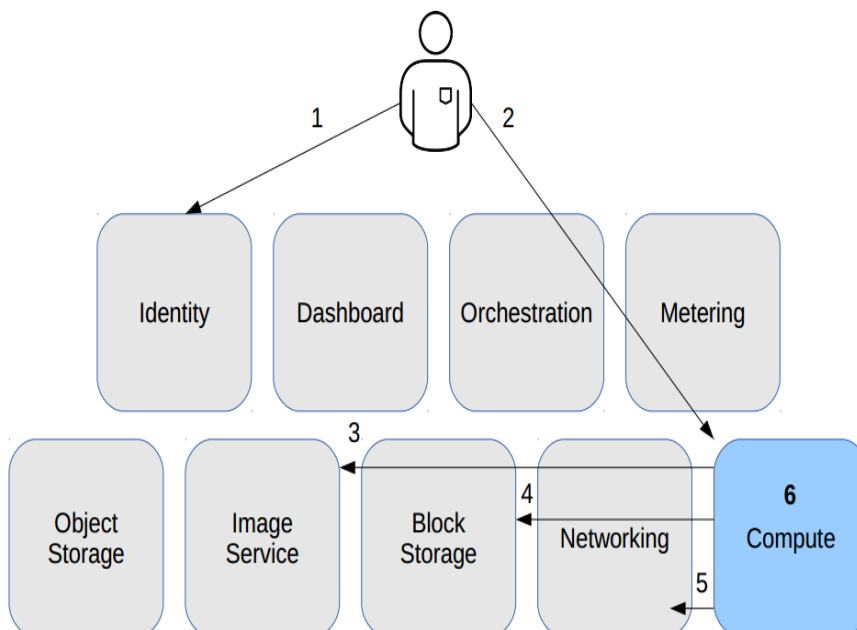
```
$ nova reboot <instance id>
```

Rebooting the vm as above solved it. But now I could ping to the instance but connection is refused when ssh to it. Then I cd to OpenStackInstaller and executed

```
$ sudo restartservices.sh
```

You may need to run this command twice if you see any warning/error first time. Then that problem is solved too.

## 5.8 Instance Boot Setup



## 5.9 Future Perspective of Cloud

The general cloud computing approach discussed so far, as well as the specific implementation of a cloud continues a number of research directions, and opens some new ones. For example economy-of-scale and economics of image and service construction depends to a large extent on the ease of construction and mobility of these images, not only within a cloud, but also among different clouds. Underlying that is a considerable amount of meta-data, Cloud provenance data, and in general metadata management, is an open issue. The classification we use divides provenance information into:-

- Cloud Process provenance – dynamics of control flows and their progression, execution information, code performance tracking, etc.
- Cloud Data provenance – dynamics of data and data flows, file locations, application input/ output information, etc.
- Cloud Workflow provenance – structure, form, evolution of the workflow itself.

## 5.10 Conclusion

Cloud computing builds on decades of research in virtualization, distributed computing, utility computing, and, more recently, networking, web and software services. It implies a service-oriented architecture, reduced information technology overhead for the end-user, great flexibility, reduced total cost of ownership on demand services and many other things.

Cloud computing is seen by many as the next wave of information technology for individuals, companies and governments. The abundant supply of information technology capabilities at a low cost offers many enticing opportunities. In addition to reducing operational costs, cloud technologies have become the basis for radical business innovation and new business models, and for significant improvements in the effectiveness of anyone using information technology – which, these days, increasingly means most of the world.

Like any new technology advancement, cloud computing also creates disruptive possibilities and potential risks. The fact that cloud computing involves the aggregation of computing power, and more importantly, information, has become a source of increasing concern. Users, providers and government policy-makers are asking many questions about the current use and future evolutionary path of cloud computing.

**CHAPTER 6**  
**CLOUD COMPUTING IN BUSINESS**

## 6. CLOUD COMPUTING IN BUSINESS

### 6.1 Introduction

Cloud computing is the result of evolution and adoption of existing technologies and paradigms. The goal of cloud computing is to allow users to take benefit from all these technologies, without the need for in depth knowledge or expertise with each one of them. Cloud computing aims to cut costs and help users focus on their core business, instead of being impeded by IT obstacles. Although the use of cloud services is still in its formative years in many emerging markets, the adoption of cloud computing is becoming more prevalent. This steadily increasing switch to the cloud by an assorted range of organizations has fueled the need for providers to invest in new data centers and cloud infrastructures as well as related offerings such as security and management services. An informed understanding of cloud computing and its benefits, limitations and risks is the key to successfully embrace the opportunities this new computing paradigm offers.

### 6.2 The emerging market of cloud services

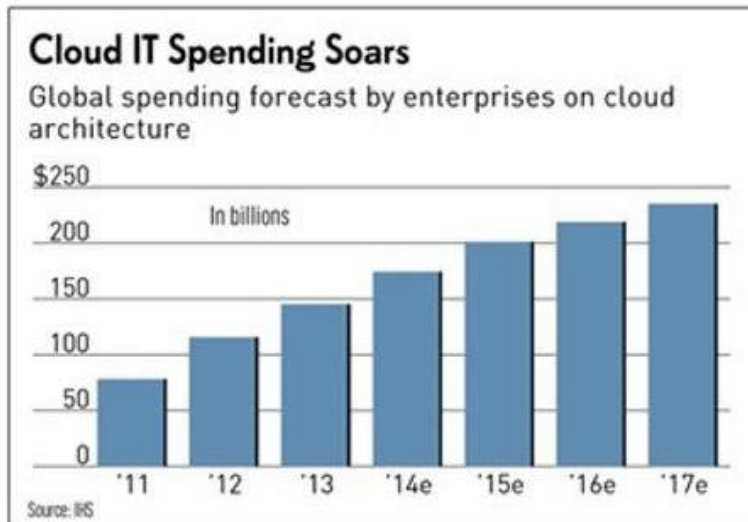
The underlying concept of cloud computing dates back to the 1950s, when large-scale mainframe computers were seen as the future of computing, and became available in academia and corporations, accessible via thin clients/terminal computers, often referred to as "static terminals", because they were used for communications but had no internal processing capacities. To make more efficient use of costly mainframes, a practice evolved that allowed multiple users to share both physical access to the computer from multiple terminals as well as CPU time. The cloud world inherited some concepts from the mainframe world that came before it. The consumption-based pricing model, Linux virtual machines, and multi-tenancy came from the previous mainframe generation.

In 1999, Salesforce.com was one of the first to invest in cloud computing, they introduced the concept of delivering enterprise applications through a simple website. Amazon launched their Amazon Web Services in 2002, which is now the largest IaaS in the world. Then in 2006 came Google Docs which spread the word of cloud computing and became the lead in increasing mass awareness.

The cloud services have increased greatly in recent years. There are several surveys done by various reputed organizations over the years that project the rapid growth of cloud computing. Some key points from these surveys are -

- By 2014, businesses in the United States will spend more than \$13 billion on cloud computing and managed hosting services.
- 82% of companies reportedly saved money by moving to the cloud.
- 80% of cloud adopters saw improvements within 6 months of moving to the cloud.
- More than half of the survey respondents say their organization currently transfers sensitive or confidential data to the cloud.
- 56% of the survey respondents trust the ability of cloud providers to protect sensitive and confidential data entrusted to them.
- 59% of all new spending on cloud computing services originates from North American enterprises, a trend projected to accelerate through 2016.

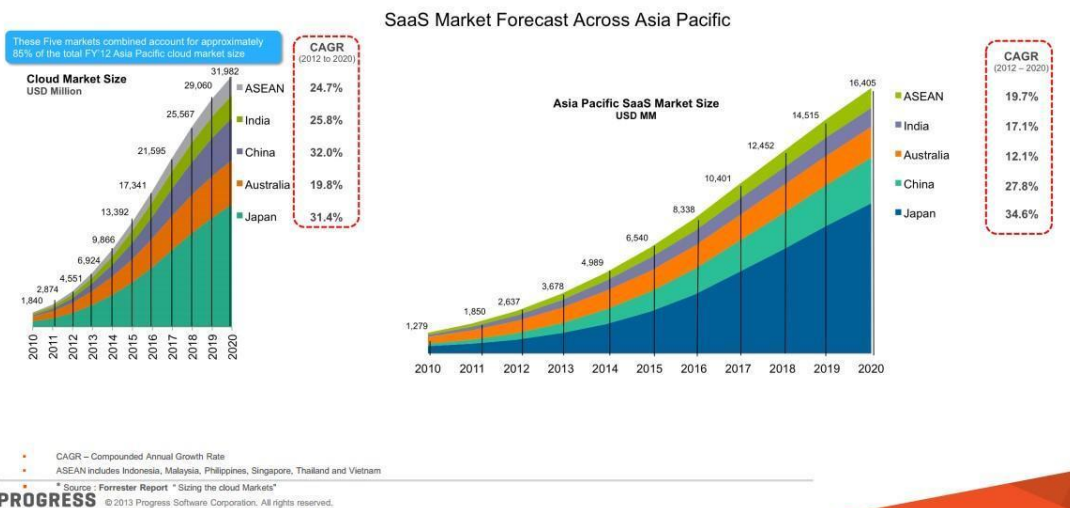
According to IHS Technology, cloud related tech expense by businesses is forecast to triple from 2011 to 2017. By 2017, enterprise spending on cloud computing will amount to a projected \$235.1B, triple the \$78.2B spent in 2011 according to the research firm's analysis. In 2014, global business spending for infrastructure and services related to the cloud will reach an estimated \$174.2B, 20% more than the amount spent in 2013.



Although cloud utilization has steadily become the standard in more established markets like the U.S. and the U.K., it is also gaining a foothold in emerging markets with a focus in Asia. The adoption curves aren't uniform across all emerging markets; the cloud offers significant advantages in these markets and companies are experiencing these benefits regardless of business size.

According to Forrester, the Asia-Pacific cloud computing market will grow from \$6.9B in 2013 to \$31.9B in 2020. The following graphic illustrates the forecasted growth for the Asia Pacific and the SaaS Market Forecast across Asia-Pacific.

### Asia Pacific Cloud Market Forecast

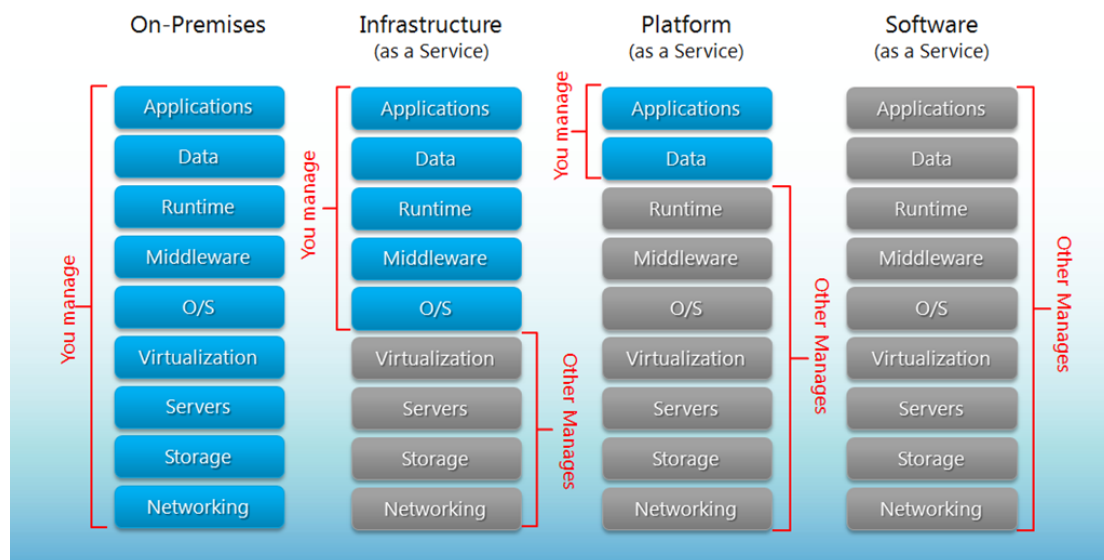




### 6.3 Different types of cloud services available in the market

The Cloud services market is crowded. The upside of this is that an organization can select Cloud services from countless providers and from a huge variety of cloud offerings. The downside is that it makes it difficult to find the viable cloud provider that specifically meets the organization’s needs. The chart below breaks down the responsibilities, from the traditional and totally on-premises solution on to the complete purchased software as a service idea, and pointing out which portions of the stack is users responsibility vs. those provided by the service provider.

## Separation of Responsibilities



#### 6.3.1 Infrastructure as a Service (IaaS)

IaaS definitely is the least disruptive cloud service that a business can adopt. It is at the very bottom of the IT stack and serves as an enabler for the cloud services above such as, PaaS or SaaS. Cloud providers in the IaaS space compete on their performance, pricing, and SLAs. Some IaaS providers are:



Some example cloud services that IaaS providers offer are:

**Compute:** This Cloud service offers CPU cycles, memory storage, network, and, depending on the Cloud provider, management for those layers.

**Backup and Recovery:** Many small and medium IT organizations don’t have the staff and/or physical space required to operate an off-site data center but still have the requirement (for compliance purposes or simply because they want to stay in business after a disaster) to recover critical services and related data in the event of a loss at their primary location. Backup and recovery cloud services address exactly this requirement by offering an on-demand and pay-per-use infrastructure.

**Web Hosting:** A Cloud service that offers the traditional web hosting but with cloud characteristics such as being on-demand, pay-per-use, and highly scalable.

**Storage as a Service (StaaS):** StaaS is generally seen as a good alternative for a small or mid-sized business that lacks the capital budget and/or technical personnel to implement and maintain their own storage infrastructure. StaaS is also being promoted as a way for businesses to mitigate risks in disaster recovery, provide long-term retention for records, and enhance both business continuity and availability.

### 6.3.2 Platform as a Service (PaaS)

PaaS is good for businesses who want to develop applications but don't want to invest heavily in development environments (i.e., hardware, operating system, development software) and don't want to operate it from in-house. In general, PaaS can provide services for the entire lifecycle of applications, from design to retirement. Some PaaS service providers are:



**General-purpose:** Many PaaS offerings are geared towards the development of web applications. Besides providing the development environment for web applications, PaaS offers the platform for mobile applications development.

**Testing:** IT organizations can take advantage of the many PaaS offerings in the testing space. These Cloud services allow for testing software; i.e., web applications or mobile applications. This use-case is a perfect Cloud candidate as a testing phase is something which has a temporary nature and any extra investment in infrastructure for testing may become redundant after the testing has been completed.

**Database Integration:** Some of the PaaS Cloud providers offer services around integrating applications with back-end databases. This can range from doing the configuration and setup of the application to database connection to providing the actual integrators.

### 6.3.3 Software as as Service (SaaS)

SaaS appears to provide the most value it supports the “plug and go” concept where IT organizations source applications from public cloud service providers and make them available to their consumers. This is the scenario where IT organizations of SMEs literally become IT service brokers. The list of SaaS offerings is endless. So are the providers. Some SaaS providers are:



On the other hand, this scenario is the most disruptive for IT organizations to adopt unless the SME business is in start-up mode and has no existing software stack and data. Otherwise, IT organizations need to find a way of migrating existing data from the in-house applications onto the public SaaS environment. Many Cloud providers offer migration services to overcome this exact hurdle. To list a few common SaaS offerings:

**ERP Solutions:** SMEs typically haven't had an opportunity to adopt ERP solutions (as opposed to LEs) due to the massive cost of implementing them. This has changed thanks to the introduction of Cloud-based ERP solutions. SMEs now have a real chance to enjoy the benefits of integrated business management systems at low cost.

**CRM, BI, and Content Management Solutions:** CRM Cloud solutions have been available on the market for more than a decade. It is probably safe to say that the CRM SaaS offering was the spark that ignited the rest of the IT industry to produce Cloud-based solutions. Today, SMEs can subscribe to CRM, BI, and content management solutions that have been proven and in the market for quite some time.

**Messaging and Collaboration:** Often these Cloud offerings are adopted first, as migrating them to Cloud is less disruptive to the SME's business.

**IT Service Management (ITSM):** Small and medium IT organizations have an opportunity to completely rely on the ITSM software stack that is provided by the SaaS Cloud service provider. This is also interesting when IT organizations source from different providers. This integration level from a service management perspective is important as IT organizations need to successfully manage all the different aspects of their IT, from planning to building to running their entire IT stack.

#### **6.4 Impact of cloud services in an organization**

Cloud service increases efficiency of the business when deployed properly. Key points on the cloud services impact on an organization are -

- Productivity: More business with less IT
- Costing: Pay for only what you use
- Speed: Getting there more quickly
- Size: Adapting to business needs
- Quality: Improved Margin from Better Service
- Security: Organization's top priority
- Risk: Your data, your concern

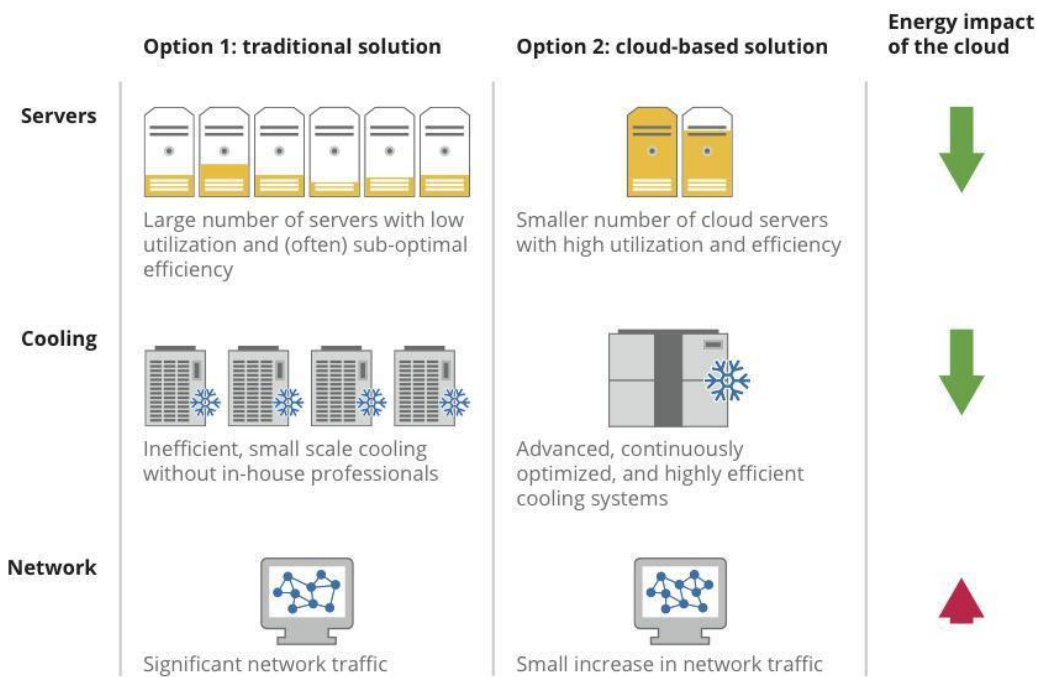
#### **6.5 Productivity: More business with less IT**

Productivity is valued in every workplace, but there are differing schools of thought as to how best to achieve it. Some businesses prefer a lot of structure and accountability, while others advocate a more flexible and permissive environment, giving employees more freedom and latitude. In any case, the measure of productivity is to get things done by proper utilization of resources, both human and IT.

##### **6.5.1 Resource Utilization**

Resources that are sized to handle peak loads are under-utilized at off-peak times. Also, in many enterprises, servers are dedicated to specific functions or departments, and can be massively under-utilized. It is commonplace for departments to request new servers for new projects, even though other departments have capacity that is going begging.

Cloud computing enables resources to be shared by different loads, and thus improves utilization. The sharing can be between enterprises, with public or community cloud, or within an enterprise, with private cloud. The rewards can be high. As an indication of what can be achieved, enterprises that adopt private cloud as a solution are reporting server consolidation ratios of as much as 12 to 1.



At the infrastructure level, resource sharing means supporting many platforms and applications on the same physical resources, using the virtualization techniques described under IaaS in the section on Service Models.

At the platform and application levels, providers can achieve resource sharing through multi-tenancy. For PaaS, this means applications from many clients running on the same operating system. For SaaS, it means clients sharing the same application instance. By doing this, using specially-designed application architectures, they can achieve better utilization of the underlying assets than if each client uses either a separate application instance on the same operating system or another virtualized instance sharing hardware.

### 6.5.2 Improved Communication

One of the most undeniable ways that the cloud improves productivity is through a more integrated and connected staff than ever before. The cloud allows for instant and comprehensive collaboration among team members. The cloud allows users to stay in sync in real time, with all members of a team having access to the most current and most recently updated versions.

Whether it's a project, document or file, everyone is always on the same page. Team members do not even have to share the same office space; using cloud technology, users can be scattered around the globe and still collaborate effectively. Also, staff members are no longer bound by regular office hours; they can put in time around the clock with full access to the same resources that available during regular business hours.

### 6.5.3 Flexible working environment

Cloud Computing solutions allows to work efficiently from home. With the Cloud and virtual desktops, employees can easily access everything they would otherwise have at the office. This helps increase employee satisfaction and, of course, business productivity.

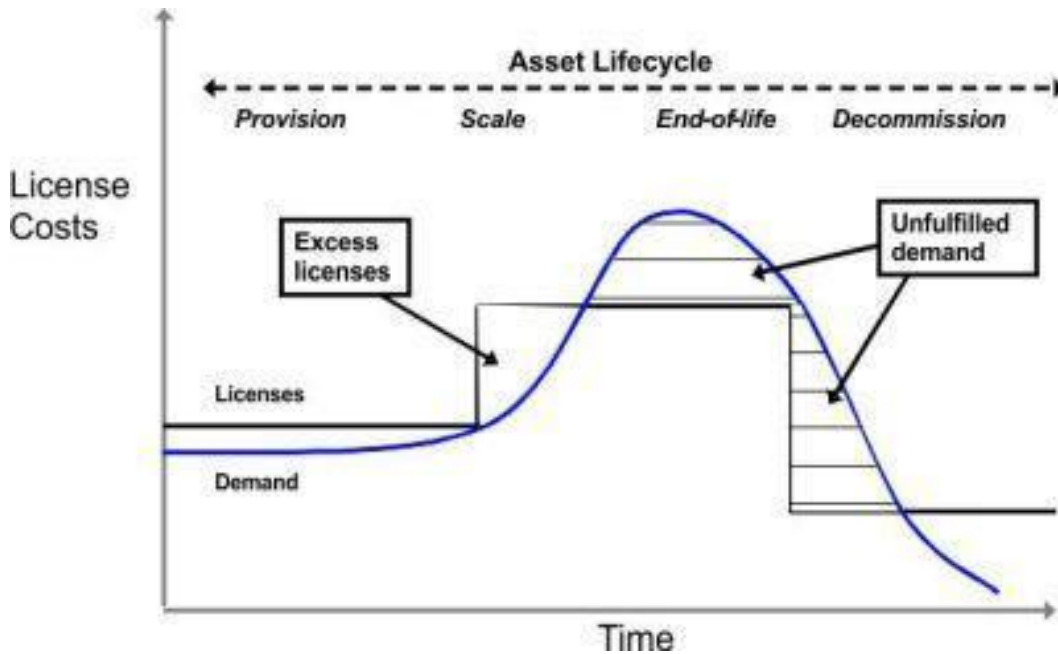
### 6.5.4 Costing: Pay for only what you use

#### 6.5.4.1 Usage-Based Pricing

Usage-based pricing translates the higher utilization achieved by providers into lower costs for consumers.

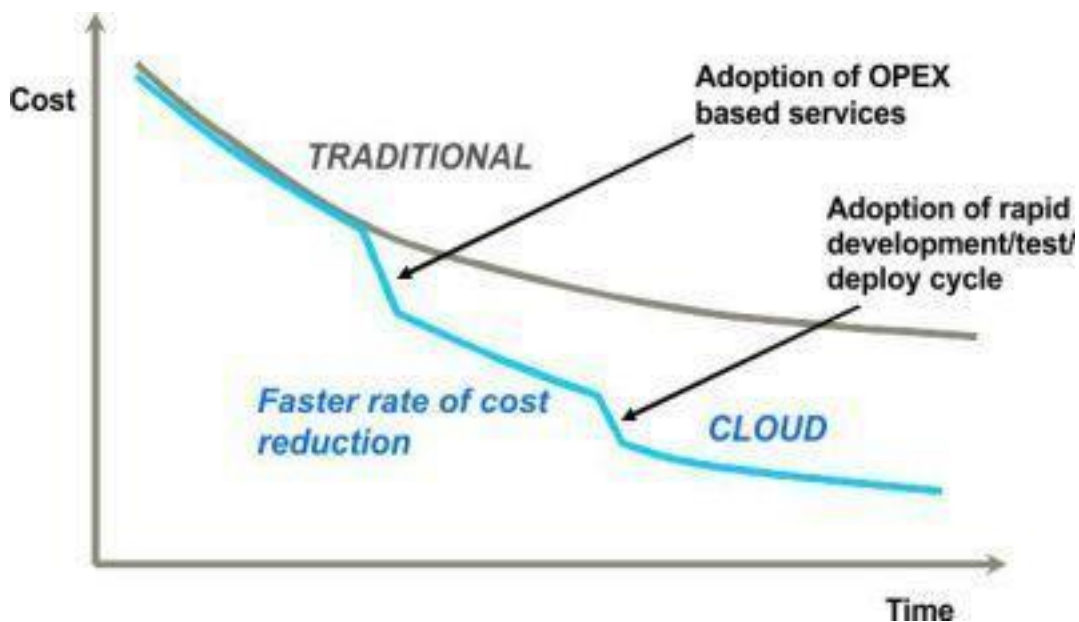
The benefit is not that they are using less resource; it is that they are paying for less resource. Their cost curve follows their resource utilization, resulting in a lower total than for a dedicated, under-used system.

Usage-based pricing for software licenses gives similar benefits to SaaS users. Traditional licensing associated with ownership, number of users, support, and maintenance costs and services lead to capacity-utilization gaps similar to those for hardware resources.



### 6.5.5 Lifetime Cost Models

Increased speed of execution has a positive impact on lifetime cost models. Typically, cost is reduced over the lifetime of a product or service as the depreciation cost of purchased assets decreases and as efficiencies are introduced. The speed of cost reduction can be much higher using cloud computing than traditional investment and divestment of IT assets, as illustrated in the figure below.



## Speed of Cost Reduction

This higher rate of cost reduction means that profitability increases more quickly, giving shorter pay-back times and increased ROI.

### 6.5.6 Specialization and Scale

In addition to the advantages of load sharing, cloud computing can result in lower IT costs because of skill specialization and economies of scale. A large cloud provider – or private cloud division within a corporation – can be much better at providing IT services than a small IT department. And it can amortize the cost of problem-solving over a larger user base: the problems experienced by one user can, once they have been solved, be proactively fixed for all the other users of the cloud service.

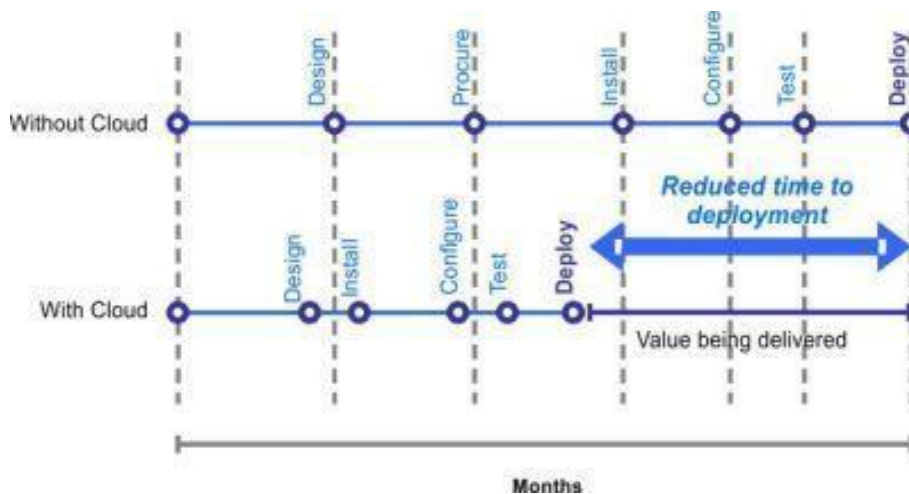
## 6.6 Speed: Getting there more quickly

The second major way in which cloud computing contributes to ROI is through improved speed of operation.

### 6.6.1 Time to Deployment

Cloud computing provides an increase in provisioning speed, which enables enterprises to acquire the resources they need faster. And, because resource configurations are visible, it speeds up the choice of multi-sourced resources, which can dramatically cut the time to deployment of new products and services. This applies to application and platform resources, as well as to physical infrastructure.

Elastic provisioning creates a new way for enterprises to scale their IT to enable business to expand. The compression of provisioning time from weeks to hours, for example, demonstrated by cloud computing providers is a means to rapid execution that is not just about saving time, but is also defining a new business operating model.



Organizations can review and develop business plans and then deploy infrastructure and services in a more rapid and proactive way. Customization and development, testing, and support can also be seen in a new light with the provision of IT services in a dynamic fashion targeted at business needs.

### 6.6.2 IT Asset Management

The use of IT has become an enduring feature in all organizations today. The investment in data, knowledge, and infrastructure assets and software now represent lifeblood operations for many businesses. But many enterprises have IT asset portfolios that do not reflect their real requirements. The issues of cost of ownership are often decoupled from choices made during selection of new IT, and the impact on the long-term running and maintenance of these IT services and subsequent business usage is not properly considered.

Technology design choices and purchasing are often done by strategic or tactical contractual purchasing based on project requirements, with little consideration for optimizing running and maintenance over the whole system lifecycle, but the cost of maintenance and modifications often represent a significant part of the asset lifecycle beyond initial provisioning.

A key aspect of moving to cloud computing is the ability to select hardware, software, and services from defined design configurations to run in production. It bridges the design-time and run-time divide and optimizes service performance. Patches and upgrades to the cloud resources are provided as part of the service.

Cloud computing facilitates optimization of the total asset portfolio. It can help an enterprise achieve the goal of a more cost-effective asset-management lifecycle process for the IT portfolio, to optimize both design and run-time performance.

## **6.7 Size: Adapting to business needs**

The combination of lower cost and faster delivery makes products and services more competitive, which generates more business and leads to a larger scale of operation. But what makes cloud computing really exciting is that its potential effect on business is not just incremental improvement but disruptive transformation, which it enables through new operating models.

### **6.7.1 Entering New Markets**

Cloud computing removes the need for additional infrastructure to test and enter markets (a key advantage particularly for small to medium-sized organizations). It enables businesses to pursue new and existing markets by rapid entry and exit of the products and services. It enables enterprises to “land and expand” in markets with an infrastructure and service capacity that can grow with the business. Existing and new markets can be attacked and entered through speculative and well-timed interventions to exploit and grow business performance.

### **6.7.2 High-Value Services**

Internet services such as Facebook, MySpace, Flickr, YouTube, and Twitter have become part of our everyday social interaction, and businesses are increasingly making use of IT-based collaboration services for communication, information exchange, and virtual meetings. In many cases, these are not provided by in-house IT; they are cloud services. Use of these services can transform business operation.

From a provider’s point of view, these are prime examples of simple, high-value services that are massively successful. But they do not have traditional price-based revenue models; they are often free to use. Advertising is one way in which such services can obtain revenue. Another is through offering additional “annuity” services – essential or useful add-ons that are paid for; just as many printer companies make money from the ink cartridges rather than from the printers themselves.

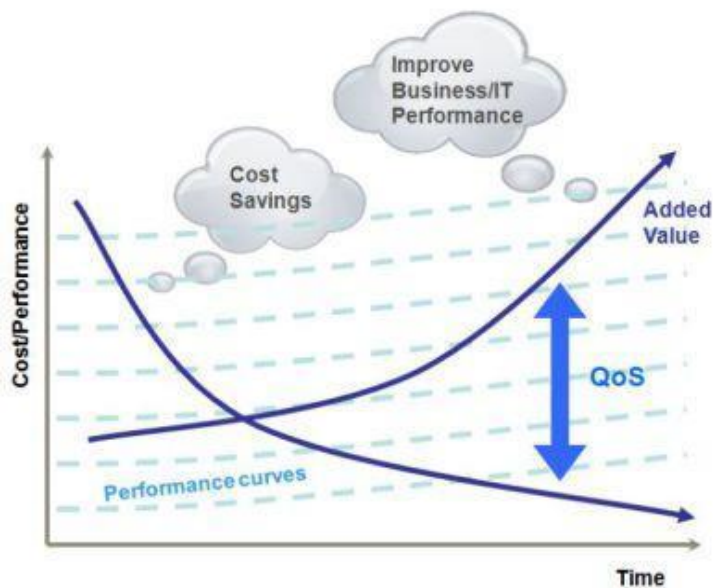
## **6.8 Quality: Improved margin from better service**

Increased scale and scope of operation delivers more revenue. Efficiencies resulting from better use of assets and faster operation deliver lower costs. The result is a bigger margin, and more profit.



### 6.8.1 Competitive Pressure

The positioning of cloud computing, while initially seen as a disruptive technology influence on both buyers and seller prospects, is now evolving into a trade-off between low-cost arbitrage and added value QoS.



In a market driven by cost, there can be intense competition between participants to make cost savings. This is often seen in a negative context, as the lower costs and margins are seen as a detriment to the participants overall. Massively scalable services from cloud computing providers have the effect of driving down costs and prices, as the dynamics of competition are shifted by the presence of potentially rapid cost reductions.

### 6.8.2 The Importance of Quality

The differentiator of cloud computing is not just the utility computing services, but includes all the higher-level services that enhance and build service value. This is part of a movement from IT-centric to business-centric services across a wider services continuum, with utility services for infrastructure at one end, and with business-centric software and business processes delivered as a service from the cloud at the other.



## **6.9 Security: Organization's top priority**

The security of a system depends on the security of all of its components and on the security of all links between them. Security should be considered at all stages of system and component design. The Open Group Security Forum and Jericho Forum are sources of such materials. Another excellent source is the Cloud Security Alliance (CSA), both for providing security assurance within cloud computing, and for use of cloud computing to help secure other forms of computing.

### **6.9.1 Privileged user access**

Sensitive data processed outside the enterprise brings with it an inherent level of risk, because outsourced services bypass physical, logical and personnel controls. IT shops exert over in-house programs. The organization needs to get as much information as possible about the people who manage its data.

Various information security concerns relating to the IT and other professionals associated with cloud services are typically handled through proper recruiting and monitoring process

### **6.9.2 Physical security**

Cloud service providers physically secure the IT hardware (servers, routers, cables etc.) against unauthorized access, interference, theft etc. to minimize the possibility of disruption. This is normally achieved by serving cloud applications from 'world-class' (i.e., professionally specified, designed, constructed, managed, monitored and maintained) data centers.

### **6.9.3 Regulatory compliance**

Customers are ultimately responsible for the security and integrity of their own data, even when it is held by a service provider. Traditional service providers are subjected to external audits and security certifications.

### **6.9.4 Data location**

Cloud computing service providers have data centers in different parts of the world. Before committing to a service an organization should investigate where the data will be stored and which privacy and security laws will apply to the data.

### **6.9.5 Data segregation**

Data in the cloud is typically stored in a shared environment alongside data from other customers. At the infrastructure level, multi-tenancy is generally achieved by virtualization; at the platform level it is generally achieved within the context of a multi-user operating system; and at the application level it is achieved by design of the application concerned. Generally, each tenant is unaware of the others, and uses the resources as though they were dedicated to him.

## **6.10 Risks: Your data, your concern**

Every large cloud provider is a huge user of virtualization. However, it holds every risk posed by physical machines, plus its own unique threats, including exploits that target the virtual server hosts and the guests.

### **6.10.1 Data loss**

The prospect of seeing valuable data disappear into the ether without a trace is applicable to all platforms. Any system is prone to hardware or software failures, and disasters. A malicious hacker might delete a target's data out of spite -- but then, valuable data might get lost due to a careless cloud service provider or a natural disaster, such as a fire, flood, or earthquake. Compounding the challenge, encrypting data to ward off theft can backfire if the encryption key is lost.

### 6.10.2 Availability

Cloud providers help ensure that customers can rely on access to their data and applications, at least in part. But failures at any point, not just within the cloud service providers' domains, may disrupt the communication chains between users and applications. Power outages, network or hardware failure, all these may cause data to be temporarily unavailable.

### 6.10.3 Long-term viability

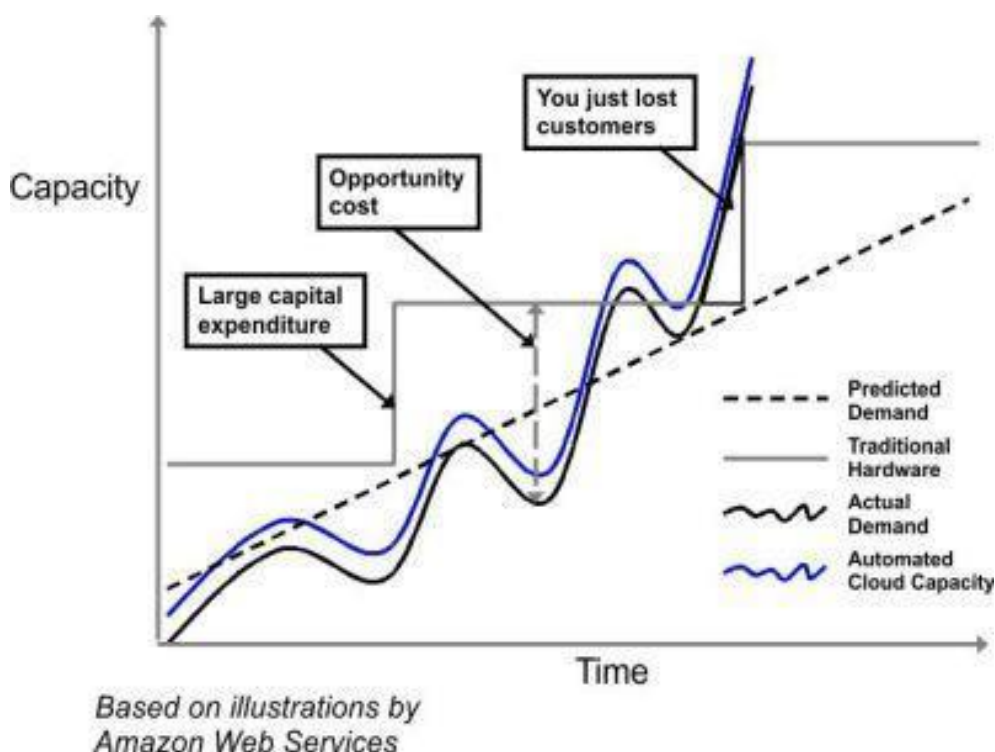
Ideally, a cloud computing provider will never go broke or get acquired and swallowed up by a larger company. But the organization must be sure that data will remain available even after such an event. "Ask potential providers how you would get your data back and if it would be in a format that you could import into a replacement application," Gartner says.

## 6.11 Case studies

### 6.11.1 How cloud services can affect product pricing

By increasing the number of customers, organizations can make the same profit with less margin and with the help of IT the organization can support the large customer base with the same manpower.

Again, by maximising utilization of resources, organizations can reduce the cost of their products. The figure below illustrates the proper resource utilization in cloud computing, it is based on the capacity/utilization curve used by Amazon Web Services to illustrate the central idea of utility-based services enabled through on-demand provisioning to meet actual usage.



Consumers should of course monitor cost, and should look at it in conjunction with workload. If payment is directly related to resource use – “pay-by-the-drink” – then costs should track workload. If there is an element of time-based payment – “pay-as-you-go” – then this may not be the case, and there may be scope to lower costs by tuning resource usage.

Private cloud users often look at the server consolidation ratio – the ratio of servers in their private cloud to those that were used before the cloud was introduced.

Generally cloud services are designed for quickly adapting to changes. When an organization undergoes some changes, the speed of adapting the changes directly affects the profit or loss during that time. In the long run if these losses are minimized the service cost can be reduced. Speed of provisioning contributes to speed of cost reduction, which can provide a long-term indicator of success.

Workload is probably the best IT measures of this. It should be looked at in conjunction with revenue and cost. Cloud providers – including providers of private cloud – will also monitor their investment in IT resources.

## **6.12 Cloud services in software development - Enhancing agile software development**

Cloud service has greatly improved software development process. Organizations leveraging on cloud computing vastly speed up release cycles. Company's cloud infrastructure helps it maintain a single, unified code base that geographically distributed development teams can use. Those teams are successfully combining agile development and continuous integration/delivery with cloud computing.

### **6.12.1 Cloud Computing Provides an Unlimited Number of Testing and Staging Servers**

When agile development is used without virtualization or clouds, development teams are limited to one physical server per development, staging and production server need. However, when virtual machines or cloud instances are used, development teams have practically an unlimited number of servers available to them. They do not need to wait for physical servers to become free to begin or continue their work.

### **6.12.2 It Turns Agile Development Into a Truly Parallel Activity**

An organization may use agile development but still experience delays in provisioning server instances and in installing necessary underlying platforms such as database software. Just as the Commonwealth Bank reduced the provisioning of an Oracle database from three months to two minutes, agile development teams can provision the servers they need quickly themselves, rather than wait for IT operations to do it for them.

### **6.12.3 It Encourages Innovation and Experimentation**

Being able to spawn as many instances as needed enables agile development groups to innovate. If a feature or a story looks interesting, a team can spawn a development instance quickly to code it and test it out. There's no need to wait for the next build or release, as is the case when a limited number of physical servers are available. When adding cloud computing to agile development, builds are faster and less painful, which encourages experimentation.

### **6.12.4 It Enhances Continuous Integration and Delivery**

As stated, cloud instances and virtualization greatly enhance continuous integration and delivery. Builds and automated tests take time. Agile development groups may need to subsequently fix the code for tests that fail during the automated testing—and they need to do this again and again until the build passes all the tests.

### **6.12.5 It Makes More Development Platforms and External Services Available**

Agile development groups may need to use a variety of project management, issue management, and, if continuous integration is used, automated testing environments. A number of these services are available as Software as a Service (SaaS) offerings in the cloud.

## **6.13 Use of cloud services in different industries**

### **6.13.1 Small and medium industries**

Cloud services are especially intriguing for SMEs. For a start up company or a company who want to explore new horizons cloud provides a very cost effective way to

1. No need to manage an IT team to maintain servers and to do things such as install and update software, install and manage email servers and/or file servers, run backups. The beauty of cloud computing is that all of the business of maintaining the service or application is the responsibility of the cloud vendor.
2. Instead of capital expenditure on purchasing hardware and software the organization rents both hardware and software, which is great if the company is low on funds or uncertain about business outcomes.
3. You may be able to consolidate your separate application needs into one multi-application cloud computing service. For instance, Google Apps for Business includes email, a calendar scheduling application, Google Docs for creating documents, presentations and forms and using online file storage and Google Sites for creating websites, all for only \$5/month for each person on your account. Now think about the price of, let's say, Microsoft Office (including Microsoft Outlook for email) – and note that Office doesn't include a website application.
4. Organization can cut back on system hardware. File storage, data backup and software programs all take up a lot of space on servers/computers. With cloud computing, you use someone else's servers to store all this data instead, freeing up your in-house computer equipment for other purposes or even letting you get rid of some of it.

### **6.14 Large industries**

Larger corporations have struggled to trust the cloud computing model, especially when it comes to migrating mission-critical applications, however this attitude is beginning to change.

Large companies with hundreds of departments all placing heavy demands on IT are good candidates for a private cloud. The self-service, automation and chargeback capabilities of a private cloud make a huge difference, but only at scale. This is because the economic benefits of moving to a cloud model are achievable only when all or most IT services run on a cloud platform and where the use of compute resources is high. Simply put: The bigger the cloud, the more cost-effective it is.

According to the Microsoft white paper "The Economics of the Cloud," for small and medium-sized organizations with fewer than 100 servers, private clouds are prohibitively expensive compared with public cloud services. The only way for these small organizations or departments to share the benefits of at-scale cloud computing is by moving to a public cloud model, the white paper argues. For large organizations with an installed base of approximately 1,000 servers, however, private clouds are feasible. But they still come at a significant cost of about 10 times the price of a public cloud for the same unit of service given the combined effect of scale, demand diversification and multitenancy. And the questions of scale and cost are just the beginning; management software immaturity is also a major hurdle.

From the Forrester Inc. survey, which surveyed 2,000 IT decision makers from North America and Europe, it's clear that there's a long way to go before most companies are ready to build a private cloud. One of the biggest barriers is cultural. While historically IT has largely enabled operational efficiency, today's IT workers need to focus more on bringing strategic gains. Some may resist the changes inherent in a cloud model in the name of organizational security and control. But this resistance often stems from an old-school view of IT resources as assets to be doled out by guardians of the "fortress."

## **6.15 Common Pitfalls in using Cloud Services**

### **6.15.1 Not fully understanding cloud security**

The current lack of standards and lack of enterprise experience with cloud make private corporate clouds ripe for abuse. The truth about security and the cloud is quite simple. With the proper security architecture, the public cloud can be more secured than most on-premises data centers since most of the cloud service vendors utilize the highest security certifications. However, It is imperative for cloud service provider and user to use an information security management system (ISMS) to effectively manage their information assets. ISMS is basically consist of sets of policies put place by an organization to define, construct, develop and maintain security of their computer based on hardware and software resources. These policies dictate the way in which computer resources could use. Before choosing any vendor for cloud service the consumer should be aware of ISMS certifications and their scopes, and whether the vendor complies with the security scheme.

### **6.15.2 Not understanding the cost model and usage**

Since it shifts IT from capital expenses to monthly subscription payments, cloud may appear to be far more inexpensive up front. However, primary IT costs tend to remain constant. The most expensive part of cloud computing usually has nothing to do with the cloud at all. Often, companies underestimate the effort it takes to build software in the cloud

### **6.15.3 Disasters and lack of recovery plan**

Cloud service providers offer to save client's files and documents, thus it is not reassuring to use since the data saved might be lost. As an example is experienced by T-Mobile's Sidekick smart phone services where in October 2009; the data of the clients have been lost and were not able to be retrieved. Therefore, it is quite risky to use cloud computing as there would be loss of control over the saved data.

### **6.15.4 Not preparing for outages**

Cloud services are subject to outages that are beyond a business' control. In 2010, Intuit's site went down for two days, leaving its SMB customers unable to access their data in Intuit's online offerings including Quicken, QuickBooks and TurboTax. In such downtime companies should be prepared to manage the business manually.

### **6.15.5 The fine print of Terms and Conditions**

Key concerns about service level agreements are tied to the primary areas of risk. Data ownership policies, service availability, monitoring and response, and service baselines, such as regulatory compliance or security due diligence for vulnerability and configuration management should be assessed properly before contracting with the provider.

### **6.15.6 Underestimating the impacts of organizational change**

Change goes way beyond IT, and it always boils down to people management. People often resist change, or changes to business processes. Smaller, lower-risk initiatives as the early candidates for cloud computing projects can be very effective.

### **6.15.7 Migrating applications to the cloud solely to drive down costs**

Not all applications are good candidates to move to the cloud in their current architecture. Most legacy architectures were never built in a manner where the system automatically scales as the number of transactions increases.

### **6.15.8 Having inflated expectations of suddenly becoming a digital enterprise**

Many people look at the big web stars such as Instagram or Facebook “built on the cloud” and assume they can take that same route. But in reality many organizations have a very complex enterprise consisting of numerous vendor and proprietary solutions ranging from mainframe technologies, to midsize computers, n-tier architectures, and every other architectural pattern that was popular at one time or another. Starting with a blank slate or getting an initiative from the CEO to re-platform the entire product line with a new cloud-based architecture is not the norm for most companies.

### **6.15.9 Selecting a favorite vendor, not an appropriate vendor**

Cloud often represents a different set of business circumstances than the IT services everyone is familiar with over the years. Understanding the differences between three cloud service models: Software as a Service, Platform as a Service, and Infrastructure as a Service and knowing what business cases are best suited for each service model is key to selecting a vendor.

### **6.15.10 Not bringing in enough of the right skills**

Deploying, monitoring, and maintaining software in the cloud can be drastically different from how an organization currently handles those tasks. Broad knowledge of networking, security, distributed computing, SOA web architectures, and much more are required to properly manage services.

### **6.15.11 Misunderstanding customer requirements**

Sometimes IT people neglect the business side of the equation and build the cloud solution that is best for IT. Customer’s need should be the first priority. While the Cloud is here to stay, it will not replace traditional hosting or on-premise deployments, but rather complement them. There will always be situations where security requirements, flexibility, performance or control will preclude the cloud. In those cases, a hybrid solution involving both cloud and either traditionally hosted or on-premise servers may make sense.

## **6.16 Final words**

Cloud computing can create significant return on investment, affording energy, licensing, and administrative cost savings for many applications enterprises run today, as well as providing the flexibility to invent new applications. By leveraging economies of scale, it frees up capital and personnel to innovate new ideas quickly, with limited financial penalties. Cloud computing, or anything in computing, is not perfect. Data centers, whether they are public or private, go down. Outages happen in-house as well as to the industry’s leading cloud-hosting providers.

Though it will keep evolving, the cloud is here to stay, not only because the model is more efficient and more cost effective than the traditional IT infrastructure, but because it promotes the promise of specialization—a value that gives companies an edge and consumers a better product.

But moving to the cloud is not a decision to be made lightly, or without considering all aspects of the organization. As with any worthwhile investment, cloud computing has an organizational impact that must be carefully considered, not only for IT, but for the entire business. Moving to the cloud is a transformational investment, in every sense of the word—but it’s a move that many of today’s organizations find compelling.

All surveys forecast significant growth in cloud computing. However, pure cloud implementations are the exception and not the rule. And this is to be expected. Because it would be very difficult, if at all possible, to move everything wholesale to the cloud because of the complexity of today’s environments. The hybrid cloud—a mix of on and off premises—offers the best of both worlds: a combination of strengths allowing organizations to achieve the performance of on-premises solutions yet also the management convenience of the cloud business model.

## 6.17 References

1. <http://www.forbes.com/sites/tjmccue/2014/01/29/cloud-computing-united-states-businesses-will-spend-13-billion-on-it/>
2. <http://www.cio.com/article/2393022/enterprise-architecture/6-ways-the-cloud-enhances-agile-software-development.html>
3. <http://www.progress.com/~media/Progress/Documents/News%20and%20Events/Exchange2013/Track%208%20-%20Expand%20Market%20Grow%20PFuller%20v2.pdf>
4. <http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=PM&subtype=XB&htmlfid=GBE03580USEN>
5. [http://www.ciozone.com/index.php/Cloud-Computing/IBM-Cloud-Computing-to-Contribute-\\$7-Billion-in-Revenue-By-2015.html](http://www.ciozone.com/index.php/Cloud-Computing/IBM-Cloud-Computing-to-Contribute-$7-Billion-in-Revenue-By-2015.html)
6. <http://siliconangle.com/blog/2014/01/27/20-cloud-computing-statistics-tc0114/>

**CHAPTER 7**  
**CLOUD INFRASTRUCTURE: OPENSTACK**



## 7. CLOUD INFRASTRUCTURE: OPENSTACK

### 7.1 Introduction to OpenStack

OpenStack is an open source platform that lets you build an Infrastructure as a Service (IaaS) cloud that runs on commodity hardware. OpenStack believes in open source, open design, open development, all in an open community that encourages participation by anyone. The long-term vision for OpenStack is to produce a ubiquitous open source cloud computing platform that meets the needs of public and private cloud providers regardless of size. OpenStack services control large pools of compute, storage, and networking resources throughout a data center.

The technology behind OpenStack consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. Each service provides an open API so that all of these resources can be managed through a dashboard that gives administrators control while empowering users to provision resources through a web interface, a command-line client, or software development kits that support the API. Many OpenStack APIs are extensible, meaning you can keep compatibility with a core set of calls while providing access to more resources and innovating through API extensions. The OpenStack project is a global collaboration of developers and cloud computing technologists. The project produces an open standard cloud computing platform for both public and private clouds. By focusing on ease of implementation, massive scalability, a variety of rich features, and tremendous extensibility, the project aims to deliver a practical and reliable cloud solution for all types of organizations.

#### 7.1.1 Getting Started with OpenStack

As an open source project, one of the unique aspects of OpenStack is that it has many different levels at which you can begin to engage with it—you don't have to do everything yourself.

#### 7.1.2 Using OpenStack

You could ask, "Do I even need to build a cloud?" If you want to start using a computer or storage service by just swiping your credit card, you can go to eNovance, HP, Rackspace, or other organizations to start using their public OpenStack clouds. Using their OpenStack cloud resources is similar to accessing the publicly available Amazon Web Services Elastic Compute Cloud (EC2) or Simple Storage Solution (S3).

#### 7.1.3 Plug and Play OpenStack

However, the enticing part of OpenStack might be to build your own private cloud, and there are several ways to accomplish this goal. Perhaps the simplest of all is an appliance-style solution. You purchase an appliance, unpack it, plug in the power and the network, and watch it transform into an OpenStack cloud with minimal additional configuration. Few, if any other open source cloud products have such turnkey options. If a turnkey solution is interesting to you, take a look at Nebula One.

However, hardware choice is important for many applications, so if that applies to you, consider that there are several software distributions available that you can run on servers, storage, and network products of your choosing. Canonical (where OpenStack replaced Eucalyptus as the default cloud option in 2011), Red Hat, and SUSE offer enterprise OpenStack solutions and support. You may also want to take a look at some of the specialized distributions, such as those from Rackspace, Piston, Swift Stack, or Cloud scaling.

Alternatively, if you want someone to help guide you through the decisions about the underlying hardware or your applications, perhaps adding in a few features or integrating components along the way, consider contacting one of the system integrators with OpenStack experience, such as Mirantis or Metacloud.

If your preference is to build your own OpenStack expertise internally, a good way to kick-start that might be to attend or arrange a training session. The OpenStack Foundation recently launched a Training Market place where you can look for nearby events. Also, the OpenStack community is working to produce open source training materials.

## 7.2 Architecture

Designing an OpenStack cloud is a great achievement. It requires a robust understanding of the requirements and needs of the cloud's users to determine the best possible configuration to meet them. OpenStack provides a great deal of flexibility to achieve your needs, and this part of the book aims to shine light on many of the decisions you need to make during the process.

To design, deploy, and configure OpenStack, administrators must understand the logical architecture. A diagram can help you envision all the integrated services within OpenStack and how they interact with each other.

OpenStack modules are one of the following types:

|                              |                                                                                               |
|------------------------------|-----------------------------------------------------------------------------------------------|
| Daemon                       | Runs as a background process. On Linux platforms, a daemon is usually installed as a service. |
| Script                       | Installs a virtual environment and runs tests.                                                |
| Command-line interface (CLI) | Enables users to submit API calls to OpenStack services through commands.                     |

As shown, end users can interact through the dashboard, CLIs, and APIs. All services authenticate through a common Identity Service, and individual services interact with each other through public APIs, except where privileged administrator commands are necessary. Fig. 7.1, “OpenStack Logical Architecture ()” shows the most common, but not the only logical architecture for an OpenStack cloud.

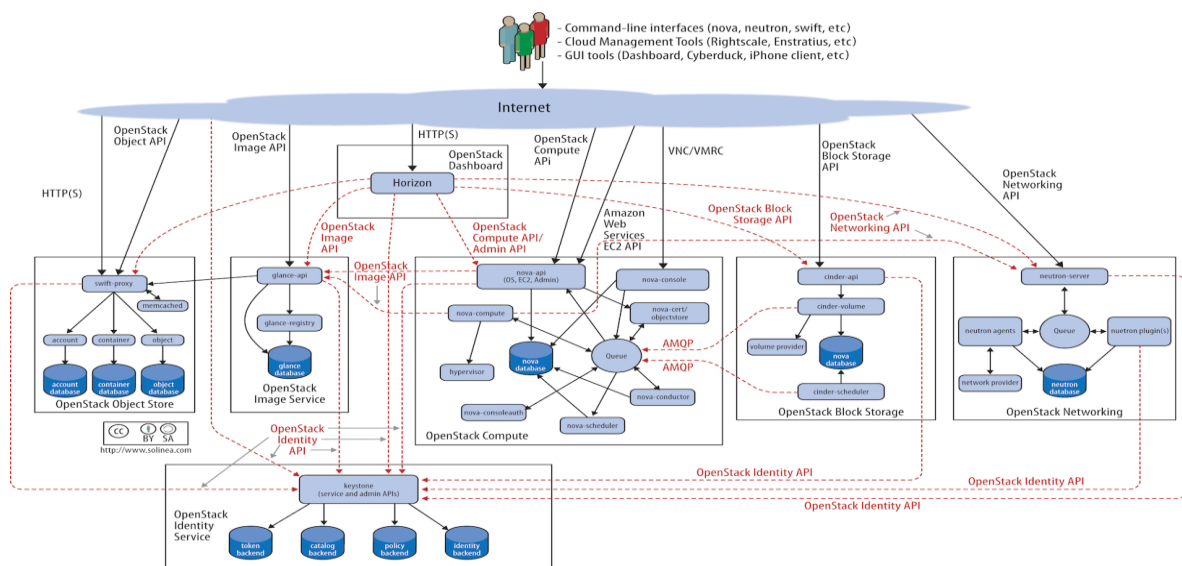


Fig. 7.1. OpenStack Logical Architecture

## 7.2.1 Example Architectures

To understand the possibilities OpenStack offers, it's best to start with basic architectures that are tried-and-true and have been tested in production environments. We offer two such examples with basic pivots on the base operating system (Ubuntu and Red Hat Enterprise Linux) and the net-working architectures. There are other differences between these two examples, but you should find the considerations made for the choices in each as well as a rationale for why it worked well in a given environment.

Because OpenStack is highly configurable, with many different back ends and network configuration options, it is difficult to write documentation that covers all possible OpenStack deployments. Therefore, this guide de-fines example architectures to simplify the task of documenting, as well as to provide the scope for this guide. Both of the offered architecture examples are currently running in production and serving users.

### 7.2.1.1 Example Architecture—Legacy Networking (nova)

This particular example architecture has been upgraded from Grizzly to Havana and tested in production environments where many public IP addresses are available for assignment to multiple instances. You can find a second example architecture that uses OpenStack Networking (neutron) after this section. Each example offers high availability, meaning that if a particular node goes down, another node with the same configuration can take over the tasks so that service continues to be available.

#### Overview

The simplest architecture you can build upon for Compute has a single cloud controller and multiple compute nodes. The simplest architecture for Object Storage has five nodes: one for identifying users and proxying re-quests to the API, then four for storage itself to provide enough replication for eventual consistency. This example architecture does not dictate a particular number of nodes, but shows the thinking and considerations that went into choosing this architecture including the features offered.

#### Components

| Component                               | Details                                                                                                     |
|-----------------------------------------|-------------------------------------------------------------------------------------------------------------|
| OpenStack release                       | Havana                                                                                                      |
| Host operating system                   | Ubuntu 12.04 LTS or Red Hat Enterprise Linux 6.5, including derivatives such as CentOS and Scientific Linux |
| OpenStack package repository            | Ubuntu Cloud Archive or RDO*                                                                                |
| Hypervisor                              | KVM                                                                                                         |
| Database                                | MySQL*                                                                                                      |
| Message queue                           | RabbitMQ for Ubuntu; Qpid for Red Hat Enterprise Linux and derivatives                                      |
| Networking service                      | nova-network                                                                                                |
| Network manager                         | FlatDHCP                                                                                                    |
| Single nova-network or multi-host?      | multi-host*                                                                                                 |
| Image Service (glance) backend          | File                                                                                                        |
| Identity Service (keystone) driver      | SQL                                                                                                         |
| Block Storage Service (cinder) back-end | LVM/iSCSI                                                                                                   |
| Live Migration backend                  | Shared storage using NFS*                                                                                   |
| Object storage                          | OpenStack Object Storage (swift)                                                                            |

An asterisk (\*) indicates when the example architecture deviates from the settings of a default installation. We'll offer explanations for those deviations next.

### Note

The following features of OpenStack are supported by the ex-ample architecture documented in this guide, but are optional:

- **Dashboard:** You probably want to offer a dashboard, but your users may be more interested in API access only.
- **Block storage:** You don't have to offer users block storage if their use case only needs ephemeral storage on compute nodes, for example.
- **Floating IP address:** Floating IP addresses are public IP addresses that you allocate from a predefined pool to assign to virtual machines at launch. Floating IP address ensure that the public IP address is available whenever an instance is booted. Not every organization can offer thousands of public floating IP addresses for thousands of instances, so this feature is considered optional.
- **Live migration:** If you need to move running virtual machine instances from one host to another with little or no service interruption, you would enable live migration, but it is considered optional.
- **Object storage:** You may choose to store machine images on a file system rather than in object storage if you do not have the extra hardware for the required replication and redundancy that OpenStack Object Storage offers.

### Rationale

This example architecture has been selected based on the current default feature set of OpenStack Havana, with an emphasis on stability. We believe that many clouds that currently run OpenStack in production have made similar choices.

You must first choose the operating system that runs on all of the physical nodes. While OpenStack is supported on several distributions of Linux, we used Ubuntu 12.04 LTS (Long Term Support), which is used by the majority of the development community, has feature completeness compared with other distributions and has clear future support plans.

We recommend that you do not use the default Ubuntu OpenStack install packages and instead use the Ubuntu Cloud Archive. The Cloud Archive is a package repository supported by Canonical that allows you to upgrade to future OpenStack releases while remaining on Ubuntu 12.04.

KVM as a hypervisor complements the choice of Ubuntu—being a matched pair in terms of support, and also because of the significant degree of attention it garners from the OpenStack development community (including the authors, who mostly use KVM). It is also feature complete, free from licensing charges and restrictions.

MySQL follows a similar trend. Despite its recent change of ownership, this database is the most tested for use with OpenStack and is heavily documented. We deviate from the default database, SQLite, because SQLite is not an appropriate database for production usage.

The choice of RabbitMQ over other AMQP compatible options that are gaining support in OpenStack, such as ZeroMQ and Qpid, is due to its ease of use and significant testing in production. It also is the only option that supports features such as Compute cells. We recommend clustering with RabbitMQ, as it is an integral component of the system and fairly simple to implement due to its inbuilt nature.

As discussed in previous chapters, there are several options for networking in OpenStack Compute. We recommend Flat DHCP and to use Multi-Host networking mode for high availability, running one nova-network daemon per OpenStack compute host. This provides a robust mechanism for ensuring network interruptions are isolated to individual compute hosts, and allows for the direct use of hardware network gateways.

Live Migration is supported by way of shared storage, with NFS as the distributed file system.

Acknowledging that many small-scale deployments see running Object Storage just for the storage of virtual machine images as too costly, we opted for the file backend in the OpenStack Image Service (Glance). If your cloud will include Object Storage, you can easily add it as a backend.

We chose the SQL backend for Identity Service (keystone) over others, such as LDAP. This backend is simple to install and is robust. The authors acknowledge that many installations want to bind with existing directory services and caution careful understanding of the array of options available.

Block Storage (cinder) is installed natively on external storage nodes and uses the LVM/iSCSI plugin. Most Block Storage Service plug-ins are tied to particular vendor products and implementations limiting their use to consumers of those hardware platforms, but LVM/iSCSI is robust and stable on commodity hardware.

While the cloud can be run without the OpenStack Dashboard, we consider it to be indispensable, not just for user interaction with the cloud, but also as a tool for operators. Additionally, the dashboard's use of Django makes it a flexible framework for extension.

#### **Why not use the OpenStack Network Service (neutron)?**

This example architecture does not use the OpenStack Network Service (neutron), because it does not yet support multi-host networking and our organizations (university, government) have access to a large range of publicly-accessible IPv4 addresses.

#### **Why use multi-host networking?**

In a default OpenStack deployment, there is a single nova-network service that runs within the cloud (usually on the cloud controller) that provides services such as network address translation (NAT), DHCP, and DNS to the guest instances. If the single node that runs the nova-network service goes down, you cannot access your instances, and the instances cannot access the Internet. The single node that runs the nova-network service can become a bottleneck if excessive network traffic comes in and goes out of the cloud.

#### **Detailed Description**

The reference architecture consists of multiple compute nodes, a cloud controller, an external NFS storage server for instance storage, and an OpenStack Block Storage server for volume storage. A network time service (Network Time Protocol, or NTP) synchronizes time on all the nodes. Flat DHCP Manager in multi-host mode is used for the networking. A logical diagram for this example architecture shows which services are running on each node:

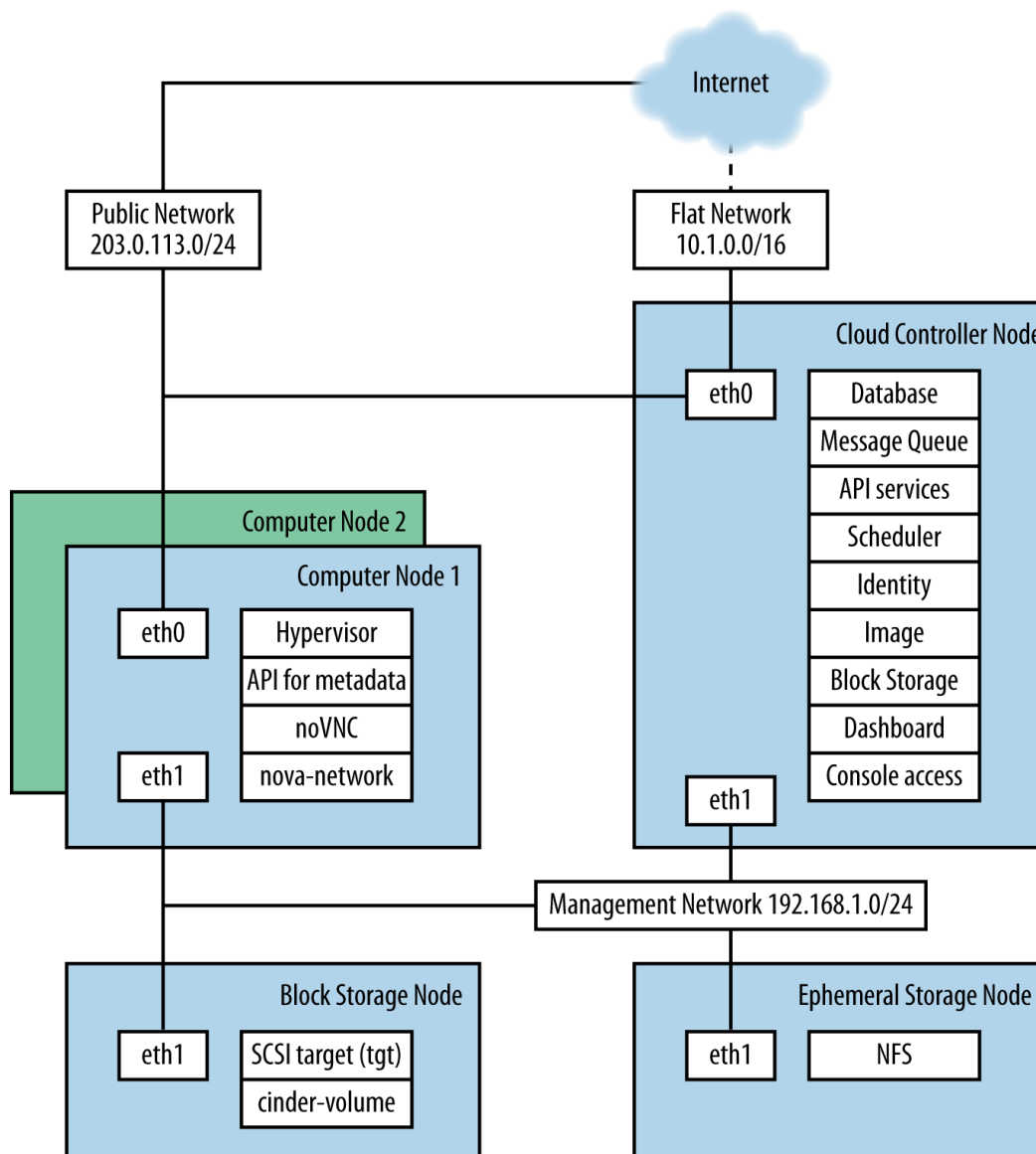


Fig. 7.2

The cloud controller runs the dashboard, the API services, the database (MySQL), a message queue server (RabbitMQ), the scheduler for choosing compute resources (nova-scheduler), Identity services (keystone, nova-console auth), Image services (glance-api, glance-registry), services for console access of guests, and Block Storage services, including the scheduler for storage resources (cinder-api and cinder-scheduler).

Compute nodes are where the computing resources are held, and in our example architecture, they run the hypervisor (KVM), libvirt (the driver for the hypervisor, which enables live migration from node to node), nova-compute, nova-api-metadata (generally only used when running in multi-host mode, it retrieves instance-specific metadata), nova-vncproxy, and nova-network.

The network consists of two switches, one for the management or private traffic, and one that covers public access, including floating IPs. To support this, the cloud controller and the compute nodes have two network cards. The OpenStack Block Storage and NFS storage servers only need to access the private network and therefore only need one network card, but multiple cards run in a bonded configuration are recommended if possible. Floating IP access is direct to the Internet, whereas Flat IP access goes through a NAT. To envision the network traffic, use this diagram:

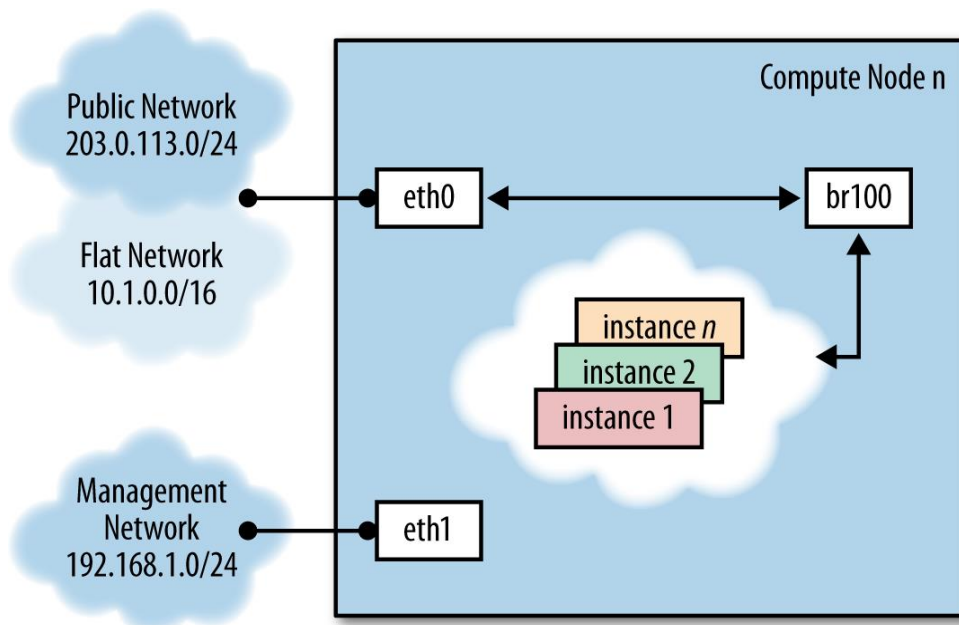


Fig. 7.3

### Optional Extensions

You can extend this reference architecture as follows:

- Add additional cloud controllers
- Add an OpenStack Storage service.
- Add additional OpenStack Block Storage hosts.

#### 7.2.1.2 Example Architecture—OpenStack Networking

This chapter provides an example architecture using OpenStack Networking, also known as the Neutron project, in a highly available environment.

### Overview

A highly-available environment can be put into place if you require an environment that can scale horizontally, or want your cloud to continue to be operational in case of node failure. This example architecture has been written based on the current default feature set of OpenStack Havana, with an emphasis on high availability.

### Components

| Component                               | Details                             |
|-----------------------------------------|-------------------------------------|
| OpenStack release                       | Havana                              |
| Host operating system                   | Red Hat Enterprise Linux 6.5        |
| OpenStack package repository            | Red Hat Distributed OpenStack (RDO) |
| Hypervisor                              | KVM                                 |
| Database                                | MySQL                               |
| Message queue                           | Qpid                                |
| Networking service                      | OpenStack Networking                |
| Tenant Network Separation               | VLAN                                |
| Image Service (glance) backend          | GlusterFS                           |
| Identity Service (keystone) driver      | SQL                                 |
| Block Storage Service (cinder) back-end | GlusterFS                           |

### Rationale

This example architecture has been selected based on the current default feature set of OpenStack Havana, with an emphasis on high availability. This architecture is currently being deployed in an

internal Red Hat Open-Stack cloud and used to run hosted and shared services, which by their nature must be highly available.

This architecture's components have been selected for the following reasons:

|                                 |                                                                                                                                                                                                                                                                                                                            |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Red Hat Enterprise Linux</b> | You must choose an operating system that can run on all of the physical nodes. This example architecture is based on Red Hat Enterprise Linux, which offers reliability, long-term support, certified testing, and is hardened. Enterprise customers, now moving into OpenStack usage, typically require these advantages. |
| <b>RDO</b>                      | The Red Hat Distributed OpenStack package offers an easy way to download the most current OpenStack release that is built for the Red Hat Enterprise Linux platform.                                                                                                                                                       |
| <b>KVM</b>                      | KVM is the supported hypervisor of choice for Red Hat Enterprise Linux (and included in distribution). It is feature complete and free from licensing charges and restrictions.                                                                                                                                            |
| <b>MySQL</b>                    | MySQL is used as the database back-end for all databases in the OpenStack environment. MySQL is the supported database of choice for Red Hat Enterprise Linux (and included in distribution); the database is open source, scalable, and handles memory well.                                                              |
| <b>Qpid</b>                     | Apache Qpid offers 100 percent compatibility with the Advanced Message Queuing Protocol Standard, and its broker is available for both C++ and Java.                                                                                                                                                                       |
| <b>OpenStack Networking</b>     | OpenStack Networking offers sophisticated networking functionality, including Layer 2 (L2) network segregation and provider networks.                                                                                                                                                                                      |
| <b>VLAN</b>                     | Using a virtual local area network offers broadcast control, security, and physical layer transparency. If needed, use VXLAN to extend your address space.                                                                                                                                                                 |
| <b>GlusterFS</b>                | GlusterFS offers scalable storage. As your environment grows, you can continue to add more storage nodes (instead of being restricted, for example, by an expensive storage array).                                                                                                                                        |

## Detailed Description

### Node types

This section gives you a breakdown of the different nodes that make up the OpenStack environment. A node is a physical machine that is provisioned with an operating system, and running a defined software stack on top of it. Table 7.1, “Node types” provides node descriptions and specifications.

| Type              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Example hardware                                                                                                                                                                           |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Controller</b> | <p>Controller nodes are responsible for running the management software services needed for the OpenStack environment to function. These nodes:</p> <ul style="list-style-type: none"> <li>• Provide the front door that people access as well as the API services that all other components in the environment talk to.</li> <li>• Run a number of services in a highly available fashion, utilizing Pacemaker and HAProxy to provide a virtual IP and load-balancing functions so all controller nodes are being used.</li> <li>• Supply highly available "infrastructure" services, such as MySQL and Qpid that underpin all the services.</li> <li>• Provide what is known as "persistent storage"</li> </ul> | <p>Model: Dell R620</p> <p>CPU: 2x Intel® Xeon®</p> <p>CPU E5-2620 0 @ 2.00 GHz</p> <p>Memory: 32 GB</p> <p>Disk: two 300 GB 10000 RPM SAS Disks</p> <p>Network: two 10G network ports</p> |



| Type           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Example hardware                                                                                                                                                                                                                                                                                             |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | <p>through services run on the host as well. This persistent storage is backed onto the storage nodes for reliability.</p> <p>See Fig. 7.3</p>                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                              |
| <b>Compute</b> | <p>Compute nodes run the virtual machine instances in Open-Stack. They:</p> <ul style="list-style-type: none"> <li>• Run the bare minimum of services needed to facilitate these instances.</li> <li>• Use local storage on the node for the virtual machines so that no VM migration or instance recovery at node failure is possible.</li> </ul> <p>See Fig. 7.4</p>                                                                                                                                                  | <p>Model: Dell R620</p> <p>CPU: 2x Intel® Xeon® CPU E5-2650 0 @ 2.00 GHz</p> <p>Memory: 128 GB</p> <p>Disk: two 600 GB 1000 RPM SAS Disks</p> <p>Network: four 10G network ports (For future proofing expansion)</p>                                                                                         |
| <b>Storage</b> | <p>Storage nodes store all the data required for the environment, including disk images in the Image Service library, and the persistent storage volumes created by the Block Storage service. Storage nodes use GlusterFS technology to keep the data highly available and scalable.</p> <p>See Fig. 7.6</p>                                                                                                                                                                                                           | <p>Model: Dell R720xd</p> <p>CPU: 2x Intel® Xeon® CPU E5-2620 0 @ 2.00 GHz</p> <p>Memory: 64 GB</p> <p>Disk: two 500 GB 7200 RPM SAS Disks and twenty-four 600 GB 10000 RPM SAS Disks</p> <p>Raid Controller: PERC H710P Integrated RAID Controller, 1 GB NV Cache</p> <p>Network: two 10G network ports</p> |
| <b>Network</b> | <p>Network nodes are responsible for doing all the virtual networking needed for people to create public or private networks and uplink their virtual machines into external net-works. Network nodes:</p> <ul style="list-style-type: none"> <li>• Form the only ingress and egress point for instances running on top of OpenStack.</li> <li>• Run all of the environment's networking services, with the exception of the networking API service (which runs on the controller node).</li> </ul> <p>See Fig. 7.5</p> | <p>Model: Dell R620</p> <p>CPU: 1x Intel® Xeon® CPU E5-2620 0 @ 2.00 GHz</p> <p>Memory: 32 GB</p> <p>Disk: two 300 GB 10000 RPM SAS Disks</p> <p>Network: five 10G network ports</p>                                                                                                                         |

| Type    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Example hardware                                                                                                                                                                   |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Utility | <p>Utility nodes are used by internal administration staff only to provide a number of basic system administration functions needed to get the environment up and running and to maintain the hardware, OS, and software on which it runs.</p> <p>These nodes run services such as provisioning, configuration management, monitoring, or GlusterFS management software. They are not required to scale, although these machines are usually backed up.</p> | <p>Model: Dell R620</p> <p>CPU: 2x Intel® Xeon® CPU E5-2620 0 @ 2.00 GHz</p> <p>Memory: 32 GB</p> <p>Disk: two 500 GB 7200 RPM SAS Disks</p> <p>Network: two 10G network ports</p> |

### Networking layout

The network contains all the management devices for all hardware in the environment (for example, by including Dell iDrac7 devices for the hardware nodes, and management interfaces for network switches). The network is accessed by internal staff only when diagnosing or recovering a hardware issue.

### OpenStack internal network

This network is used for OpenStack management functions and traffic, including services needed for the provisioning of physical nodes (pxe, tftp, kickstart), traffic between various OpenStack node types using OpenStack APIs and messages (for example, nova-compute talking to key-stone or cinder-volume talking to nova-api), and all traffic for storage data to the storage layer underneath by the Gluster protocol. All physical nodes have at least one network interface (typically eth0) in this network. This network is only accessible from other VLANs on port 22 (for ssh access to manage machines).

### Public Network

This network is a combination of:

- IP addresses for public-facing interfaces on the controller nodes (which end users will access the OpenStack services)
- A range of publicly routable, IPv4 network addresses to be used by OpenStack Networking for floating IPs. You may be restricted in your access to IPv4 addresses; a large range of IPv4 addresses is not necessary.
- Routers for private networks created within OpenStack.

This network is connected to the controller nodes so users can access the OpenStack interfaces, and connected to the network nodes to provide VMs with publicly routable traffic functionality. The network is also connected to the utility machines so that any utility services that need to be made public (such as system monitoring) can be accessed.

### VM traffic network

This is a closed network that is not publicly routable and is simply used as a private, internal network for traffic between virtual machines in OpenStack, and between the virtual machines and the network nodes that provide I3 routes out to the public network (and floating IPs for connections back in to the VMs). Because this is a closed network, we are using a different address space to the others to clearly define the separation. Only Compute and OpenStack Networking nodes need to be connected to this network.

### Node connectivity

The following section details how the nodes are connected to the different networks (see the section called “Networking layout”) and what other considerations need to take place (for example, bonding) when connecting nodes to the networks.

### Initial deployment

Initially, the connection setup should revolve around keeping the connectivity simple and straightforward in order to minimize deployment complexity and time to deploy. The deployment shown in Fig. 7.1, “Basicnode deployment” aims to have  $1 \times 10G$  connectivity available to all compute nodes, while still leveraging bonding on appropriate nodes for maximum performance.

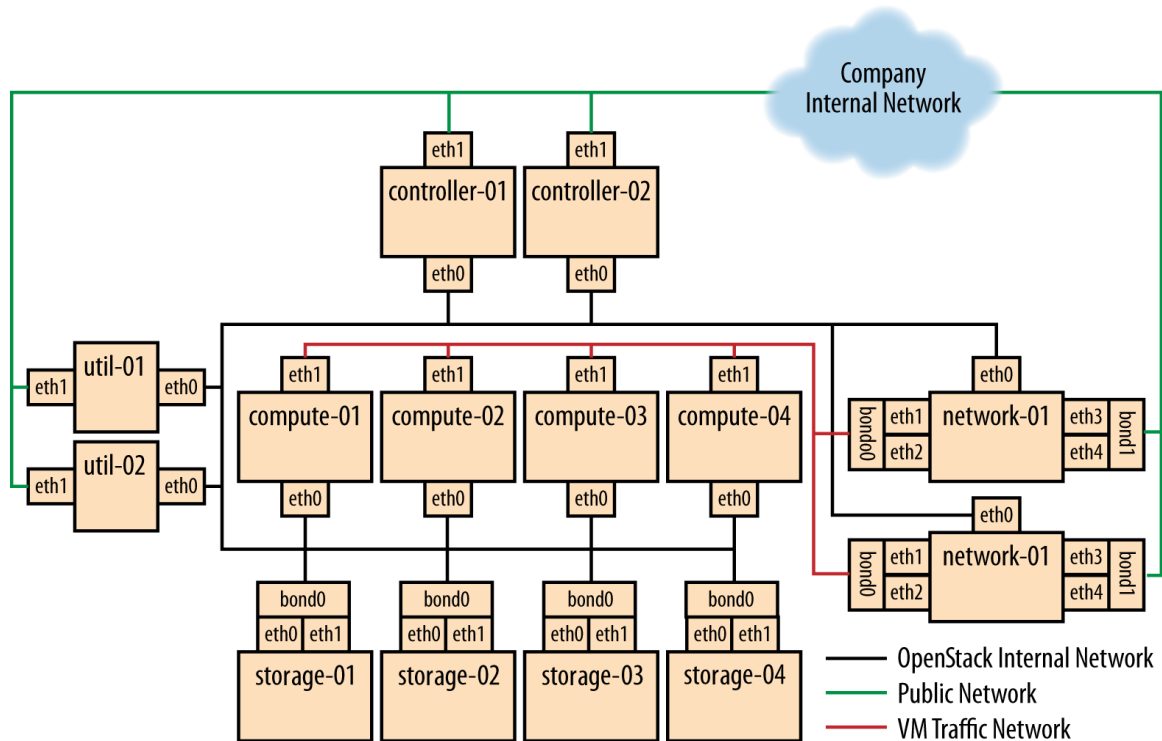


Fig. 7.4

**Connectivity for maximum performance**

If the networking performance of the basic layout is not enough, you can move to Fig. 7.2, “Performance node deployment”, which provides  $2 \times 10\text{G}$  network links to all instances in the environment as well as providing more network bandwidth to the storage layer.

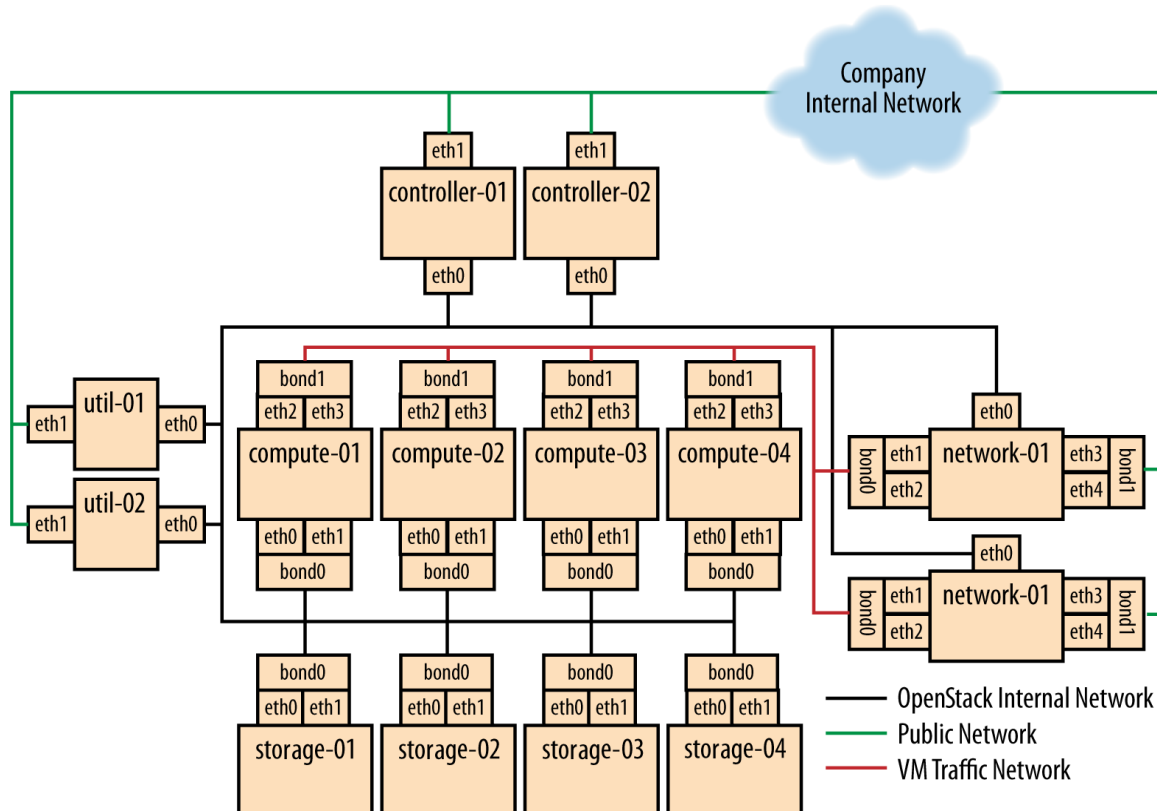


Fig. 7.5

**Node diagrams**

The following diagrams (Fig. 7.3, through Fig. 7.6) include logical information about the different types of nodes, indicating what services will be running on top of them and how they interact with each other. The diagrams also illustrate how the availability and scalability of services are achieved.

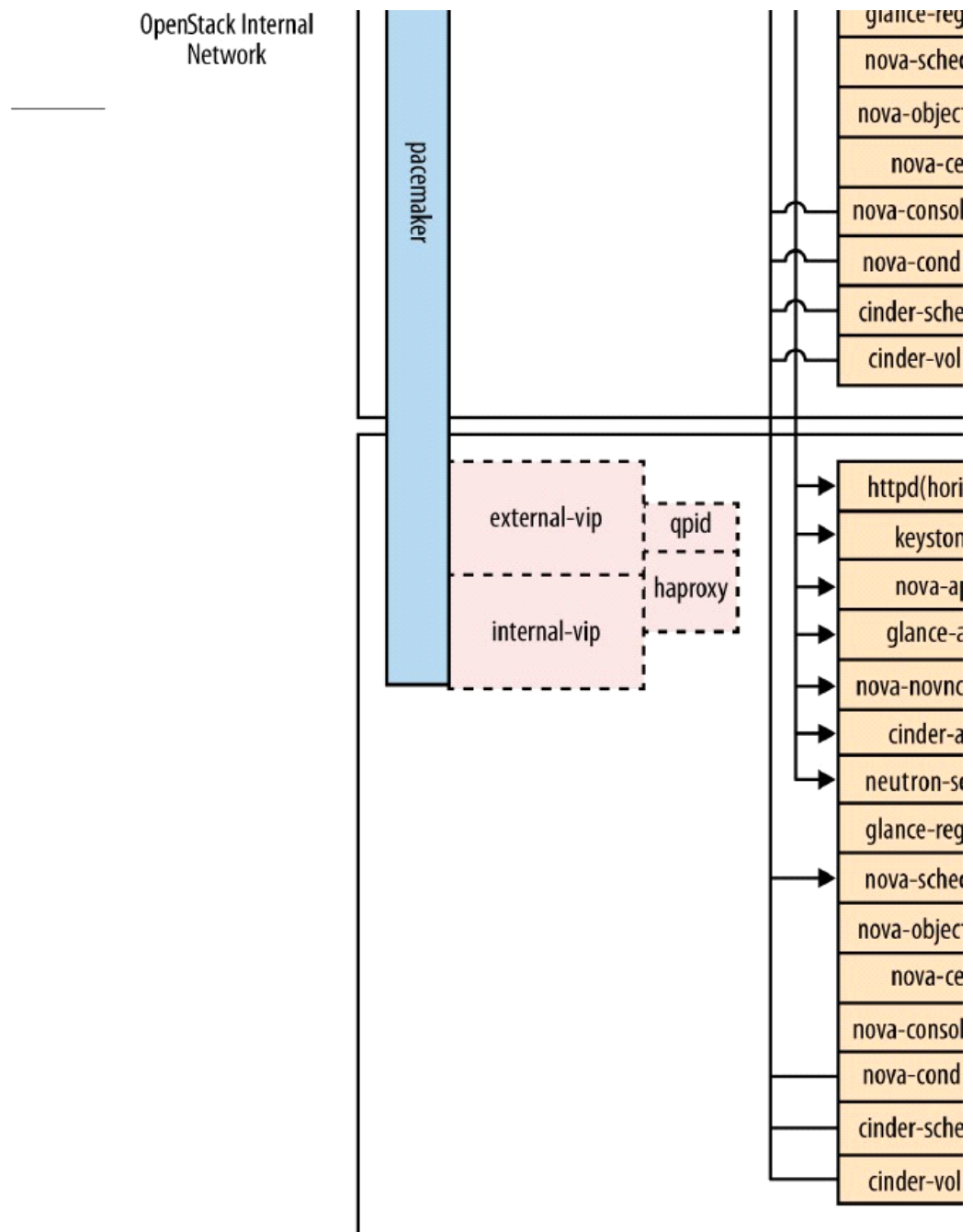


Fig. 7.6

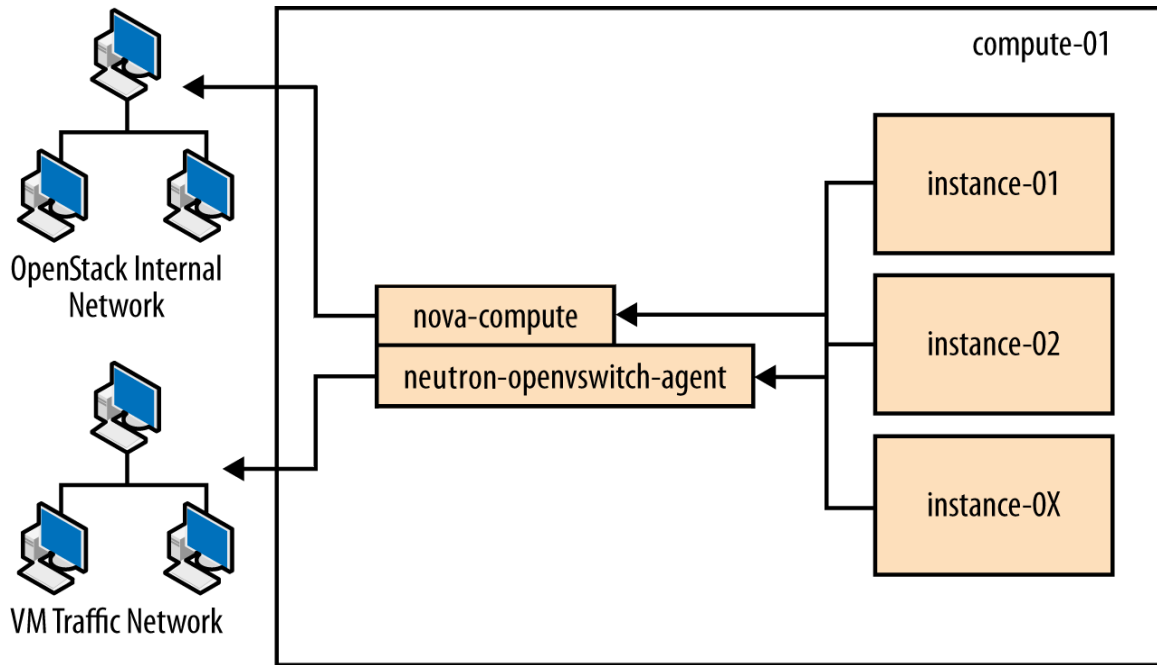


Fig. 7.7

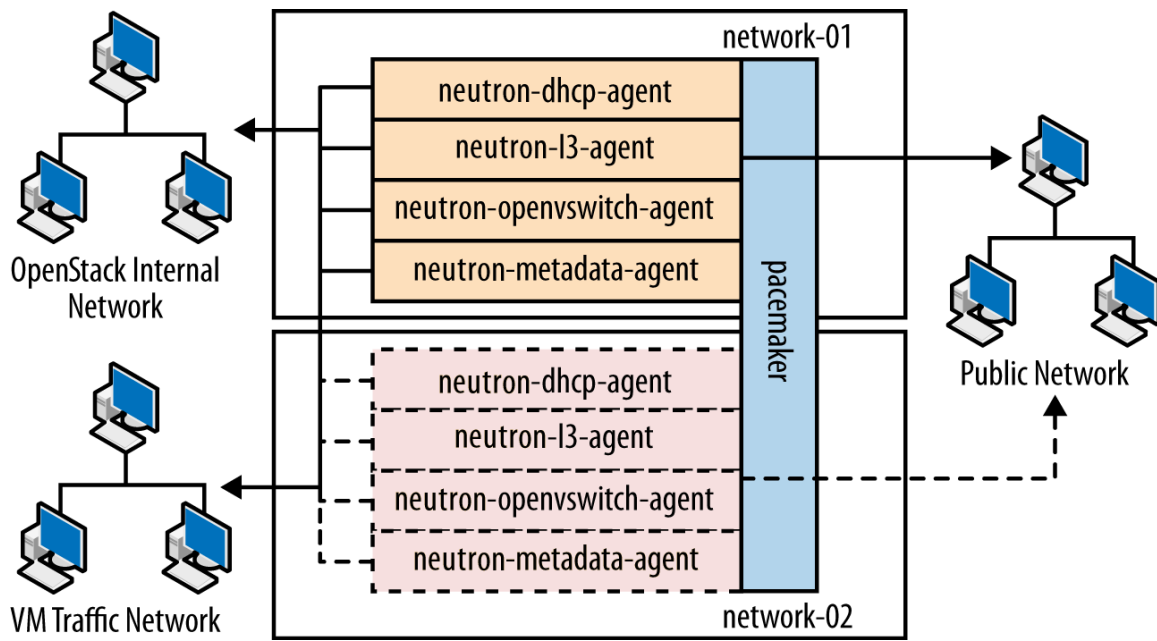


Fig. 7.8

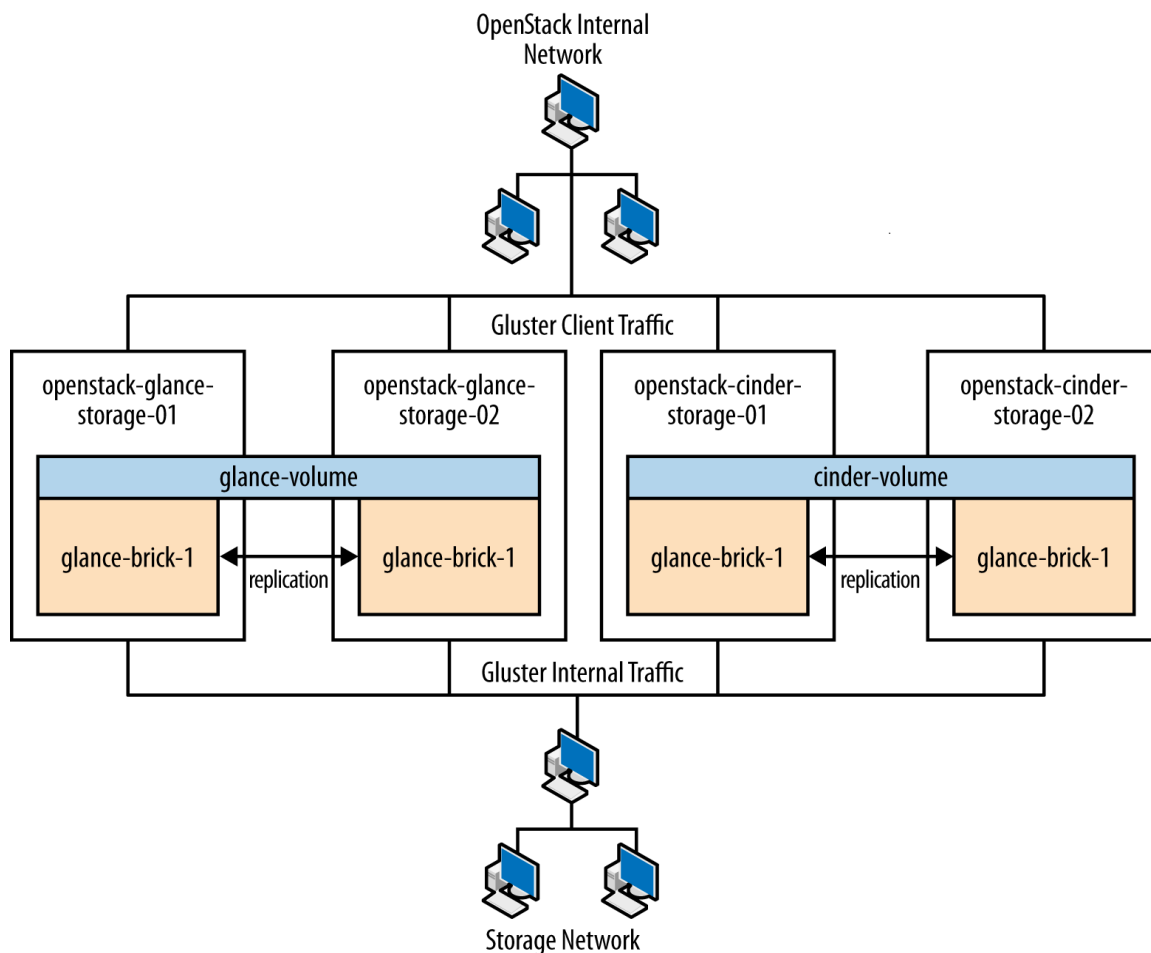


Fig. 7.9

**Example Component Configuration**

Table 7.2, “Third-party component configuration” and Table 7.3, “OpenStack component configuration” include example configuration and considerations for both third-party and OpenStack components:

**Table 7.2. Third-party component configuration**

| Component | Tuning              | Availability                                                                                                                                                                                                                                                                                                                                                                                                           | Scalability                                                                                                                                                           |
|-----------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MySQL     | binlog-format = row | Master/master replication. However, both nodes are not used at the same time. Replication keeps all nodes as close to being up to date as possible (although the asynchronous nature of the replication means a fully consistent state is not possible). Connections to the database only happen through a Pacemaker virtual IP, ensuring that most problems that occur with master-master replication can be avoided. | Not heavily considered. Once load on the MySQL server increases enough that scalability needs to be considered, multiple masters or a master/slave setup can be used. |

| Component | Tuning                                                                                                                       | Availability                                                                                                                                                                                                                                                                                                                                                                                                    | Scalability                                                                                                                                                                                                                                                                                            |
|-----------|------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Qpid      | max-connections=1000<br>worker-threads=20<br>connection-back-log=10,<br>sasl security enabled with SASL-BASIC authentication | Qpid is added as a resource to the Pacemaker software that runs on Controller nodes where Qpid is situated. This ensures only one Qpid instance is running at one time, and the node with the Pacemaker virtual IP will always be the node running Qpid.                                                                                                                                                        | Not heavily considered. However, Qpid can be changed to run on all controller nodes for scalability and availability purposes, and removed from Pacemaker.                                                                                                                                             |
| HAProxy   | maxconn 3000                                                                                                                 | HAProxy is a software layer-7 load balancer used to front door all clustered OpenStack API components and do SSL termination. HAProxy can be added as a resource to the Pacemaker software that runs on the Controller nodes where HAProxy is situated. This ensures that only one HAProxy instance is running at one time, and the node with the Pacemaker virtual IP will always be the node running HAProxy. | Not considered. HAProxy has small enough performance overheads that a single instance should scale enough for this level of workload. If extra scalability is needed, keep alive or other Layer-4 load balancing can be introduced to be placed in front of multiple copies of HAProxy.                |
| Memcached | MAXCONN="8192"<br>CACHE_SIZE="30457"                                                                                         | Memcached is a fast in memory key-value cache software that is used by OpenStack components for caching data and increasing performance. Memcached runs on all controller nodes, ensuring that should one go down, another instance of Memcached is available.                                                                                                                                                  | Not considered. A single instance of Memcached should be able to scale to the desired workloads. If scalability is desired, HAProxy can be placed in front of Memcached (in raw tcp mode) to utilize multiple Memcached instances for scalability. However, this might cause cache consistency issues. |
| Pacemaker | Configured to use corosync and cman as a cluster communication stack/quorum manager, and as a two-node cluster.              | Pacemaker is the clustering software used to ensure the availability of services running on the controller and network nodes: <ul style="list-style-type: none"> <li>• Because Pacemaker is cluster software, the software itself handles its own availability, leveraging corosync and cman underneath.</li> </ul>                                                                                             | If more nodes need to be made cluster aware, Pacemaker can scale to 64 nodes                                                                                                                                                                                                                           |



| Component | Tuning                                                                                                  | Availability                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Scalability                                                                            |
|-----------|---------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| GlusterFS | Glusterfs performance profile "virt" enabled on all volumes. Volumes are setup in two-node replication. | <ul style="list-style-type: none"> <li>If you use the GlusterFS native client, no virtual IP is needed, since the client knows all about nodes after initial connection and automatically routes around failures on the client side.</li> <li>If you use the NFS or SMB adaptor, you will need a virtual IP on which to mount the GlusterFS volumes.</li> </ul> <p>Glusterfs is a clustered file system that is run on the storage nodes to provide persistent scalable data storage in the environment. Because all connections to gluster use the gluster native mount points, the gluster instances themselves provide availability and failover functionality.</p> | The scalability of GlusterFS storage can be achieved by adding in more storage volumes |

**Table 7.3. OpenStack component configuration**

| Component              | Node type  | Tuning                                                                                                  | Availability                                                                                                                                                                                                                                 | Scalability                                                                                                                                                                          |
|------------------------|------------|---------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Dashboard (horizon)    | Controller | Configured to use Memcached as a session store, neutron support is enabled, can_set_mount_point = False | The dashboard is run on all controller nodes, ensuring at least one instance will be available in case of node failure. It also sits behind HAProxy, which detects when the software fails and routes requests around the failing instance.  | The dashboard is run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for the dashboard as more nodes are added. |
| Identity (keystone)    | Controller | Configured to use Memcached for caching and PKI for tokens.                                             | Identity is run on all controller nodes, ensuring at least one instance will be available in case of node failure. Identity also sits behind HAProxy, which detects when the software fails and routes requests around the failing instance. | Identity is run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for Identity as more nodes are added.           |
| Image Service (glance) | Controller | /var/lib/glance/images is a GlusterFS native mount to a                                                 | The Image Service is run on all controller nodes, ensuring at least one instance will be available in case of                                                                                                                                | The Image Service is run on all controller nodes, so scalability                                                                                                                     |

| Component                      | Node type                    | Tuning                                                                                                                                                                                                                                                                                           | Availability                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Scalability                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------------------|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compute (nova)                 | Controller, Compute          | <p>Configured to use Qpid, qpid_heartbeat = 10, configured to use Memcached for caching, configured to use libvirt, configured to use neutron.</p> <p>Configured no-va-consoleauth to use Memcached for session management (so that it can have multiple copies and run in a load balancer).</p> | <p>node failure. It also sits behind HAProxy, which detects when the software fails and routes requests around the failing instance.</p> <p>The nova API, scheduler, objectstore, cert, consoleauth, conductor, and vncproxy services are run on all controller nodes, ensuring at least one instance will be available in case of node failure. Compute is also behind HAProxy, which detects when the software fails and routes requests around the failing instance.</p> <p>Compute's compute and conductor services, which run on the compute nodes, are only needed to run services on that node, so availability of those services is coupled tightly to the nodes that are available. As long as a compute node is up, it will have the needed services running on top of it.</p> | <p>can be achieved with additional controller nodes. HAProxy allows scalability for the Image Service as more nodes are added.</p> <p>The nova API, scheduler, objectstore, cert, consoleauth, conductor, and vncproxy services are run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for Compute as more nodes are added. The scalability of services running on the compute nodes (compute, conductor) is achieved linearly by adding in more compute nodes.</p> |
| Block Storage (cinder)         | Controller                   | Configured to use Qpid, qpid_heartbeat = 10, configured to use a Gluster volume from the storage layer as the backend for Block Storage, using the Gluster native client.                                                                                                                        | Block Storage API, scheduler, and volume services are run on all controller nodes, ensuring at least one instance will be available in case of node failure. Block Storage also sits behind HAProxy, which detects if the software fails and routes requests around the failing instance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Block Storage API, scheduler and volume services are run on all controller nodes, so scalability can be achieved with additional controller nodes. HAProxy allows scalability for Block Storage as more nodes are added.                                                                                                                                                                                                                                                                                                                  |
| OpenStack Networking (neutron) | Controller, Compute, Network | Configured to use QPID, qpid_heartbeat = 10, kernel namespace support enabled, tenant_network_type = vlan, allow_overlapping_ip                                                                                                                                                                  | The OpenStack Networking service is run on all controller nodes, ensuring at least one instance will be available in case of node failure. It also sits behind HAProxy, which detects if the software fails and routes requests around the failing instance.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | The OpenStack Networking server service is run on all controller nodes, so scalability can be achieved with additional controller nodes.                                                                                                                                                                                                                                                                                                                                                                                                  |

| Component | Node type | Tuning                                                                                                              | Availability                                                                                                                                                                                                                                                                                                                                                                                                                                        | Scalability                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------|-----------|---------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           |           | s= true,<br>tenant_network_type<br>= vlan,<br>bridge_uplinks<br>= br-ex:em2,<br>bridge_mappings<br>= physnet1:br-ex | OpenStack Networking's ovs-agent, l3-agent, dhcp-agent, and meta-data-agent services run on the network nodes, as lsb resources inside of Pacemaker. This means that in the case of network node failure, services are kept running on another node. Finally, the ovs-agent service is also run on all compute nodes, and in case of compute node failure, the other nodes will continue to function using the copy of the service running on them. | HAProxy allows scalability for OpenStack Networking as more nodes are added. Scalability of services running on the network nodes is not currently supported by OpenStack Networking, so they are not be considered. One copy of the services should be sufficient to handle the workload. Scalability of the ovs-agent running on compute nodes is achieved by adding in more compute nodes as necessary. |

### 7.2.2 Parting Thoughts on Architectures

With so many considerations and options available, our hope is to provide a few clearly marked and tested paths for your OpenStack exploration.

### 7.2.3 Provisioning and Deployment

A critical part of a cloud's scalability is the amount of effort that it takes to run your cloud. To minimize the operational cost of running your cloud, set up and use an automated deployment and configuration infrastructure with a configuration management system, such as Puppet or Chef. Combined, these systems greatly reduce manual effort and the chance for operator error. This infrastructure includes systems to automatically install the operating system's initial configuration and later coordinate the configuration of all services automatically and centrally, which reduces both manual effort and the chance for error. Examples include Ansible, Chef, Puppet, and Salt. a

#### 7.2.3.1 Automated Deployment

An automated deployment system installs and configures operating systems on new servers, without intervention, after the absolute minimum amount of manual work, including physical racking, MAC-to-IP assignment, and power configuration. Typically, solutions rely on wrappers around PXE boot and TFTP servers for the basic operating system install and then hand off to an automated configuration management system. Both Ubuntu and Red Hat Enterprise Linux include mechanisms for configuring the operating system, including preseed and kick start that you can use after a network boot. Typically, these are used to bootstrap an automated configuration system. Alternatively, you can use an image-based approach for deploying the operating system, such as system imager. You can use both approaches with a virtualized infrastructure, such as when you run VMs to separate your control services and physical infrastructure. When you create a deployment plan, focus on a few vital areas

because they are very hard to modify post deployment. The next two sections talk about configurations for:

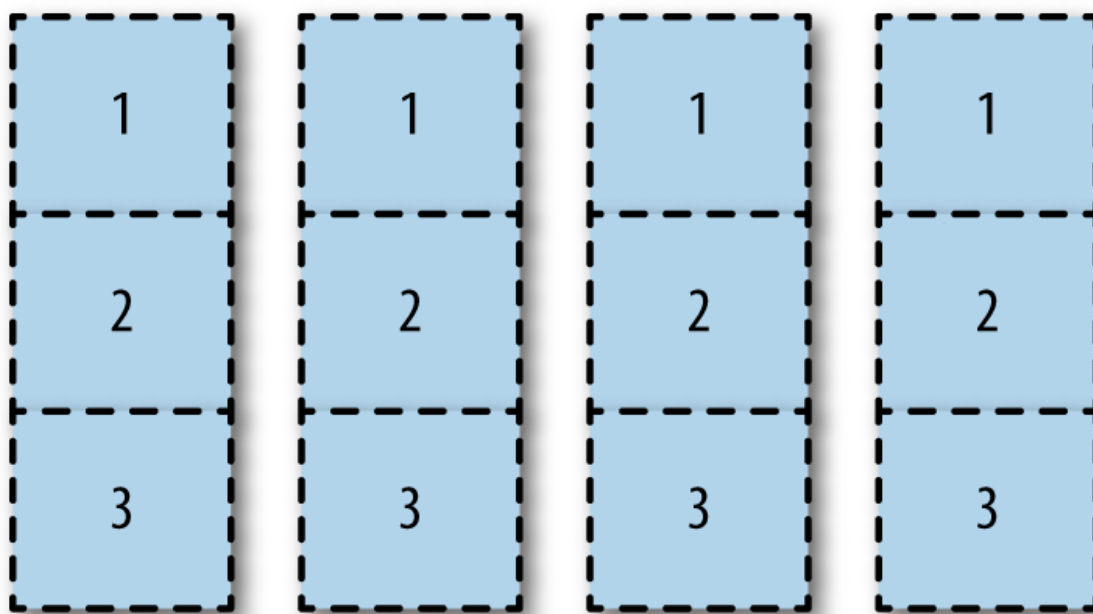
- Disk partitioning and disk array setup for scalability
- Networking configuration just for PXE booting

### Disk Partitioning and RAID

At the very base of any operating system are the hard drives on which the operating system (OS) is installed. You must complete the following configurations on the server's hard drives:

- Partitioning, which provides greater flexibility for layout of operating system and swap space, as described below.
- Adding to a RAID array (RAID stands for redundant array of independent disks), based on the number of disks you have available, so that you can add capacity as your cloud grows. Some options are described in more detail below. The simplest option to get started is to use one hard drive with two partitions:
- File system to store files and directories, where all the data lives, including the root partition that starts and runs the system
- Swap space to free up memory for processes, as an independent area of the physical disk used only for swapping and nothing else RAID is not used in this simplistic one-drive setup because generally for production clouds, you want to ensure that if one disk fails, another can take its place. Instead, for production, use more than one disk. The number of disks determine what types of RAID arrays to build. We recommend that you choose one of the following multiple disk options:

**Option 1** Partition all drives in the same way in a horizontal fashion, as shown in Fig. 7.10, “Partition setup of drives”. With this option, you can assign different partitions to different RAID arrays. You can allocate partition 1 of disk one and two to the /boot partition mirror. You can make partition 2 of all disks the root partition mirror. You can use partition 3 of all disks for a cinder-volumes LVM partition running on a RAID 10 array. Open Stack Ops Guide November 8, 2014



**Fig. 7.10. Partition setup of drives**

While you might end up with unused partitions, such as partition 1 in disk three and four of this example, this option allows for maximum utilization of disk space. I/O performance might be an issue as a result of all disks being used for all tasks.

**Option 2** Add all raw disks to one large RAID array, either hardware or software based. You can partition this large array with the boot, root, swap, and LVM areas. This option is simple to implement and uses all partitions. However, disk I/O might suffer.

**Option 3** Dedicate entire disks to certain partitions. For example, you could allocate disk one and two entirely to the boot, root, and swap partitions under a RAID 1 mirror. Then, allocate disk three and four entirely to the LVM partition, also under a RAID 1 mirror. Disk I/O should be better because I/O is focused on dedicated tasks. However, the LVM partition is much smaller.

### Tip

You may find that you can automate the partitioning itself. For example, MIT uses Fully Automatic Installation (FAI) to do the initial PXE-based partition and then install using a combination of min/max and percentage-based partitioning. As with most architecture choices, the right answer depends on your environment. If you are using existing hardware, you know the disk density of your servers and can determine some decisions based on the options above. If you are going through a procurement process, your user's requirements also help you determine hardware purchases. Here are some examples from a private cloud providing web developer's custom environments at AT&T. This example is from a specific deployment, so your existing hardware or procurement opportunity may vary from this. AT&T uses three types of hardware in its deployment:

- Hardware for controller nodes, used for all stateless Open Stack API services. About 32–64 GB memory, small attached disk, one processor, varied number of cores, such as 6–12.
- Hardware for compute nodes. Typically 256 or 144 GB memory, two processors, 24 cores. 4–6 TB direct attached storage, typically in a RAID 5 configuration.
- Hardware for storage nodes. Typically for these, the disk space is optimized for the lowest cost per GB of storage while maintaining rack space efficiency. Again, the right answer depends on your environment. You have to make your decision based on the trade-offs between space utilization, simplicity, and I/O performance.

### Network Configuration

Network configuration is a very large topic that spans multiple areas of this book. For now, make sure that your server's can PXE boot and successfully communicate with the deployment server. For example, you usually cannot configure NICs for VLANs when PXE booting. Additionally, you usually cannot PXE boot with bonded NICs. If you run into this scenario, consider using a simple 1 GB switch in a private network on which only your cloud communicates.

#### 7.2.3.2 Automated Configuration

The purpose of automatic configuration management is to establish and maintain the consistency of a system without using human intervention. You want to maintain consistency in your deployments so that you can have the same cloud every time, repeatable. Proper use of automatic configuration-management tools ensures that components of the cloud systems are in particular states, in addition to simplifying deployment, and configuration change propagation. These tools also make it possible to test and roll back changes, as they are fully repeatable. Conveniently, a large body of work has been done by the OpenStack community in this space. Puppet, a configuration management OpenStack Ops Guide November 8, 2014 tool, even provides official modules for OpenStack in an OpenStack infrastructure system known as Stack forge. Chef configuration management is provided within <https://github.com/stackforge/openstack-chef-repo>. Additional configuration management systems include Juju, Ansible, and Salt. Also, Pack Stack is a command-line utility for Red Hat Enterprise Linux and derivatives that uses Puppet modules to support rapid deployment of OpenStack on existing servers over an SSH connection.

An integral part of a configuration-management system is the items that it controls. You should carefully consider all of the items that you want, or do not want, to be automatically managed. For example, you may not want to automatically format hard drives with user data.

### 7.2.3.3 Remote Management

In our experience, most operators don't sit right next to the servers running the cloud, and many don't necessarily enjoy visiting the data center. OpenStack should be entirely remotely configurable, but sometimes not everything goes according to plan.

In this instance, having an out-of-band access into nodes running Open-Stack components is a boon. The IPMI protocol is the de facto standard here, and acquiring hardware that supports it is highly recommended to achieve that lights-out data center aim.

In addition, consider remote power control as well. While IPMI usually controls the server's power state, having remote access to the PDU that the server is plugged into can really be useful for situations when everything seems wedged.

### 7.2.3.4 Parting Thoughts for Provisioning and Deploying OpenStack

You can save time by understanding the use cases for the cloud you want to create. Use cases for OpenStack are varied. Some include object storage only; others require preconfigured compute resources to speed development- environment set up; and others need fast provisioning of compute resources that are already secured per tenant with private networks. Your users may have need for highly redundant servers to make sure their legacy applications continue to run. Perhaps a goal would be to architect these legacy applications so that they run on multiple instances in a cloudy, fault tolerant way, but not make it a goal to add to those clusters over time. OpenStack Ops Guide November 8, 2014 your users may indicate that they need scaling considerations because of heavy Windows server use.

You can save resources by looking at the best fit for the hardware you have in place already. You might have some high-density storage hardware available. You could format and repurpose those servers for Open- Stack Object Storage. All of these considerations and input from users help you build your use case and your deployment plan.

#### Tip

For further research about OpenStack deployment, investigate the supported and documented preconfigured, prepackaged installers for OpenStack from companies such as Canonical, Cisco, Cloud scaling, IBM, Metacloud, Mirantis, Piston, Rackspace, Red Hat, SUSE, and SwiftStack.

### 7.2.3.5 Conclusion

The decisions you make with respect to provisioning and deployment will affect your day-to-day, week-to-week, and month-to-month maintenance of the cloud. Your configuration management will be able to evolve over time. However, more thought and design need to be done for upfront choices about deployment, disk partitioning, and network configuration.

## 7.2.4 Designing for Cloud Controllers and Cloud Management

OpenStack is designed to be massively horizontally scalable, which allows all services to be distributed widely. However, to simplify this guide, we have decided to discuss services of a more central nature, using the concept of a *cloud controller*. A cloud controller is just a conceptual simplification. In the real world, you design an architecture for your cloud controller that enables high availability so that if any node fails, another can take over the required tasks. In reality, cloud controller tasks are spread out across more than a single node.

The cloud controller provides the central management system for Open-Stack deployments. Typically, the cloud controller manages authentication and sends messaging to all the systems through a message queue.

For many deployments, the cloud controller is a single node. However, to have high availability, you have to take a few considerations into account, which we'll cover in this chapter.

The cloud controller manages the following services for the cloud:

|                                                                 |                                                                                                                                       |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>Databases</b>                                                | Tracks current information about users and instances, for example, in a database, typically one database instance managed per service |
| <b>Message queue services</b>                                   | All AMQP—Advanced Message Queue Protocol—messages for services are received and sent according to the queue broker                    |
| <b>Conductor services</b>                                       | Proxy requests to a database                                                                                                          |
| <b>Authentication and authorization for identity management</b> | Indicates which users can do what actions on certain cloud resources; quota management is spread out among services, however          |
| <b>Image-management services</b>                                | Stores and serves images with metadata on each, for launching in the cloud                                                            |
| <b>Scheduling services</b>                                      | Indicates which resources to use first; for example, spreading out where instances are launched based on an algorithm                 |
| <b>User dashboard</b>                                           | Provides a web-based frontend for users to consume OpenStack cloud services                                                           |
| <b>API endpoints</b>                                            | Offers each service's REST API access, where the API endpoint catalog is managed by the Identity Service                              |

For our example, the cloud controller has a collection of nova-\* components that represent the global state of the cloud; talks to services such as authentication; maintains information about the cloud in a database; communicates to all compute nodes and storage *workers* through a queue; and provides API access. Each service running on a designated cloud controller may be broken out into separate nodes for scalability or availability.

As another example, you could use pairs of servers for a collective cloud controller—one active, one standby—for redundant nodes providing a given set of related services, such as:

- Frontend web for API requests, the scheduler for choosing which compute node to boot an instance on, Identity services, and the dashboard
- Database and message queue server (such as MySQL, RabbitMQ)
- Image Service for the image management

Now that you see the myriad designs for controlling your cloud, read more about the further considerations to help with your design decisions. OpenStack Ops Guide November 8, 2014

#### 7.2.4.1 Hardware Considerations

A cloud controller's hardware can be the same as a compute node, though you may want to further specify based on the size and type of cloud that you run.

It's also possible to use virtual machines for all or some of the services that the cloud controller manages, such as the message queuing. In this guide, we assume that all services are running directly on the cloud controller.

Table 7.3, “Cloud controller hardware sizing considerations” contains common considerations to review when sizing hardware for the cloud controller design.

**Table 7.3 Cloud controller hardware sizing considerations**

| Consideration                                                                 | Ramification                                                                                                                                                                                                                                |
|-------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| How many instances will run at once?                                          | Size your database server accordingly, and scale out beyond one cloud controller if many instances will report status at the same time and scheduling where a new instance starts up needs computing power.                                 |
| How many compute nodes will run at once?                                      | Ensure that your messaging queue handles requests successfully and size accordingly.                                                                                                                                                        |
| How many users will access the API?                                           | If many users will make multiple requests, make sure that the CPU load for the cloud controller can handle it.                                                                                                                              |
| How many users will access the <i>dashboard</i> versus the REST API directly? | The dashboard makes many requests, even more than the API access, so add even more CPU if your dashboard is the main interface for your users.                                                                                              |
| How many nova-api services do you run at once for your cloud?                 | You need to size the controller with a core per service.                                                                                                                                                                                    |
| How long does a single instance run?                                          | Starting instances and deleting instances is demanding on the compute node but also demanding on the controller node because of all the API queries and scheduling needs.                                                                   |
| Does your authentication system also verify externally?                       | External systems such as LDAP or <i>Active Directory</i> require network connectivity between the cloud controller and an external authentication system. Also ensure that the cloud controller has the CPU power to keep up with requests. |

#### 7.2.4.2 Separation of Services:

While our example contains all central services in a single location, it is possible and indeed often a good idea to separate services onto different physical servers. Table 7.4, “Deployment scenarios” is a list of deployment scenarios we've seen and their justifications. OpenStack Ops Guide November 8, 2014

**Table 7.4 Deployment scenarios**

| Scenario                                        | Justification                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Run glance-* servers on the swift-proxy server. | This deployment felt that the spare I/O on the Object Storage proxy server was sufficient and that the Image Delivery portion of glance benefited from being on physical hardware and having good connectivity to the Object Storage backend it was using.                                                                                                                       |
| Run a central dedicated database server.        | This deployment used a central dedicated server to provide the databases for all services. This approach simplified operations by isolating database server updates and allowed for the simple creation of slave database servers for failover.                                                                                                                                  |
| Run one VM per service.                         | This deployment ran central services on a set of servers running KVM. A dedicated VM was created for each service (nova-scheduler, rabbitmq, database, etc). This assisted the deployment with scaling because administrators could tune the resources given to each virtual machine based on the load it received (something that was not well understood during installation). |
| Use an external load balancer.                  | This deployment had an expensive hardware load balancer                                                                                                                                                                                                                                                                                                                          |



|  |                                                                                                                                                        |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | in its organization. It ran multiple nova-api and swift-proxy servers on different physical servers and used the load balancer to switch between them. |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------|

One choice that always comes up is whether to virtualize. Some services, such as nova-compute, swift-proxy and swift-object servers, should not be virtualized. However, control servers can often be happily virtualized—the performance penalty can usually be offset by simply running more of the service.

#### 7.2.4.3 Database

OpenStack Compute uses a SQL database to store and retrieve stateful information. MySQL is the popular database choice in the OpenStack community.

Loss of the database leads to errors. As a result, we recommend that you cluster your database to make it failure tolerant. Configuring and maintaining a database cluster is done outside OpenStack and is determined by the database software you choose to use in your cloud environment. MySQL/Galera is a popular option for MySQL-based databases.

#### 7.2.4.4 Message Queue

Most OpenStack services communicate with each other using the *message queue*. For example, Compute communicates to block storage services and networking services through the message queue. Also, you can optionally enable notifications for any service. RabbitMQ, Qpid, and Omq are all OpenStack Ops Guide November 8, 2014 popular choices for a message-queue service. In general, if the message queue fails or becomes inaccessible, the cluster grinds to a halt and ends up in a read-only state, with information stuck at the point where the last message was sent. Accordingly, we recommend that you cluster the message queue. Be aware that clustered message queues can be a pain point for many OpenStack deployments. While RabbitMQ has native clustering support, there have been reports of issues when running it at a large scale. While other queuing solutions are available, such as Omq and Qpid, Omq does not offer stateful queues. Qpid is the messaging system of choice for Red Hat and its derivatives. Qpid does not have native clustering capabilities and requires a supplemental service, such as Pacemaker or Corsync. For your message queue, you need to determine what level of data loss you are comfortable with and whether to use an OpenStack project's ability to retry multiple MQ hosts in the event of a failure, such as using Compute's ability to do so.

#### 7.2.4.5 Conductor Services

In the previous version of OpenStack, all nova-compute services required direct access to the database hosted on the cloud controller. This was problematic for two reasons: security and performance. With regard to security, if a compute node is compromised, the attacker inherently has access to the database. With regard to performance, nova-compute calls to the database are single-threaded and blocking. This creates a performance bottleneck because database requests are fulfilled serially rather than in parallel. The conductor service resolves both of these issues by acting as a proxy for the nova-compute service. Now, instead of nova-compute directly accessing the database, it contacts the nova-conductor service, and nova-conductor accesses the database on nova-compute's behalf. Since nova-compute no longer has direct access to the database, the security issue is resolved. Additionally, nova-conductor is a non-blocking service, so requests from all compute nodes are fulfilled in parallel.

#### Note

If you are using nova-network and multi-host networking in your cloud environment, nova-compute still requires direct access to the database.

The nova-conductor service is horizontally scalable. To make nova-conductor highly available and fault tolerant, just launch more instances of the nova-conductor process, either on the same server or across multiple servers.

#### 7.2.4.6 Application Programming Interface (API)

All public access, whether direct, through a command-line client, or through the web-based dashboard, uses the API service. Find the API reference at <http://api.openstack.org/>.

You must choose whether you want to support the Amazon EC2 compatibility APIs, or just the OpenStack APIs. One issue you might encounter when running both APIs is an inconsistent experience when referring to images and instances.

For example, the EC2 API refers to instances using IDs that contain hexadecimal, whereas the OpenStack API uses names and digits. Similarly, the EC2 API tends to rely on DNS aliases for contacting virtual machines, as opposed to OpenStack, which typically lists IP addresses.

If OpenStack is not set up in the right way, it is simple to have scenarios in which users are unable to contact their instances due to having only an incorrect DNS alias. Despite this, EC2 compatibility can assist users migrating to your cloud.

As with databases and message queues, having more than one *API server* is a good thing. Traditional HTTP load-balancing techniques can be used to achieve a highly available nova-api service.

#### 7.2.4.7 Extensions

The API Specifications define the core actions, capabilities, and mediatypes of the OpenStack API. A client can always depend on the availability of this core API, and implementers are always required to support it in its entirety. Requiring strict adherence to the core API allows clients to rely upon a minimal level of functionality when interacting with multiple implementations of the same API.

The OpenStack Compute API is extensible. An extension adds capabilities to an API beyond those defined in the core. The introduction of new features, MIME types, actions, states, headers, parameters, and resources can all be accomplished by means of extensions to the core API. This allows the introduction of new features in the API without requiring a version change and allows the introduction of vendor-specific niche functionality.

#### 7.2.4.8 Scheduling

The scheduling services are responsible for determining the compute or storage node where a virtual machine or block storage volume should be created. The scheduling services receive creation requests for these resources from the message queue and then begin the process of determining the appropriate node where the resource should reside. This process is done by applying a series of user-configurable filters against the available collection of nodes.

There are currently two schedulers: nova-scheduler for virtual machines and cinder-scheduler for block storage volumes. Both schedulers are able to scale horizontally, so for high-availability purposes, or for very large or high-schedule-frequency installations, you should consider running multiple instances of each scheduler. The schedulers all listen to the shared message queue, so no special load balancing is required.

#### 7.2.4.9 Images

The OpenStack Image Service consists of two parts: glance-api and glance-registry. The former is responsible for the delivery of images; the compute node uses it to download images from the backend. The latter maintains the metadata information associated with virtual machine images and requires a database.

The glance-api part is an abstraction layer that allows a choice of backend. Currently, it supports:

|                                 |                                        |
|---------------------------------|----------------------------------------|
| <b>OpenStack Object Storage</b> | Allows you to store images as objects. |
|---------------------------------|----------------------------------------|

|                    |                                                                                           |
|--------------------|-------------------------------------------------------------------------------------------|
| <b>File system</b> | Uses any traditional file system to store the images as files.                            |
| <b>S3</b>          | Allows you to fetch images from Amazon S3.                                                |
| <b>HTTP</b>        | Allows you to fetch images from a web server. You cannot write images by using this mode. |

If you have an OpenStack Object Storage service, we recommend using this as a scalable place to store your images. You can also use a file system with sufficient performance or Amazon S3—unless you do not need the ability to upload new images through OpenStack.

#### **7.2.4.10 Dashboard**

The OpenStack dashboard (horizon) provides a web-based user interface to the various OpenStack components. The dashboard includes an end-user area for users to manage their virtual infrastructure and an admin area for cloud operators to manage the OpenStack environment as a whole.

The dashboard is implemented as a Python web application that normally runs in *Apache* httpd. Therefore, you may treat it the same as any other web application, provided it can reach the API servers (including their admin endpoints) over the network.

#### **7.2.4.11 Authentication and Authorization**

The concepts supporting OpenStack's authentication and authorization are derived from well-understood and widely used systems of a similar nature. Users have credentials they can use to authenticate, and they can be a member of one or more groups (known as projects or tenants, interchangeably).

For example, a cloud administrator might be able to list all instances in the cloud, whereas a user can see only those in his current group. Resources quotas, such as the number of cores that can be used, disk space, and so on, are associated with a project.

The OpenStack Identity Service (keystone) is the point that provides the authentication decisions and user attribute information, which is then used by the other OpenStack services to perform authorization. Policy is set in the `policy.json` file.

The Identity Service supports different plug-ins for authentication decisions and identity storage. Examples of these plug-ins include:

- In-memory key-value Store (a simplified internal storage structure)
- SQL database (such as MySQL or PostgreSQL)
- PAM (Pluggable Authentication Module)
- LDAP (such as OpenLDAP or Microsoft's Active Directory)

Many deployments use the SQL database; however, LDAP is also a popular choice for those with existing authentication infrastructure that needs to be integrated.

#### **7.2.4.12 Network Considerations**

Because the cloud controller handles so many different services, it must be able to handle the amount of traffic that hits it. For example, if you choose to host the OpenStack Imaging Service on the cloud controller, the cloud controller should be able to support the transferring of the images at an acceptable speed.

As another example, if you choose to use single-host networking where the cloud controller is the network gateway for all instances, then the cloud controller must support the total amount of traffic that travels between your cloud and the public Internet.

We recommend that you use a fast NIC, such as 10 GB. You can also choose to use two 10 GB NICs and bond them together. While you might not be able to get a full bonded 20 GB speed, different transmission streams use different NICs. For example, if the cloud controller transfers two images, each image uses a different NIC and gets a full 10 GB of bandwidth.

### 7.2.5 Network Design

OpenStack provides a rich networking environment, and this chapter details the requirements and options to deliberate when designing your cloud.

#### Warning

If this is the first time you are deploying a cloud infrastructure in your organization, after reading this section, your first conversations should be with your networking team. Network usage in a running cloud is vastly different from traditional network deployments and has the potential to be disruptive at both a connectivity and a policy level.

For example, you must plan the number of IP addresses that you need for both your guest instances as well as management infrastructure. Additionally, you must research and discuss cloud network connectivity through proxy servers and firewalls.

In this chapter, we'll give some examples of network implementations to consider and provide information about some of the network layouts that OpenStack uses. Finally, we have some brief notes on the networking services that are essential for stable operation.

#### 7.2.5.1 Management Network

A *management network* (a separate network for use by your cloud operators) typically consists of a separate switch and separate NICs (network interface cards), and is a recommended option. This segregation prevents system administration and the monitoring of system access from being disrupted by traffic generated by guests.

Consider creating other private networks for communication between internal components of OpenStack, such as the message queue and OpenStack Compute. Using a virtual local area network (VLAN) works well for these scenarios because it provides a method for creating multiple virtual networks on a physical network.

#### 7.2.5.2 Public Addressing Options

There are two main types of IP addresses for guest virtual machines: fixed IPs and floating IPs. Fixed IPs are assigned to instances on boot, whereas floating IP addresses can change their association between instances by action of the user. Both types of IP addresses can be either public or private, depending on your use case.

Fixed IP addresses are required, whereas it is possible to run OpenStack without floating IPs. One of the most common use cases for floating IPs is to provide public IP addresses to a private cloud, where there are a limited number of IP addresses available. Another is for a public cloud user to have a "static" IP address that can be reassigned when an instance is upgraded or moved.

Fixed IP addresses can be private for private clouds, or public for public clouds. When an instance terminates, its fixed IP is lost. It is worth noting that newer users of cloud computing may find their ephemeral nature frustrating.

#### 7.2.5.3 IP Address Planning

An OpenStack installation can potentially have many subnets (ranges of IP addresses) and different types of services in each. An IP address plan can assist with a shared understanding of network partition purposes and scalability. Control services can have public and private IP addresses, and as noted above, there are a couple of options for an instance's public addresses.

An IP address plan might be broken down into the following sections:

|                                                       |                                                                                                                                                                                              |
|-------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Subnet router</b>                                  | Packets leaving the subnet go via this address, which could be a dedicated router or a nova-network service.                                                                                 |
| <b>Control services public interfaces</b>             | Public access to swift-proxy, nova-api, glance-api, and horizon come to these addresses, which could be on one side of a load balancer or pointing at individual machines.                   |
| <b>Object Storage cluster internal communications</b> | Traffic among object/account/container servers and between these and the proxy server's internal interface uses this private network.                                                        |
| <b>Compute and storage communications</b>             | If ephemeral or block storage is external to the compute node, this network is used.                                                                                                         |
| <b>Out-of-band remote management</b>                  | If a dedicated remote access controller chip is included in servers, often these are on a separate network.                                                                                  |
| <b>In-band remote management</b>                      | Often, an extra (such as 1 GB) interface on compute or storage nodes is used for system administrators or monitoring tools to access the host instead of going through the public interface. |
| <b>Spare space for future growth</b>                  | Adding more public-facing control services or guest instance IPs should always be part of your plan.                                                                                         |

For example, take a deployment that has both OpenStack Compute and Object Storage, with private ranges 172.22.42.0/24 and 172.22.87.0/26 available. One way to segregate the space might be as follows:

|                 |                 |                                                            |
|-----------------|-----------------|------------------------------------------------------------|
| 172.22.42.0/24: |                 |                                                            |
| 172.22.42.1 -   | 172.22.42.3 -   | subnet routers                                             |
| 172.22.42.4 -   | 172.22.42.20 -  | spare for networks                                         |
| 172.22.42.21 -  | 172.22.42.104 - | Compute node remote access controllers (inc spare)         |
| 172.22.42.105 - | 172.22.42.188 - | Compute node management interfaces (inc spare)             |
| 172.22.42.189 - | 172.22.42.208 - | Swift proxy remote access controllers (inc spare)          |
| 172.22.42.209 - | 172.22.42.228 - | Swift proxy management interfaces (inc spare)              |
| 172.22.42.229 - | 172.22.42.252 - | Swift storage servers remote access controllers(inc spare) |
| 172.22.42.253 - | 172.22.42.254 - | spare                                                      |
| 172.22.87.0/26: |                 |                                                            |
| 172.22.87.1 -   | 172.22.87.3 -   | subnet routers                                             |
| 172.22.87.4 -   | 172.22.87.24 -  | Swift proxy server internal interfaces (inc spare)         |
| 172.22.87.25 -  | 172.22.87.63 -  | Swift object server internal interfaces (inc spare)        |

A similar approach can be taken with public IP addresses, taking note that large, flat ranges are preferred for use with guest instance IPs. Take into account that for some OpenStack networking options, a public IP address in the range of a guest instance public IP address is assigned to the nova-compute host.

### 7.2.5.4 Network Topology

OpenStack Compute with nova-network provides predefined network deployment models, each with its own strengths and weaknesses. The selection of a network manager changes your network topology, so the choice should be made carefully. You also have a choice between the tried and true legacy nova-network settings or the neutron project for Open- Stack Networking. Both offer networking for launched instances with different implementations and requirements.

For OpenStack Networking with the neutron project, typical configurations are documented with the idea that any setup you can configure with real hardware you can re-create with a software-defined equivalent. Each tenant can contain typical network elements such as routers, and services such as DHCP.

Table 7.5 “Networking deployment options” discusses the networking deployment options for both legacy nova-network options and an equivalent neutron configuration.

**Table 7.5 Networking deployment options**

| Network deployment model                         | Strengths                                                                                                                   | Weaknesses                                                                                                                                                                                                                    | Neutron equivalent                                                                                                     |
|--------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| Flat                                             | Extremely simple topology.<br><br>No DHCP overhead.                                                                         | Requires file injection into the instance to configure network interfaces.                                                                                                                                                    | Configure a single bridge connect it to a physical 2 (ML2) plug-in,                                                    |
| Flat DHCP                                        | Relatively simple to deploy.<br><br>Standard networking.<br><br>Works with all guest operating systems.                     | Requires its own DHCP broadcast domain.                                                                                                                                                                                       | Configure DHCP agents Translation (NAT) performed on one or more network                                               |
| Vlan Manager                                     | Each tenant is isolated to its own VLANs.                                                                                   | More complex to set up.<br><br>Requires its own DHCP broadcast domain.<br><br>Requires many VLANs to be trunked onto a single port.<br><br>Standard VLAN number limitation.<br><br>Switches must support 802.1q VLAN tagging. | Isolated tenant networks layer 2 traffic between concept, where traffic the VLAN. Isolated include additional services |
| Flat DHCP Multi-host with high availability (HA) | Networking failure is isolated to the VMs running on the affected hypervisor.<br><br>DHCP traffic can be isolated within an | More complex to set up.<br><br>Compute nodes typically need IP addresses accessible by external networks.                                                                                                                     | Configure neutron Network nodes are controller runs networking nodes run vSwitch or                                    |

| Network deployment model | Strengths                                                                    | Weaknesses                                                                                | Neutron equivalent |
|--------------------------|------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|--------------------|
|                          | individual host.<br><br>Network traffic is distributed to the compute nodes. | Options must be carefully configured for live migration to work with networking services. |                    |

Both nova-network and neutron services provide similar capabilities, such as VLAN between VMs. You also can provide multiple NICs on VMs with either service. Further discussion follows.

### VLAN Configuration within OpenStack VMs

VLAN configuration can be as simple or as complicated as desired. The use of VLANs has the benefit of allowing each project its own subnet and broadcast segregation from other projects. To allow OpenStack to efficiently use VLANs, you must allocate a VLAN range (one for each project) and turn each compute node switch port into a trunk port.

For example, if you estimate that your cloud must support a maximum of 100 projects, pick a free VLAN range that your network infrastructure is currently not using (such as VLAN 200–299). You must configure Open- Stack with this range and also configure your switch ports to allow VLAN traffic from that range.

### Multi-NIC Provisioning

OpenStack Compute has the ability to assign multiple NICs to instances on a per-project basis. This is generally an advanced feature and not an everyday request. This can easily be done on a per-request basis, though. However, be aware that a second NIC uses up an entire subnet or VLAN. This decrements your total number of supported projects by one.

### Multi-Host and Single-Host Networking

The nova-network service has the ability to operate in a multi-host or single-host mode. Multi-host is when each compute node runs a copy of nova-network and the instances on that compute node use the compute node as a gateway to the Internet. The compute nodes also host the floating IPs and security groups for instances on that node. Single-host is when a central server—for example, the cloud controller—runs the nova- network service. All compute nodes forward traffic from the instances to the cloud controller. The cloud controller then forwards traffic to the Internet. The cloud controller hosts the floating IPs and security groups for all instances on all compute nodes in the cloud.

There are benefits to both modes. Single-node has the downside of a single point of failure. If the cloud controller is not available, instances cannot communicate on the network. This is not true with multi-host, but multi-host requires that each compute node has a public IP address to communicate on the Internet. If you are not able to obtain a significant block of public IP addresses, multi-host might not be an option.

#### 7.2.5.5 Services for Networking

OpenStack, like any network application, has a number of standard considerations to apply, such as NTP and DNS.

### NTP

Time synchronization is a critical element to ensure continued operation of OpenStack components. Correct time is necessary to avoid errors in instance scheduling, replication of objects in the object store, and even matching log timestamps for debugging.

All servers running OpenStack components should be able to access an appropriate NTP server. You may decide to set up one locally or use the public pools available from the Network Time Protocol project.

### **DNS**

OpenStack does not currently provide DNS services, aside from the dnsmasq daemon, which resides on nova-network hosts. You could consider providing a dynamic DNS service to allow instances to update a DNS entry with new IP addresses. You can also consider making a generic forward and reverse DNS mapping for instances' IP addresses, such as vm-203-0-113-123.example.com.

#### **7.2.5.6 Conclusion**

Armed with your IP address layout and numbers and knowledge about the topologies and services you can use, it's now time to prepare the network for your installation. Be sure to also check out the *OpenStack Security Guide* for tips on securing your network. We wish you a good relationship with your networking team!



**CHAPTER 8**  
**BIG DATA**

## 8. BIG DATA

### 8.1 What Is Big Data?

We shall start this chapter with a definition of “Big Data”, because the term is a bit of a misnomer since it implies that pre-existing data is somehow small (it isn’t) or that the only challenge is its sheer size (size is one of them, but there are often more). In short, the term Big Data applies to information that can’t be processed or analyzed using traditional processes or tools. Increasingly, organizations today are facing more and more Big Data challenges. They have access to a wealth of information, but they don’t know how to get value out of it because it is sitting in its most raw form or in a semi-structured or unstructured format; and as a result, they don’t even know whether it’s worth keeping (or even able to keep it for that matter). An IBM survey found that over half of the business leaders today realize they don’t have access to the insights they need to do their jobs. Companies are facing these challenges in a climate where they have the ability to store anything and they are generating data like never before in history; combined, this presents a real information challenge. It’s a conundrum: today’s business has more access to potential insight than ever before, yet as this potential gold mine of data piles up, the percentage of data the business can process is going down—fast. In this chapter, we shall talk about the characteristics of Big Data and how it fits into the current information management landscape.<sup>1</sup>

Quite simply, the Big Data era is in full force today because the world is changing. Through instrumentation, we’re able to sense more things, and if we can sense it, we tend to try and store it (or at least some of it). Through advances in communications technology, people and things are becoming increasingly interconnected—and not just some of the time, but all of the time. This interconnectivity rate is a runaway train. Generally referred to as machine-to-machine (M2M), interconnectivity is responsible for double-digit year over year data growth rates. Finally, because small integrated circuits are now so inexpensive, we’re able to add intelligence to almost everything.

Even something as mundane as a railway car has hundreds of sensors. On a railway car, these sensors track such things as the conditions experienced by the rail car, the state of individual parts, and GPS-based data for shipment tracking and logistics. After train derailments that claimed extensive losses of life, governments introduced regulations that this kind of data be stored and analyzed to prevent future disasters. Rail cars are also becoming more intelligent: processors have been added to interpret sensor data on parts prone to wear, such as bearings, to identify parts that need repair before they fail and cause further damage—or worse, disaster. But it’s not just the rail cars that are intelligent—the actual rails have sensors every few feet. What’s more, the data storage requirements are for the whole ecosystem: cars, rails, railroad crossing sensors, weather patterns that cause rail movements, and so on. Now add this to tracking a rail car’s cargo load, arrival and departure times, and you can very quickly see you’ve got a Big Data problem on your hands. Even if every bit of this data was relational (and it’s not), it is all going to be raw and have very different formats, which makes processing it in a traditional relational system impractical or impossible. Rail cars are just one example, but everywhere we look, we see domains with velocity, volume, and variety combining to create the Big Data problem.

#### 8.1.1 Characteristics of Big Data

Three characteristics define Big Data: volume, variety, and velocity (as shown in Fig. 8.1). Together, these characteristics define what we refer to as “Big Data.” They have created the need for a new class of capabilities to augment the way things are done today to provide better line of site and controls over our existing knowledge domains and the ability to act on them. Let’s spend some time explicitly defining these terms.

---

This chapter is compiled based on Chapter 1: What is Big Data? Hint: You’re Part of It Every Day of the book Understanding Big Data by Chris Eaton, Dirk Deroos, Tom Deutsch, George Lapis and Paul Zikopoulos

### 8.1.2 Can There Be Enough? The Volume of Data

The sheer volume of data being stored today is exploding. In the year 2000, 800,000 petabytes (PB) of data were stored in the world. Of course, a lot of the data that's being created today isn't analyzed at all and that's another problem we're trying to address with Big Insights. We expect this number to reach 35 zettabytes (ZB) by 2020. Twitter alone generates more than 7 terabytes (TB) of data every day, Facebook 10 TB, and some enterprises generate

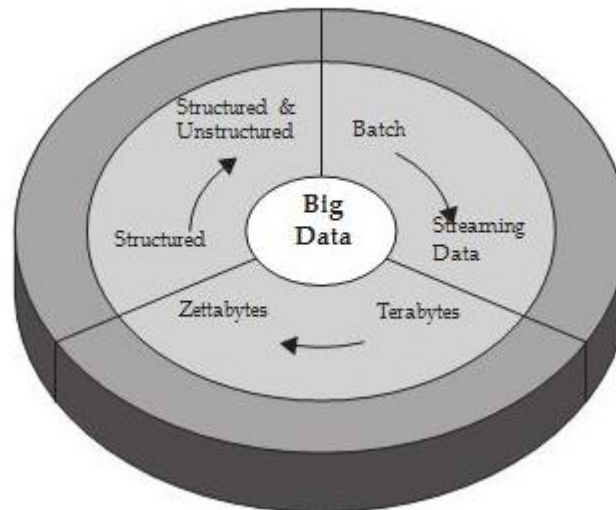


Fig. 8.1: Big Data Characteristics

terabytes of data every hour of every day of the year. It's no longer unheard of for individual enterprises to have storage clusters holding petabytes of data. We're going to stop right there with the factoids: Truth is, these estimates will be out of date by the time you read this book, and they'll be further out of date by the time you bestow your great knowledge of data growth rates on your friends and families when you're done reading this book.

When you stop and think about it, it's little wonder we're drowning in data. If we can track and record something, we typically do. (And notice we didn't mention the analysis of this stored data, which is going to become a theme of Big Data—the newfound utilization of data we track and don't use for decision making.) We store everything: environmental data, financial data, medical data, surveillance data, and the list goes on and on. For example, taking your Smartphone out of your holster generates an event; when your commuter train's door opens for boarding, that's an event; check in for a plane, badge into work, buy a song on iTunes, change the TV channel, take an electronic toll route—everyone of these actions generates data. Need more? The St. Anthony Falls Bridge (which replaced the 2007 collapse of the I-35W Mississippi River Bridge) in Minneapolis has more than 200 embedded sensors positioned at strategic points to provide a fully comprehensive monitoring system where all sorts of detailed data is collected and even a shift in temperature and the bridge's concrete reaction to that change is available for analysis. Okay, you get the point: There's more data than ever before and all you have to do is look at the terabyte penetration rate for personal home computers as the telltale sign. We used to keep a list of all the data warehouses we knew that surpassed a terabyte almost a decade ago—suffice to say, things have changed when it comes to volume.

As implied by the term “Big Data,” organizations are facing massive volumes of data. Organizations that don't know how to manage this data are overwhelmed by it. But the opportunity exists, with the right technology platform, to analyze almost all of the data (or at least more of it by identifying the data that's useful to you) to gain a better understanding of your business, your customers, and the marketplace. And this leads to the current conundrum facing today's businesses across all industries. As the amount of data available to the enterprise is on the rise, the percent of data it can process, understand, and analyze is on the decline, thereby creating the blind zone you see in Fig.8.2. What's in that blind zone. You don't know: it might be something great, or may be nothing at all, but the “don't know” is the problem (or the opportunity, depending how you look at it).



Fig. 8.2: The volume of data available to organizations today is on the rise, while the percent of data they can analyze is on the decline

The conversation about data volumes has changed from terabytes to petabytes with an inevitable shift to zettabytes, and all this data can't be stored in your traditional systems for reasons that we'll discuss in this chapter and others.

### 8.1.3 Variety Is the Spice of Life

The volume associated with the Big Data phenomena brings along new challenges for data centers trying to deal with it: its variety. With the explosion of sensors, and smart devices, as well as social collaboration technologies, data in an enterprise has become complex, because it includes not only traditional relational data, but also raw, semi structured, and unstructured data from web pages, web log files (including click-stream data), search indexes, social media forums, e-mail, documents, sensor data from active and passive systems, and so on. What's more, traditional systems can struggle to store and perform the required analytics to gain understanding from the contents of these logs because much of the information being generated doesn't lend itself to traditional database technologies. In our experience, although some companies are moving down the path, by and large, most are just beginning to understand the opportunities of Big Data (and what's at stake if it's not considered).

Quite simply, variety represents all types of data—a fundamental shift in analysis requirements from traditional structured data to include raw, semi structured, and unstructured data as part of the decision-making and insight process. Traditional analytic platforms can't handle variety. However, an organization's success will rely on its ability to draw insights from the various kinds of data available to it, which includes both traditional and nontraditional.

When we look back at our database careers, sometimes it's humbling to see that we spent more of our time on just 20 percent of the data: the relational kind that's neatly formatted and fits ever so nicely into our strict schemas. But the truth of the matter is that 80 percent of the world's data (and more and more of this data is responsible for setting new velocity and volume records) is unstructured, or semi structured at best. If you look at a Twitter feed, you'll see structure in its JSON format—but the actual text is not structured, and understanding that can be rewarding. Video and picture images aren't easily or efficiently stored in a relational database, certain event information can dynamically change (such as weather patterns), which isn't well suited for strict schemas, and more. To capitalize on the Big Data opportunity, enterprises must be able to analyze all types of data, both relational and non relational: text, sensor data, audio, video, transactional, and more.

#### **8.1.4 How Fast Is Fast? The Velocity of Data**

Just as the sheer volume and variety of data we collect and store has changed, so, too, has the velocity at which it is generated and needs to be handled. A conventional understanding of velocity typically considers how quickly the data is arriving and stored, and its associated rates of retrieval. While managing all of that quickly is good—and the volumes of data that we are looking at are a consequence of how quick the data arrives—we believe the idea of velocity is actually something far more compelling than these conventional definitions.

To accommodate velocity, a new way of thinking about a problem must start at the inception point of the data. Rather than confining the idea of velocity to the growth rates associated with your data repositories, we suggest you apply this definition to data in motion: The speed at which the data is flowing. After all, we're in agreement that today's enterprises are dealing with petabytes of data instead of terabytes, and the increase in RFID sensors and other information streams has led to a constant flow of data at a pace that has made it impossible for traditional systems to handle.

Sometimes, getting an edge over your competition can mean identifying a trend, problem, or opportunity only seconds, or even microseconds, before someone else. In addition, more and more of the data being produced today has a very short shelf-life, so organizations must be able to analyze this data in near real time if they hope to find insights in this data. Big Data scale streams computing is a concept that IBM has been delivering on for some time and serves as a new paradigm for the Big Data problem. In traditional processing, you can think of running queries against relatively static data: for example, the query "Show me all people living in the New Jersey flood zone" would result in a single result set to be used as a warning list of an incoming weather pattern. With streams computing, you can execute a process similar to a continuous query that identifies people who are currently "in the New Jersey flood zones," but you get continuously updated results, because location information from GPS data is refreshed in real time.

Dealing effectively with Big Data requires that you perform analytics against the volume and variety of data while it is still in motion, not just after it is at rest. Consider examples from tracking neonatal health to financial markets; in every case, they require handling the volume and variety of data in new ways.

Now imagine a cohesive Big Data platform that can leverage the best of both worlds and take streaming real-time insight to spawn further research based on emerging data. As you think about this, we're sure you'll start to share the same excitement we have around the unique proposition available with an IBM Big Data platform.

#### **8.1.5 Data in the Warehouse and Data in Hadoop**

In our experience, traditional warehouses are mostly ideal for analyzing structured data from various system and producing insights with known and relatively stable measurements. On the other hand, we feel a Hadoop based platform is well suited to deal with semi structured and unstructured data, as well as when a data discovery process is needed. That isn't to say that Hadoop can't be used for structured data that is readily available in a raw format; because it can, and we talk about that in Chapter 2.

In addition, when you consider where data should be stored, you need to understand how data is stored today and what features characterize your persistence options. Consider your experience with storing data in a traditional data warehouse. Typically, this data goes through a lot of rigor to make it into the warehouse. Builders and consumers of warehouses have it etched in their minds that the data they are looking at in their warehouses must shine with respect to quality; subsequently, it's cleaned up via cleansing, enrichment, matching, glossary, metadata, master data management, modeling, and other services before it's ready for analysis. Obviously, this can be an expensive process. Because of that expense, it's clear that the data that lands in the warehouse is deemed not just of high value, but it has a broad purpose: it's going to go places and will be used in reports and dash-boards where the

accuracy of that data is key. For example, Sarbanes-Oxley (SOX) compliance, introduced in 2002, requires the CEO and CFO of publicly traded companies on U.S.-based exchanges to certify the accuracy of their financial statements (Section 302, “Corporate Responsibility for Financial Reports”). There are serious (we’re talking the potential for jail time here) penalties associated if the data being reported isn’t accurate or “true.” Do you think these folks are going to look at reports of data that aren’t pristine?

In contrast, Big Data repositories rarely undergo (at least initially) the full quality control rigors of data being injected into a warehouse, because not only is prepping data for some of the newer analytic methods characterized by Hadoop use cases cost prohibitive (which we talk about in the next chapter), but the data isn’t likely to be distributed like data warehouse data. We could say that data warehouse data is trusted enough to be “public,” while Hadoop data isn’t as trusted (public can mean vastly distributed within the company and not for external consumption), and although this will likely change in the future, today this is something that experience suggests characterizes these repositories.

Our experiences also suggest that in today’s IT landscape, specific pieces of data have been stored based on their perceived value, and therefore any information beyond those preselected pieces is unavailable. This is in contrast to a Hadoop based repository scheme where the entire business entity is likely to be stored and the fidelity of the Tweet, transaction, Facebook post, and more is kept intact. Data in Hadoop might seem of low value today, or its value non quantified, but it can in fact be the key to questions yet unasked. IT departments pick and choose high-valued data and put it through rigorous cleansing and transformation processes because they know that data has a high known value per byte (a relative phrase, of course). Why else would a company put that data through so many quality control processes? Of course, since the value per byte is high, the business is willing to store it on relatively higher cost infrastructure to enable that interactive, often public, navigation with the end user communities, and the CIO is willing to invest in cleansing the data to increase its value per byte.

With Big Data, you should consider looking at this problem from the opposite view: With all the volume and velocity of today’s data, there’s just no way that you can afford to spend the time and resources required to cleanse and document every piece of data properly, because it’s just not going to be economical. What’s more, how do you know if this Big Data is even valuable? Are you going to go to your CIO and ask her to increase her capital expenditure (CAPEX) and operational expenditure (OPEX) costs by four-fold to quadruple the size of your warehouse on a hunch? For this reason, we like to characterize the initial non analyzed raw Big Data as having a low value per byte, and, therefore, until it’s proven otherwise, you can’t afford to take the path to the warehouse; however, given the vast amount of data, the potential for great insight (and therefore greater competitive advantage in your own market) is quite high if you can analyze all of that data.

At this point, it’s pertinent to introduce the idea of cost per compute, which follows the same pattern as the value per byte ratio. If you consider the focus on the quality data in traditional systems we outlined earlier, you can conclude that the cost per compute in a traditional data warehouse is relatively high (which is fine, because it’s a proven and known higher value per byte), versus the cost of Hadoop, which is low.

Of course, other factors can indicate that certain data might be of high value yet never make its way into the warehouse, or there’s a desire for it to make its way out of the warehouse into a lower cost platform; either way, you might need to cleanse some of that data in Hadoop. For example, unstructured data can’t be easily stored in a warehouse.

Indeed, some warehouses are built with a predefined corpus of questions in mind. Although such a warehouse provides some degree of freedom for query and mining, it could be that it’s constrained by what is in the schema (most unstructured data isn’t found here) and often by a performance envelope that can be a functional/operational hard limit. Again, we are not saying a Hadoop platform such as IBM Info Sphere Big Insights is a replacement for your warehouse : instead , it’s a complement.

A Big Data platform lets you store all of the data in its native business object format and get value out of it through massive parallelism on readily available components. For your interactive navigational needs, you'll continue to pick and choose sources and cleanse that data and keep it in ware-houses. But you can get more value out of analyzing more data (that may even initially seem unrelated) in order to paint a more robust picture of the issue at hand. Indeed, data might sit in Hadoop for a while, and when you discover its value, it might migrate its way into the warehouse when its value is proven and sustainable.

### 8.1.6 Conclusion

We'll conclude this chapter with a gold mining analogy to articulate the points from the previous section and the Big Data opportunity that lies before you. In the "olden days" (which, for some reason, our kids think is a time when we were their age), miners could actually see nuggets or veins of gold; they clearly appreciated the value and would dig and sift near previous gold finds hoping to strike it rich. That said, although there was more gold out there—it could have been in the hill next to them or miles away—it just wasn't visible to the naked eye, and it became a gambling game. You dug like crazy near where gold was found, but you had no idea whether more gold would be found. And although history has its stories of gold rush fevers, nobody mobilized millions of people to dig everywhere and anywhere.

In contrast, today's gold rush works quite differently. Gold mining is executed with massive capital equipment that can process millions of tons of dirt that is worth nothing. Ore grades of 30 mg/kg (30 ppm) are usually needed before gold is visible to the naked eye—that is, most gold in gold mines today is invisible. Although there is all this gold (high-valued data) in all this dirt (low-valued data), by using the right equipment, you can economically process lots of dirt and keep the flakes of gold you find. The flakes of gold are then taken for integration and put together to make a bar of gold, which is stored and logged in a place that's safe, governed, valued, and trusted.

This really is what Big Data is about. You can't afford to sift through all the data that's available to you in your traditional processes; it's just too much data with too little known value and too much of a gambled cost. The Big Data platform gives you a way to economically store and process all that data and find out what's valuable and worth exploiting.

## 8.2 MapReduce and the New Software Stack

Modern data-mining applications, often called "Big Data" analysis, require us to manage immense amounts of data quickly. In many of these applications, the data is extremely regular, and there is ample opportunity to exploit parallelism.

Important examples are:

1. The ranking of Web pages by importance, which involves an iterated matrix-vector multiplication where the dimension is many billions.
2. Searches in "friends" networks at social-networking sites, which involve graphs with hundreds of millions of nodes and many billions of edges.

To deal with applications such as these, a new software stack has evolved. These programming systems are designed to get their parallelism not from a "supercomputer," but from "computing clusters" – large collections of commodity hardware, including conventional processors ("compute nodes") connected by Ethernet cables or inexpensive switches. The software stack begins with a new form of file system, called a "distributed file system," which features much larger units than the disk blocks in a conventional operating system. Distributed file systems also provide replication of data or redundancy to protect against the frequent media failures that occur when data is distributed over thousands of low-cost compute nodes.

On top of these file systems, many different higher-level programming systems have been developed. Central to the new software stack is a programming system called MapReduce. Implementations of MapReduce enable many of the most common calculations on large-scale data to be performed on computing clusters efficiently and in a way that is tolerant of hardware failures during the computation.

MapReduce systems are evolving and extending rapidly. Today, it is common for MapReduce programs to be created from still higher-level programming systems, often an implementation of SQL. Further, MapReduce turns out to be a useful, but simple, case of more general and powerful ideas. We include in this chapter a discussion of generalizations of MapReduce, first to systems that support acyclic workflows and then to systems that implement recursive algorithms.

Our last topic for this chapter is the design of good MapReduce algorithms, a subject that often differs significantly from the matter of designing good parallel algorithms to be run on a supercomputer. When designing MapReduce algorithms, we often find that the greatest cost is in the communication. We thus investigate communication cost and what it tells us about the most efficient MapReduce algorithms. For several common applications of MapReduce we are able to give families of algorithms that optimally trade the communication cost against the degree of parallelism.

### **8.2.1 Distributed File Systems**

Most computing is done on a single processor, with its main memory, cache, and local disk (a compute node). In the past, applications that called for parallel processing, such as large scientific calculations, were done on special-purpose parallel computers with many processors and specialized hardware. However, the prevalence of large-scale Web services has caused more and more computing to be done on installations with thousands of compute nodes operating more or less independently. In these installations, the compute nodes are commodity hardware, which greatly reduces the cost compared with special-purpose parallel machines.

These new computing facilities have given rise to a new generation of programming systems. These systems take advantage of the power of parallelism and at the same time avoid the reliability problems that arise when the computing hardware consists of thousands of independent components, any of which could fail at any time. In this section, we discuss both the characteristics of these computing installations and the specialized file systems that have been developed to take advantage of them.

### **8.2.2 Physical Organization of Compute Nodes**

The new parallel-computing architecture, sometimes called cluster computing, is organized as follows. Compute nodes are stored on racks, perhaps 8–64 on a rack. The nodes on a single rack are connected by a network, typically gigabit Ethernet. There can be many racks of compute nodes, and racks are connected by another level of network or a switch. The bandwidth of inter-rack communication is somewhat greater than the intrarack Ethernet, but given the number of pairs of nodes that might need to communicate between racks, this bandwidth may be essential. Fig. 8.3 suggests the architecture of a large scale computing system. However, there may be many more racks and many more compute nodes per rack.



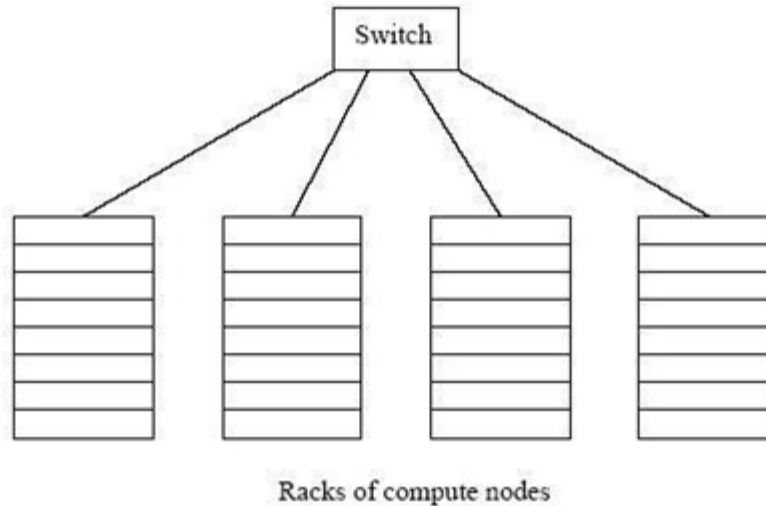


Fig. 8.3: Compute nodes are organized into racks, and racks are interconnected by a switch.

It is a fact of life that components fail, and the more components, such as compute nodes and interconnection networks, a system has, the more frequently something in the system will not be working at any given time. For systems such as Fig. 1, the principal failure modes are the loss of a single node (e.g., the disk at that node crashes) and the loss of an entire rack (e.g., the network connecting its nodes to each other and to the outside world fails). Some important calculations take minutes or even hours on thousands of compute nodes. If we had to abort and restart the computation every time one component failed, then the computation might never complete successfully. The solution to this problem takes two forms:

1. Files must be stored redundantly. If we did not duplicate the file at several compute nodes, then if one node failed, all its files would be unavailable until the node is replaced. If we did not back up the files at all, and the disk crashes, the files would be lost forever. We discuss file management later.
2. Computations must be divided into tasks, such that if any one task fails to execute to completion, it can be restarted without affecting other tasks. This strategy is followed by the MapReduce programming system that we introduce in next section.

### 8.2.3 Large-Scale File-System Organization

To exploit cluster computing, files must look and behave somewhat differently from the conventional file systems found on single computers. This new file system, often called a distributed file system or DFS (although this term has had other meanings in the past), is typically used as follows.

**DFS Implementations**

There are several distributed file systems of the type we have described that are used in practice. Among these:

1. The *Google File System* (GFS), the original of the class.
2. *Hadoop Distributed File System* (HDFS), an open-source DFS used with Hadoop, an implementation of MapReduce (see Section 2.2) and distributed by the Apache Software Foundation.
3. *CloudStore*, an open-source DFS originally developed by Kosmix.]

- Files can be enormous, possibly a terabyte in size. If you have only small files, there is no point using a DFS for them.
- Files are rarely updated. Rather, they are read as data for some calculation, and possibly additional data is appended to files from time to time. For example, an airline reservation system would not be suitable for a DFS, even if the data were very large, because the data is changed so frequently.

Files are divided into chunks, which are typically 64 megabytes in size. Chunks are replicated, perhaps three times, at three different compute nodes. Moreover, the nodes holding copies of one chunk should be located on different racks, so we don't lose all copies due to a rack failure. Normally, both the chunk size and the degree of replication can be decided by the user.

To find the chunks of a file, there is another small file called the master node or name node for that file. The master node is itself replicated, and a directory for the file system as a whole knows where to find its copies. The directory itself can be replicated, and all participants using the DFS know where the directory copies are.

### 8.2.4 MapReduce

MapReduce is a style of computing that has been implemented in several systems, including Google's internal implementation (simply called MapReduce) and the popular open-source implementation Hadoop which can be obtained, along with the HDFS file system from the Apache Foundation. You can use an implementation of MapReduce to manage many large-scale computations in a way that is tolerant of hardware faults. All you need to write are two functions, called Map and Reduce, while the system manages the parallel execution, coordination of tasks that execute Map or Reduce, and also deals with the possibility that one of these tasks will fail to execute. In brief, a MapReduce computation executes as follows:

1. Some number of Map tasks each are given one or more chunks from a distributed file system. These Map tasks turn the chunk into a sequence of key-value pairs. The way key-value pairs are produced from the input data is determined by the code written by the user for the Map function.
2. The key-value pairs from each Map task are collected by a master controller and sorted by key. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task.
3. The Reduce tasks work on one key at a time, and combine all the values associated with that key in some way. The manner of combination of values is determined by the code written by the user for the Reduce function.

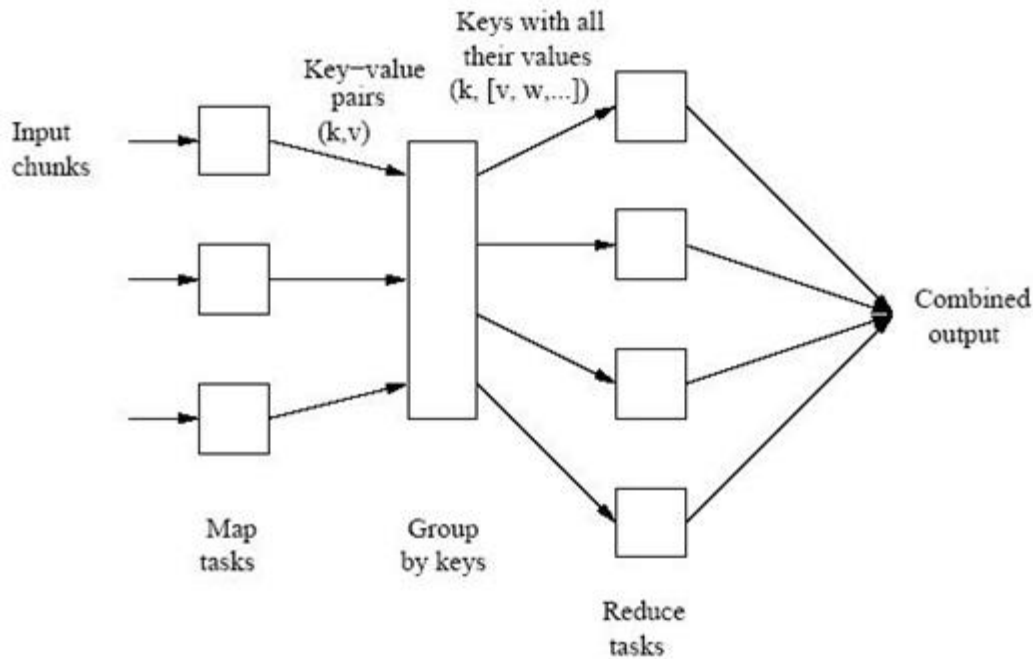


Fig. 8.4: Schematic of a MapReduce computation

#### 8.2.4.1 The Map Tasks

We view input files for a Map task as consisting of elements, which can be any type: a tuple or a document, for example. A chunk is a collection of elements, and no element is stored across two chunks. Technically, all inputs to Map tasks and outputs from Reduce tasks are of the key-value-pair form, but normally the keys of input elements are not relevant and we shall tend to ignore them. Insisting on this form for inputs and outputs is motivated by the desire to allow composition of several MapReduce processes.

The Map function takes an input element as its argument and produces zero or more key-value pairs. The types of keys and values are each arbitrary. Further, keys are not “keys” in the usual sense; they do not have to be unique. Rather a Map task can produce several key-value pairs with the same key, even from the same element.

*Example 1: We shall illustrate a MapReduce computation with what has become the standard example application: counting the number of occurrences for each word in a collection of documents. In this example, the input file is a repository of documents, and each document is an element. The Map function for this example uses keys that are of type String (the words) and values that are integers. The Map task reads a document and breaks it into its sequence of words  $w_1, w_2, \dots, w_n$ . It then emits a sequence of key-value pairs where the value is always 1. That is, the output of the Map task for this document is the sequence of key-value pairs:*

$$(w_1, 1), (w_2, 1), \dots, (w_n, 1)$$

*Note that a single Map task will typically process many documents – all the documents in one or more chunks. Thus, its output will be more than the sequence for the one document suggested above. Note also that if a word  $w$  appears  $m$  times among all the documents assigned to that process, then there will be  $m$  key-value pairs  $(w, 1)$  among its output. An option, which we discuss later, is to combine these  $m$  pairs into a single pair  $(w, m)$ , but we can only do that because, as we shall see, the Reduce tasks apply an associative and commutative operation, addition, to the values.*

#### 8.2.4.2 Grouping by Key

As soon as the Map tasks have all completed successfully, the key-value pairs are grouped by key, and the values associated with each key are formed into a list of values. The grouping is performed by the system, regardless of what the Map and Reduce tasks do. The master controller process knows how many Reduce tasks there will be, say  $r$  such tasks. The user typically tells the MapReduce system what  $r$  should be. Then the master controller picks a hash function that applies to keys and produces a bucket number from 0 to  $r - 1$ . Each key that is output by a Map task is hashed and its key-value pair is put in one of  $r$  local files. Each file is destined for one of the Reduce tasks.

To perform the grouping by key and distribution to the Reduce tasks, the master controller merges the files from each Map task that are destined for a particular Reduce task and feeds the merged file to that process as a sequence of key-list-of-value pairs. That is, for each key  $k$ , the input to the Reduce task that handles key  $k$  is a pair of the form  $(k, [v_1, v_2, \dots, v_n])$ , where  $(k, v_1), (k, v_2), \dots, (k, v_n)$  are all the key-value pairs with key  $k$  coming from all the Map tasks.

#### 8.2.4.3 The Reduce Tasks

The Reduce function's argument is a pair consisting of a key and its list of associated values. The output of the Reduce function is a sequence of zero or more key-value pairs. These key-value pairs can be of a type different from those sent from Map tasks to Reduce tasks, but often they are the same type. We shall refer to the application of the Reduce function to a single key and its associated list of values as a reducer.

A Reduce task receives one or more keys and their associated value lists. That is, a Reduce task executes one or more reducers. The outputs from all the Reduce tasks are merged into a single file. Reducers may be partitioned among a smaller number of Reduce tasks is by hashing the keys and associating each Reduce task with one of the buckets of the hash function.

*Example 2: Let us continue with the word-count example of Example 1. The Reduce function simply adds up all the values. The output of a reducer consists of the word and the sum. Thus, the output of all the Reduce tasks is a sequence of  $(w,m)$  pairs, where  $w$  is a word that appears at least once among all the input documents and  $m$  is the total number of occurrences of  $w$  among all those documents.*

#### 8.2.4.4 Combiners

Sometimes, a Reduce function is associative and commutative. That is, the values to be combined can be combined in any order, with the same result. The addition performed in Example 2 is an example of an associative and commutative operation. It doesn't matter how we group a list of numbers  $v_1, v_2, \dots, v_n$ ; the sum will be the same.

When the Reduce function is associative and commutative, we can push some of what the reducers do to the Map tasks. For example, instead of the Map tasks in Example 1 producing many pairs  $(w, 1), (w, 1), \dots$ , we could apply the Reduce function within the Map task, before the output of the Map tasks is subject to grouping and aggregation. These key-value pairs would thus be replaced by one pair with key  $w$  and value equal to the sum of all the 1's in all those pairs. That is, the pairs with key  $w$  generated by a single Map task would be replaced by a pair  $(w,m)$ , where  $m$  is the number of times that  $w$  appears among the documents handled by this Map task. Note that it is still necessary to do grouping and aggregation and to pass the result to the Reduce tasks, since there will typically be one key-value pair with key  $w$  coming from each of the Map tasks.

#### 8.2.5 Details of MapReduce Execution

Let us now consider in more detail how a program using MapReduce is executed. Fig. 3 offers an outline of how processes, tasks, and files interact. Taking advantage of a library provided by a MapReduce system such as Hadoop, the user program forks a Master Controller process and some number of Worker processes at different compute nodes. Normally, a Worker handles either Map tasks (a Map worker) or Reduce tasks (a Reduce worker), but not both.

The Master has many responsibilities. One is to create some number of Map tasks and some number of Reduce tasks, these numbers being selected by the user program. These tasks will be assigned to Worker processes by the Master. It is reasonable to create one Map task for every chunk of the input file(s), but we may wish to create fewer Reduce tasks. The reason for limiting the number of Reduce tasks is that it is necessary for each Map task to create an intermediate file for each Reduce task, and if there are too many Reduce tasks the number of intermediate files explodes.

The Master keeps track of the status of each Map and Reduce task (idle,

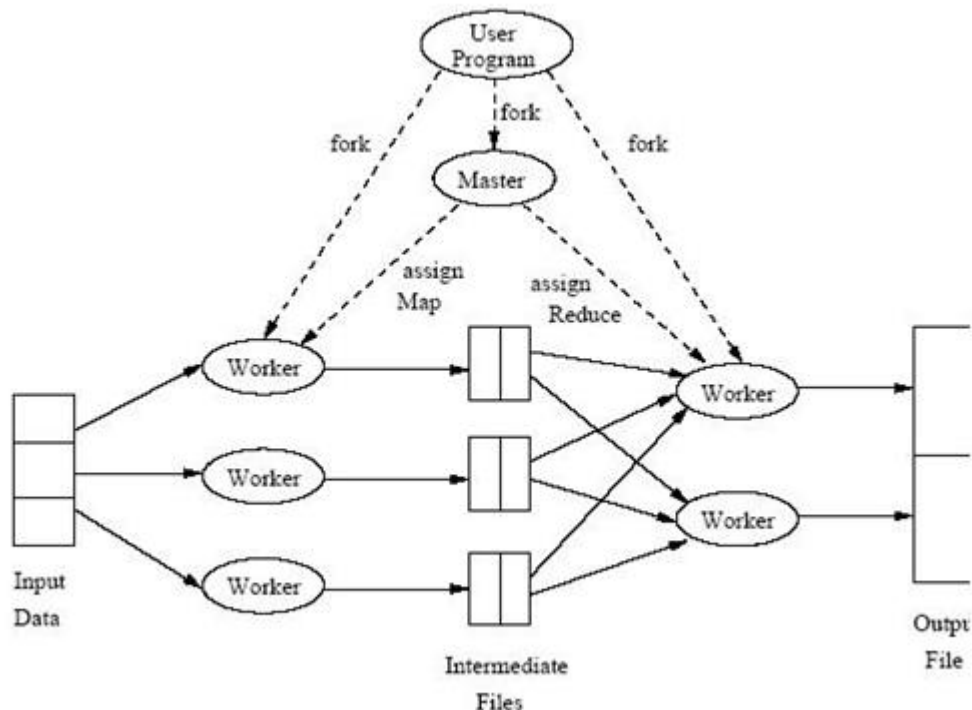


Fig. 8.5: Overview of the execution of a MapReduce program

executing at a particular Worker, or completed). A Worker process reports to the Master when it finishes a task, and a new task is scheduled by the Master for that Worker process.

Each Map task is assigned one or more chunks of the input file(s) and executes on it the code written by the user. The Map task creates a file for each Reduce task on the local disk of the Worker that executes the Map task. The Master is informed of the location and sizes of each of these files, and the Reduce task for which each is destined. When a Reduce task is assigned by the Master to a Worker process, that task is given all the files that form its input. The Reduce task executes code written by the user and writes its output to a file that is part of the surrounding distributed file system.

### 8.2.6 Coping with Node Failures

The worst thing that can happen is that the compute node at which the Master is executing fails. In this case, the entire MapReduce job must be restarted. But only this one node can bring the entire process down; other failures will be managed by the Master, and the MapReduce job will complete eventually.

Suppose the compute node at which a Map worker resides fails. This failure will be detected by the Master, because it periodically pings the Worker processes. All the Map tasks that were assigned to this Worker will have to be redone, even if they had completed. The reason for redoing completed Map tasks is that their output destined for the Reduce tasks resides at that compute node, and is now unavailable to the Reduce tasks. The Master sets the status of each of these Map tasks to idle and will schedule them on a Worker when one becomes available. The Master must also inform each Reduce task that the location of its input from that Map task has changed.

Dealing with a failure at the node of a Reduce worker is simpler. The Master simply sets the status of its currently executing Reduce tasks to idle. These will be rescheduled on another reduce worker later.

### 8.2.7 Algorithms Using MapReduce

MapReduce is not a solution to every problem, not even every problem that profitably can use many compute nodes operating in parallel. As we mentioned in previous sections, the entire distributed-file-system milieu makes sense only when files are very large and are rarely updated in place. Thus, we would not expect to use either a DFS or an implementation of MapReduce for managing online retail sales, even though a large on-line retailer such as Amazon.com uses thousands of compute nodes when processing requests over the Web. The reason is that the principal operations on Amazon data involve responding to searches for products, recording sales, and so on, processes that involve relatively little calculation and that change the database.<sup>2</sup> On the other hand, Amazon might use MapReduce to perform certain analytic queries on large amounts of data, such as finding for each user those users whose buying patterns were most similar.

The original purpose for which the Google implementation of MapReduce was created was to execute very large matrix-vector multiplications as are needed in the calculation of Page Rank. Another important class of operations that can use MapReduce effectively are the relational-algebra operations. We shall examine the MapReduce execution of these operations as well.

### 8.2.8 Relational-Algebra Operations

There are a number of operations on large-scale data that are used in database queries. Many traditional database applications involve retrieval of small amounts of data, even though the database itself may be large. For example, a query may ask for the bank balance of one particular account. Such queries are not useful applications of MapReduce.

However, there are many operations on data that can be described easily in terms of the common database-query primitives, even if the queries themselves are not executed within a database management system. Thus, a good starting point for exploring applications of MapReduce is by considering the standard operations on relations. We assume you are familiar with database systems, the query language SQL, and the relational model, but to review, a relation is a table with column headers called attributes. Rows of the relation are called tuples. The set of attributes of a relation is called its schema. We often write an expression like  $R(A_1, A_2, \dots, A_n)$  to say that the relation name is  $R$  and its attributes are  $A_1, A_2, \dots, A_n$ .

| <i>From</i> | <i>To</i> |
|-------------|-----------|
| url1        | url2      |
| url1        | url3      |
| url2        | url3      |
| url2        | url4      |
| ...         | ...       |

Fig. 8.6: Relation Links consists of the set of pairs of URL's, such that the first has one or more links to the second.

*Example 3 :* In Fig. 4 we see part of the relation Links that describes the structure of the Web. There are two attributes, *From* and *To*. A row, or tuple, of the relation is a pair of URL's, such that there is at least one link from the first URL to the second. For instance, the first row of Fig. 8.6 is the pair (url1, url2) that says the Web page url1 has a link to page url2. While we have shown only four

tuples, the real relation of the Web, or the portion of it that would be stored by a typical search engine, has billions of tuples.

A relation, however large, can be stored as a file in a distributed file system. The elements of this file are the tuples of the relation.

There are several standard operations on relations, often referred to as relational algebra, that are used to implement queries. The queries themselves usually are written in SQL. The relational-algebra operations we shall discuss are:

1. **Selection:** Apply a condition  $C$  to each tuple in the relation and produce as output only those tuples that satisfy  $C$ . The result of this selection is denoted  $\sigma_C(R)$ .
2. **Projection:** For some subset  $S$  of the attributes of the relation, produce from each tuple only the components for the attributes in  $S$ . The result of this projection is denoted  $\pi_S(R)$ .
3. **Union, Intersection, and Difference:** These well-known set operations apply to the sets of tuples in two relations that have the same schema. There are also bag (multiset) versions of the operations in SQL, with somewhat unintuitive definitions, but we shall not go into the bag versions of these operations here.
4. **Natural Join:** Given two relations, compare each pair of tuples, one from each relation. If the tuples agree on all the attributes that are common to the two schemas, then produce a tuple that has components for each of the attributes in either schema and agrees with the two tuples on each attribute. If the tuples disagree on one or more shared attributes, then produce nothing from this pair of tuples. The natural join of relations  $R$  and  $S$  is denoted  $R \bowtie S$ . While we shall discuss executing only the natural join with MapReduce, all equijoins (joins where the tuple-agreement condition involves equality of attributes from the two relations that do not necessarily have the same name) can be executed in the same manner. We shall give an illustration in Example 4.
5. **Grouping and Aggregation:** Given a relation  $R$ , partition its tuples according to their values in one set of attributes  $G$ , called the grouping attributes. Then, for each group, aggregate the values in certain other attributes. The normally permitted aggregations are SUM, COUNT, AVG, MIN, and MAX, with the obvious meanings. Note that MIN and MAX require that the aggregated attributes have a type that can be compared, e.g., numbers or strings, while SUM and AVG require that the type allow arithmetic operations. We denote a grouping-and-aggregation operation on a relation  $R$  by  $\gamma_X(R)$ , where  $X$  is a list of elements that are either
  - (a) A grouping attribute, or
  - (b) An expression  $\theta(A)$ , where  $\theta$  is one of the five aggregation operations such as SUM, and  $A$  is an attribute not among the group in attributes.

The result of this operation is one tuple for each group. That tuple has a component for each of the grouping attributes, with the value common to tuples of that group. It also has a component for each aggregation, with the aggregated value for that group. We shall see an illustration in Example 5.

**Example 4 :** Let us try to find the paths of length two in the Web, using the relation *Links* of Fig. 4. That is, we want to find the triples of URL's  $(u, v, w)$  such that there is a link from  $u$  to  $v$  and a link from  $v$  to  $w$ . We essentially want to take the natural join of *Links* with itself, but we first need to imagine that it is two relations, with different schemas, so we can describe the desired connection as a natural join. Thus, imagine that there are two copies of *Links*, namely  $L1(U1, U2)$  and  $L2(U2, U3)$ . Now, if we compute  $L1 \bowtie L2$ , we shall have exactly what we want. That is, for each tuple  $t1$  of  $L1$  (i.e., each tuple of *Links*) and each tuple  $t2$  of  $L2$  (another tuple of *Links*, possibly even the same tuple), see if their  $U2$  components are the same. Note that these components are the second component of  $t1$  and the first component of  $t2$ . If these two components agree, then produce a tuple for the result, with schema  $(U1, U2, U3)$ . This tuple consists of the first component of  $t1$ , the second component of  $t1$  (which must equal the first component of  $t2$ ), and the second component of  $t2$ .

We may not want the entire path of length two, but only want the pairs  $(u,w)$  of URL's such that there is at least one path from  $u$  to  $w$  of length two. If so, we can project out the middle components by computing  $\pi_{U1, U3}(L1 \triangleleft L2)$ .

**Example 5 :** Imagine that a social-networking site has a relation

*Friends (User, Friend)*

This relation has tuples that are pairs  $(a, b)$  such that  $b$  is a friend of  $a$ . The site might want to develop statistics about the number of friends members have. Their first step would be to compute a count of the number of friends of each user. This operation can be done by grouping and aggregation, specifically

$\gamma_{User, COUNT(Friend)}(Friends)$

This operation groups all the tuples by the value in their first component, so there is one group for each user. Then, for each group the count of the number of friends of that user is made. The result will be one tuple for each group, and a typical tuple would look like  $(Sally, 300)$ , if user "Sally" has 300 friends.

### 8.2.9 Computing Selections by MapReduce

Selections really do not need the full power of MapReduce. They can be done most conveniently in the map portion alone, although they could also be done in the reduce portion alone. Here is a MapReduce implementation of selection  $\sigma_C(R)$ .

**The Map Function:** For each tuple  $t$  in  $R$ , test if it satisfies  $C$ . If so, produce the key-value pair  $(t, t)$ . That is, both the key and value are  $t$ .

**The Reduce Function:** The Reduce function is the identity. It simply passes each key-value pair to the output.

Note that the output is not exactly a relation, because it has key-value pairs. However, a relation can be obtained by using only the value components (or only the key components) of the output.

### 8.2.10 Computing Projections by MapReduce

Projection is performed similarly to selection, because projection may cause the same tuple to appear several times, the Reduce function must eliminate duplicates. We may compute  $\pi_S(R)$  as follows.

**The Map Function:** For each tuple  $t$  in  $R$ , construct a tuple  $t'$  by eliminating from  $t$  those components whose attributes are not in  $S$ . Output the key-value pair  $(t', t')$ .

**The Reduce Function:** For each key  $t'$  produced by any of the Map tasks, there will be one or more key-value pairs  $(t', t')$ . The Reduce function turns  $(t', [t', t', \dots, t'])$  into  $(t', t')$ , so it produces exactly one pair  $(t', t')$  for this key  $t'$ .

Observe that the Reduce operation is duplicate elimination. This operation is associative and commutative, so a combiner associated with each Map task can eliminate whatever duplicates are produced locally. However, the Reduce tasks are still needed to eliminate two identical tuples coming from different Map tasks.



### 8.2.11 Union, Intersection, and Difference by MapReduce

First, consider the union of two relations. Suppose relations  $R$  and  $S$  have the same schema. Map tasks will be assigned chunks from either  $R$  or  $S$ ; it doesn't matter which. The Map tasks don't really do anything except pass their input tuples as key-value pairs to the Reduce tasks. The latter need only eliminate duplicates as for projection.

**The Map Function:** Turn each input tuple  $t$  into a key-value pair  $(t, t)$ .

**The Reduce Function:** Associated with each key  $t$  there will be either one or two values. Produce output  $(t, t)$  in either case.

To compute the intersection, we can use the same Map function. However, the Reduce function must produce a tuple only if both relations have the tuple. If the key  $t$  has a list of two values  $[t, t]$  associated with it, then the Reduce task for  $t$  should produce  $(t, t)$ . However, if the value-list associated with key  $t$  is just  $[t]$ , then one of  $R$  and  $S$  is missing  $t$ , so we don't want to produce a tuple for the intersection.

**The Map Function:** Turn each tuple  $t$  into a key-value pair  $(t, t)$ .

**The Reduce Function:** If key  $t$  has value list  $[t, t]$ , then produce  $(t, t)$ . Otherwise, produce nothing. The Difference  $R - S$  requires a bit more thought. The only way a tuple  $t$  can appear in the output is if it is in  $R$  but not in  $S$ . The Map function can pass tuples from  $R$  and  $S$  through, but must inform the Reduce function whether the tuple came from  $R$  or  $S$ . We shall thus use the relation as the value associated with the key  $t$ . Here is a specification for the two functions.

**The Map Function:** For a tuple  $t$  in  $R$ , produce key-value pair  $(t, R)$ , and for a tuple  $t$  in  $S$ , produce key-value pair  $(t, S)$ . Note that the intent is that the value is the name of  $R$  or  $S$  (or better, a single bit indicating whether the relation is  $R$  or  $S$ ), not the entire relation.

**The Reduce Function:** For each key  $t$ , if the associated value list is  $[R]$ , then produce  $(t, t)$ . Otherwise, produce nothing.

### 8.2.12 Computing Natural Join by MapReduce

The idea behind implementing natural join via MapReduce can be seen if we look at the specific case of joining  $R(A, B)$  with  $S(B, C)$ . We must find tuples that agree on their  $B$  components, that is the second component from tuples of  $R$  and the first component of tuples of  $S$ . We shall use the  $B$ -value of tuples from either relation as the key. The value will be the other component and the name of the relation, so the Reduce function can know where each tuple came from.

**The Map Function:** For each tuple  $(a, b)$  of  $R$ , produce the key-value pair  $(b, (R, a))$ . For each tuple  $(b, c)$  of  $S$ , produce the key-value pair  $(b, (S, c))$ .

**The Reduce Function:** Each key value  $b$  will be associated with a list of pairs that are either of the form  $(R, a)$  or  $(S, c)$ . Construct all pairs consisting of one with first component  $R$  and the other with first component  $S$ , say  $(R, a)$  and  $(S, c)$ . The output from this key and value list is a sequence of key-value pairs. The key is irrelevant. Each value is one of the triples  $(a, b, c)$  such that  $(R, a)$  and  $(S, c)$  are on the input list of values.

The same algorithm works if the relations have more than two attributes. You can think of  $A$  as representing all those attributes in the schema of  $R$  but not  $S$ .  $B$  represents the attributes in both schemas, and  $C$  represents attributes only in the schema of  $S$ . The key for a tuple of  $R$  or  $S$  is the list of values in all the attributes that are in the schemas of both  $R$  and  $S$ . The value for a tuple of  $R$  is the name  $R$  together with the values of all the attributes belonging to  $R$  but not to  $S$ , and the value for a tuple of  $S$  is the name  $S$  together with the values of the attributes belonging to  $S$  but not  $R$ .

The Reduce function looks at all the key-value pairs with a given key and combines those values from R with those values of S in all possible ways. From each pairing, the tuple produced has the values from R, the key values, and the values from S.

### 8.2.13 Grouping and Aggregation by MapReduce

As with the join, we shall discuss the minimal example of grouping and aggregation, where there is one grouping attribute and one aggregation. Let  $R(A,B,C)$  be a relation to which we apply the operator  $\gamma_{A,\theta(B)}(R)$ . Map will perform the grouping, while Reduce does the aggregation.

**The Map Function:** For each tuple  $(a, b, c)$  produce the key-value pair  $(a, b)$ .

**The Reduce Function:** Each key  $a$  represents a group. Apply the aggregation operator  $\theta$  to the list  $[b_1, b_2, \dots, b_n]$  of B-values associated with key  $a$ . The output is the pair  $(a, x)$ , where  $x$  is the result of applying  $\theta$  to the list. For example, if  $\theta$  is SUM, then  $x = b_1 + b_2 + \dots + b_n$ , and if  $\theta$  is MAX, then  $x$  is the largest of  $b_1, b_2, \dots, b_n$ .

If there are several grouping attributes, then the key is the list of the values of a tuple for all these attributes. If there is more than one aggregation, then the Reduce function applies each of them to the list of values associated with a given key and produces a tuple consisting of the key, including components for all grouping attributes if there is more than one, followed by the results of each of the aggregations.

## 8.3 Hadoop Framework and Big Data

Lets start with an example. Say we need to store lots of photos. We will start with a single disk. When we exceed a single disk, we may use a few disks stacked on a machine. When we max out all the disks on a single machine, we need to get a bunch of machines, each with a bunch of disks. This is exactly how Hadoop is built. Hadoop is designed to run on a cluster of machines from the get go.

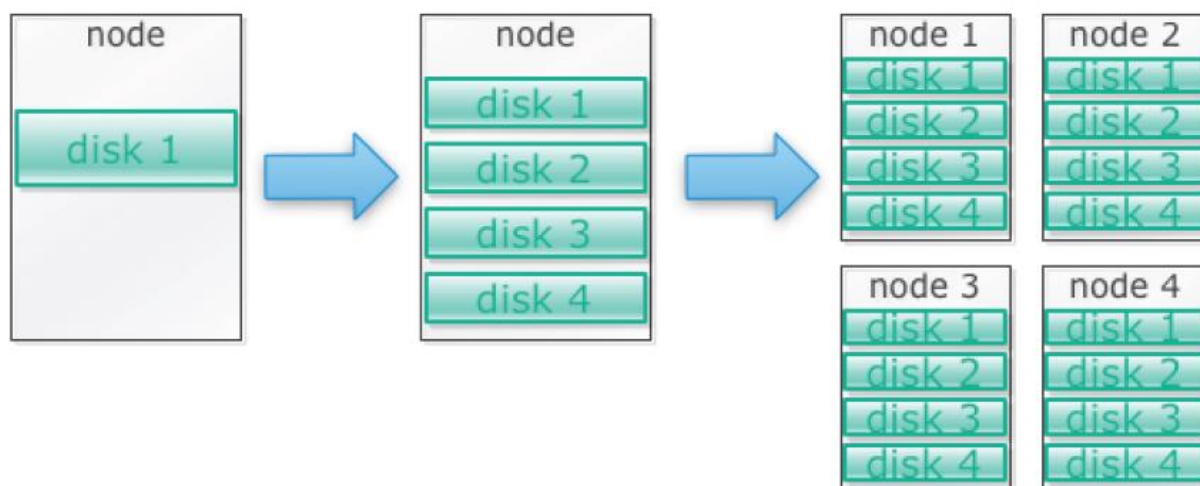


Fig. 8.7: Scaling Storage

### 8.3.1 Properties of Hadoop

- **Hadoop clusters scale horizontally:** More storage and compute power can be achieved by adding more nodes to a Hadoop cluster. This eliminates the need to buy more and more powerful and expensive hardware.
- **Hadoop can handle unstructured / semi-structured data:** Hadoop doesn't enforce a 'schema' on the data it stores. It can handle arbitrary text and binary data. So Hadoop can 'digest' any unstructured data easily.

- **Hadoop clusters provides storage and computing:** We saw how having separate storage and processing clusters is not the best fit for Big Data. Hadoop clusters provide storage and distributed computing all in one.

### 8.3.2 Business Case for Hadoop

- **Hadoop provides storage for Big Data at reasonable cost:** Storing Big Data using traditional storage can be expensive. Hadoop is built around commodity hardware. Hence it can provide fairly large storage for a reasonable cost. Hadoop has been used in the field at Petabyte scale. One study by Cloudera suggested that Enterprises usually spend around \$25,000 to \$50,000 dollars per tera byte per year. With Hadoop this cost drops to few thousands of dollars per tera byte per year. And hardware gets cheaper and cheaper this cost continues to drop.
- **Hadoop allows to capture new or more data:** Some times organizations don't capture a type of data, because it was too cost prohibitive to store it. Since Hadoop provides storage at reasonable cost, this type of data can be captured and stored. Hadoop and Big Data One example would be web site click logs. Because the volume of these logs can be very high, not many organizations captured these. Now with Hadoop it is possible to capture and store the logs
- **With Hadoop, you can store data longer:** To manage the volume of data stored, companies periodically purge older data. For example only logs for the last 3 months could be stored and older logs were deleted. With Hadoop it is possible to store the historical data longer. This allows new analytics to be done on older historical data. For example, take click logs from a web site. Few years ago, these logs were stored for a brief period of time to calculate statics like popular pages ..etc. Now with Hadoop it is viable to store these click logs for longer period of time.
- **Hadoop provides scalable analytics:** There is no point in storing all the data, if we can't analyze them. Hadoop not only provides distributed storage, but also distributed processing as well. Meaning we can crunch a large volume of data in parallel. The compute framework of Hadoop is called Map Reduce. Map Reduce has been proven to the scale of peta bytes.
- **Hadoop provides rich analytics:** Native Map Reduce supports Java as primary programming language. Other languages like Ruby, Python and R can be used as well. Of course writing custom Map Reduce code is not the only way to analyze data in Hadoop. Higher level Map Reduce is available. For example a tool named Pig takes english like data flow language and translates them into Map Reduce. Another tool Hive, takes SQL queries and runs them using Map Reduce. Business Intelligence (BI) tools can provide even higher level of analysis. Quite a few BI tools can work with Hadoop and analyze data stored in Hadoop.

### 8.3.3 Hadoop = HDFS + MapReduce

Hadoop provides two things : Storage & Compute. If you think of Hadoop as a coin, one side is storage and other side is compute.



**Fig. 8.8: Hadoop Coin**

In Hadoop speak, storage is provided by *Hadoop Distributed File System (HDFS)*. Compute is provided by *MapReduce*. *Hadoop* is an open-source implementation of Google's distributed computing framework (which is proprietary). It consists of two parts: Hadoop Distributed File System (HDFS), which is modeled after Google's GFS, and Hadoop MapReduce, which is modeled after Google's MapReduce.

*MapReduce* is a programming framework. Its description was published by Google in 2004 [<http://research.google.com/archive/mapreduce.html>]. Much like other frameworks, such as Spring, Struts, or MFC, the MapReduce framework does some things for you, and provides a place for you to fill in the blanks. What MapReduce does for you is to organize your multiple computers in a cluster in order to perform the calculations you need. It takes care of distributing the work between computers and of putting together the results of each computer's computation. Just as important, it takes care of hardware and network failures, so that they do not affect the flow of your computation. You, in turn, have to break your problem into separate pieces which can be processed in parallel by multiple machines, and you provide the code to do the actual calculation.

#### **8.3.4 Why Hadoop?**

We have already mentioned that the Hadoop is used at Yahoo and Facebook. It has seen rapid uptake in finance, retail, telco, and the government. It is making inroads into life sciences. Why is this? The short answer is that it simplifies dealing with Big Data. This answer immediately resonates with people, it is clear and succinct, but it is not complete. The Hadoop framework has built-in power and flexibility to do what you could not do before. In fact, Cloudera presentations at the latest O'Reilly Strata conference mentioned that MapReduce was initially used at Google and Facebook not primarily for its scalability, but for what it allowed you to do with the data.

In 2010, the average size of Cloudera's customers' clusters was 30 machines. In 2011 it was 70. When people start using Hadoop, they do it for many reasons, all concentrated around the new ways of dealing with the data. What gives them the security to go ahead is the knowledge that Hadoop

solutions are massively scalable, as has been proved by Hadoop running in the world's largest computer centers and at the largest companies.

As you will discover, the Hadoop framework organizes the data and the computations, and then runs your code. At times, it makes sense to run your solution, expressed in a MapReduce paradigm, even on a single machine. But of course, Hadoop really shines when you have not one, but rather tens, hundreds, or thousands of computers. If your data or computations are significant enough (and whose aren't these days?), then you need more than one machine to do the number crunching. If you try to organize the work yourself, you will soon discover that you have to coordinate the work of many computers, handle failures, retries, and collect the results together, and so on. Enter Hadoop to solve all these problems for you. Now that you have a hammer, everything becomes a nail: people will often reformulate their problem in MapReduce terms, rather than create a new custom computation platform.

No less important than Hadoop itself are its many friends. The Hadoop Distributed File System (HDFS) provides unlimited file space available from any Hadoop node. HBase is a high-performance unlimited-size database working on top of Hadoop. If you need the power of familiar SQL over your large data sets, Pig provides you with an answer. While Hadoop can be used by programmers and taught to students as an introduction to Big Data, its companion projects (including ZooKeeper, about which we will hear later on) will make projects possible and simplify them by providing tried-and-proven frameworks for every aspect of dealing with large data sets.

As you learn the concepts, and perfect your skills with the techniques described in this book you will discover that there are many cases where Hadoop storage, Hadoop computation, or Hadoop's friends can help you. Let's look at some of these situations.

- Do you find yourself often cleaning the limited hard drives in your company? Do you need to transfer data from one drive to another, as a backup? Many people are so used to this necessity, that they consider it an unpleasant but unavoidable part of life. Hadoop distributed file system, HDFS, grows by adding servers. To you it looks like one hard drive. It is self-replicating (you set the replication factor) and thus provides redundancy as a software alternative to RAID.
- Do your computations take an unacceptably long time? Are you forced to give up on projects because you don't know how to easily distribute the computations between multiple computers? MapReduce helps you solve these problems. What if you don't have the hardware to run the cluster? Amazon EC2 can run MapReduce jobs for you, and you pay only for the time that it runs - the cluster is automatically formed for you and then disbanded.
- But say you are lucky, and instead of maintaining legacy software, you are charged with building new, progressive software for your company's work flow. Of course, you want to have unlimited storage, solving this problem once and for all, so as to concentrate on what's really important. The answer is: you can mount HDFS as a FUSE file system, and you have your unlimited storage. In our cases studies we look at the successful use of HDFS as a grid storage for the Large Hadron Collider.
- Imagine you have multiple clients using your on line resources, computations, or data. Each single use is saved in a log, and you need to generate a summary of use of resources for each client by day or by hour. From this you will do your invoices, so it IS important. But the data set is large. You can write a quick MapReduce job for that. Better yet, you can use Hive, a data warehouse infrastructure built on top of Hadoop, with its ETL capabilities, to generate your invoices in no time. We'll talk about Hive later, but we hope that you already see that you can use Hadoop and friends for fun and profit. Once you start thinking without the usual limitations, you can improve on what you already do and come up with new and useful projects.

### 8.3.5 HDFS - Hadoop Distributed File System

HDFS, or the Hadoop Distributed File System, gives the programmer unlimited storage (fulfilling a cherished dream for programmers). However, here are additional advantages of HDFS.

- **Horizontal scalability:** Thousands of servers holding petabytes of data. When you need even more storage, you don't switch to more expensive solutions, but add servers instead.
- **Commodity hardware:** HDFS is designed with relatively cheap commodity hardware in mind. HDFS is self-healing and replicating.
- **Fault tolerance:** Every member of the Hadoop zoo knows how to deal with hardware failures. If you have 10 thousand servers, then you will see one server fail every day, on average. HDFS foresees that by replicating the data, by default three times, on different data node servers. Thus, if one data node fails, the other two can be used to restore the third one in a different place. HDFS implementation is modeled after GFS, Google Distributed File system, thus you can read the first paper on this, to be found here: <http://labs.google.com/papers/gfs.html>.

### 8.3.6 MapReduce

MapReduce takes care of distributed computing. It reads the data, usually from its storage, the Hadoop Distributed File System (HDFS), in an optimal way. However, it can read the data from other places too, including mounted local file systems, the web, and databases. It divides the computations between different computers (servers, or nodes). It is also fault-tolerant.

If some of your nodes fail, Hadoop knows how to continue with the computation, by re-assigning the incomplete work to another node and cleaning up after the node that could not complete its task. It also knows how to combine the results of the computation in one place.

### 8.3.7 HBase, the database for Big Data

HBase is a database for Big Data, up to millions of columns and billions of rows. Another feature of HBase is that it is a key-value database, not a relational database. We will get into the pros and cons of these two approaches to databases later, but for now let's just note that key-value databases are considered as more fitting for Big Data. Why? Because they don't store nulls! This gives them the appellation of "sparse," and as we saw above, Tao Te Chin says that they are useful for this reason.

### 8.3.8 ZooKeeper

The ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. ZooKeeper is also fault-tolerant. In your development environment, you can put the zookeeper on one node, but in production you usually run it on an odd number of servers, such as 3 or 5.

### 8.3.9 Hive - data warehousing

Hive is a way for you to get all the honey, and to leave all the work to the bees. You can do a lot of data analysis with Hadoop, but you will also have to write MapReduce tasks. Hive takes that task upon itself. Hive defines a simple SQL-like query language, called QL, that enables users familiar with SQL to query the data.

At the same time, if your Hive program does almost what you need, but not quite, you can call on your MapReduce skill. Hive allows you to write custom mappers and reducers to extend the QL capabilities.

### 8.3.10 Pig - Big Data manipulation

Pig is called pig not because it eats a lot, although you can imagine a pig pushing around and consuming big volumes of information. Rather, it is called pig because it speaks Pig Latin. Others who also speak this language are the kids (the programmers) who visit the Hadoop zoo.

So what is Pig Latin that Apache Pig speaks? As a rough analogy, if Hive is the SQL of Big Data, then Pig Latin is the language of the stored procedures of Big Data. It allows you to manipulate large volumes of information, analyze them, and create new derivative data sets. Internally it creates a sequence of MapReduce jobs, and thus you, the programmer-kid, can use this simple language to solve pretty sophisticated large-scale problems.

### **8.3.11 Hadoop alternatives**

Now that we have met the Hadoop zoo, we are ready to start our excursion. Only one thing stops us at this point - and that is, a gnawing doubt, are we in the right zoo? Let us look at some alternatives to dealing with Big Data. Granted, our concentration here is Hadoop, and we may not give justice to all the other approaches. But we will try.

### **8.3.12 Large data storage alternatives**

HDFS is not the only, and in fact, not the earliest or the latest distributed file system. CEPH claims to be more flexible and to remove the limit on the number of files. HDFS stores all of its file information in the memory of the server which is called the NameNode. This is its strong point - speed - but it is also its Achilles' heel! CEPH, on the other hand, makes the function of the NameNode completely distributed.

Another possible contender is ZFS, an open-source file system from SUN, and currently Oracle. Intended as a complete redesign of file system thinking, ZFS holds a strong promise of unlimited size, robustness, encryption, and many other desirable qualities built into the low-level file system. After all, HDFS and its role model GFS both build on a conventional file system, creating their improvement on top of it, and the premise of ZFS is that the underlying file system should be redesigned to address the core issues.

I have seen production architectures built on ZFS, where the data storage requirements were very clear and well-defined and where storing data from multiple field sensors was considered better done with ZFS.

The pros for ZFS in this case were: built-in replication, low overhead, and - given the right structure of records when written - built-in indexing for searching. Obviously, this was a very specific, though very fitting solution. While other file system start out with the goal of improving on HDFS/GFS design, the advantage of HDFS is that it is very widely used. I think that in evaluating other file systems, the reader can be guided by the same considerations that led to the design of GFS: its designers analyzed prevalent file usage in the majority of their applications, and created a file system that optimized reliability for that particular type of usage. The reader may be well advised to compare the assumptions of GFS designers with his or her case, and decide if HDFS fits the purpose, or if something else should be used in its place.

We should also note here that we compared Hadoop to other open-source storage solutions. There are proprietary and commercial solutions, but such comparison goes beyond the scope of this introduction.

### **8.3.13 Large database alternatives**

The closest to HBase is Cassandra. While HBase is a near-clone of Google's Big Table, Cassandra purports to being a "Big Table/Dynamo hybrid". It can be said that while Cassandra's "writes-never-fail" emphasis has its advantages, HBase is the more robust database for a majority of use-cases. HBase being more prevalent in use, Cassandra faces an uphill battle - but it may be just what you need.

Hypertable is another database close to Google's Big Table in features, and it claims to run 10 times faster than HBase. There is an ongoing discussion between HBase and Hypertable proponents, and the authors do not want to take sides in it, leaving the comparison to the reader. Like Cassandra, Hypertable has fewer users than HBase, and here too, the reader needs to evaluate the speed of Hypertable for his application, and weigh it with other factors.

MongoDB (from "humongous") is a scalable, high-performance, open source, document-oriented database. Written in C++, MongoDB features document-oriented storage, full index on any attribute, replication and high availability, rich, document-based queries, and it works with MapReduce. If you are specifically processing documents and not arbitrary data, it is worth a look.

Other open-source and commercial databases that may be given consideration include Vertica with its SQL support and visualization, Cloudera for OLTP, and Spire.

In the end, before embarking on a development project, you will need to compare alternatives. Below is an example of such comparison. Please keep in mind that this is just one possible point of view, and that the specifics of your project and of your view will be different. Therefore, the table 8.1 below is mainly to encourage the reader to do a similar evaluation for his own needs.

**Table.8.1 Comparison of Big Data**

| DB Pros/Cons | HBase                                                    | Cassandra                                                             | Vertica                                                                        | CloudTran                                                                                         | HyperTable                                                       |
|--------------|----------------------------------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|------------------------------------------------------------------|
| Pros         | Key-based NoSQL, active user community, Cloudera support | Key-based NoSQL, active user community, Amazon's Dynamo on EC2        | Closed-source, SQL-standard, easy to use, visualization tools, complex queries | Closed-source optimized on line transaction processing                                            | Drop-in replacement for HBase, open-source, arguably much faster |
| Cons         | Steeper learning curve, less tools, simpler queries      | Steeper learning curve, less tools, simpler queries                   | Vendor lock-in, price, RDMS/BI - may not fit every application                 | Vendor lock-in, price, transaction-optimized, may not fit every application, needs wider adoption | New, needs user adoption and more testing                        |
| Notes        | Good for new, long-term development                      | Easy to set up, no dependence on HDFS, fully distributed architecture | Good for existing SQL-based applications that needs fast scaling               | Arguably the best OLTP                                                                            | To be kept in mind as a possible alternative                     |

### 8.3.14 Alternatives for distributed massive computations

Here too, depending upon the type of application that the reader needs, other approaches make prove more useful or more fitting to the purpose.

The first such example is the JavaSpaces paradigm. JavaSpaces is a giant hash map container. It provides the framework for building large-scale systems with multiple cooperating computational nodes. The framework is thread-safe and fault-tolerant. Many computers working on the same problem can store their data in a JavaSpaces container. When a node wants to do some work, it finds the data in the container, checks it out, works on it, and then returns it. The framework provides for atomicity. While the node is working on the data, other nodes cannot see it. If it fails, its lease on the data expires, and the data is returned back to the pool of data for processing.



The champion of JavaSpaces is a commercial company called GigaSpaces. The license for a JavaSpaces container from GigaSpaces is free - provided that you can fit into the memory of one computer. Beyond that, GigaSpaces has implemented unlimited JavaSpaces container where multiple servers combine their memories into a shared pool. GigaSpaces has created a big sets of additional functionality for building large distributed systems. So again, everything depends on the reader's particular situation.

GridGain is another Hadoop alternative. The proponents of GridGain claim that while Hadoop is a compute grid and a data grid, GridGain is just a compute grid, so if your data requirements are not huge, why bother?

They also say that it seems to be enormously simpler to use. Study of the tools and prototyping with them can give one a good feel for the most fitting answer.

Terracotta is a commercial open source company, and in the open source realm it provides Java big cache and a number of other components for building large distributed systems. One of its advantages is that it allows existing applications to scale without a significant rewrite. By now we have gotten pretty far away from Hadoop, which proves that we have achieved our goal - give the reader a quick overview of various alternatives for building large distributed systems. Success in whichever way you choose to go!

### 8.3.15 Arguments for Hadoop

We have given the pro arguments for the Hadoop alternatives, but now we can put in a word for the little elephant and its zoo. It boasts wide adoption, has an active community, and has been in production use in many large companies. I think that before embarking on an exciting journey of building large distributed systems, the reader will do well to view the presentation by Jeff Dean, a Google Fellow, on the "Design, Lessons, and Advice from Building Large Distributed Systems" found on SlideShare [[http:// www.slideshare.net/xlight/google-designs-lessons-and-advice-from-building-large-distributed-systems](http://www.slideshare.net/xlight/google-designs-lessons-and-advice-from-building-large-distributed-systems)].

Google has built multiple applications on GFS, MapReduce, and Big Table, which are all implemented as open-source projects in the Hadoop zoo. According to Jeff, the plan is to continue with 1,000,000 to 10,000,000 machines spread at 100s to 1000s of locations around the world, and as arguments go, that is pretty big.<sup>3</sup>

## 8.4 Hadoop Distributed File Systems (HDFS)

In this chapter we shall learn about the Hadoop Distributed File System, also known as HDFS. When people say 'Hadoop' it usually includes two core components: HDFS and MapReduce HDFS is the 'file system' or 'storage layer' of Hadoop. It takes care of storing data -- and it can handle very large amount of data (on a petabyte scale).

In this chapter, we will look at the concepts of HDFS.

### 8.4.1 HDFS Concepts

- **Problem : Data is too big store in one computer**  
Today's big data is 'too big' to store in ONE single computer -- no matter how powerful it is and how much storage it has. This eliminates lot of storage system and databases that were built for single machines. So we are going to build the system to run on multiple networked computers. The file system will look like a unified single file system to the 'outside' world.

**Hadoop solution : Data is stored on multiple computers**

---

<sup>3</sup> This chapter is compiled based on the book "Hadoop Illuminated", by Mark Kerzner and Sujee Maniyam, available on <https://github.com/hadoop-illuminated/hadoop-book>

- **Problem : Very high end machines are expensive**

Now that we have decided that we need a cluster of computers, what kind of machines are they? Traditional storage machines are expensive with top-end components, sometimes with 'exotic' components (e.g. fiber channel for disk arrays, etc). Obviously these computers cost a pretty penny.

We want our system to be cost-effective, so we are not going to use these 'expensive' machines. Instead we will opt to use commodity hardware. By that we don't mean cheapo desktop class machines. We will use performant server class machines -- but these will be commodity servers that you can order from any of the vendors (Dell, HP, etc). So what do these server machines look like?

**Hadoop solution : Run on commodity hardware**

- **Problem : Commodity hardware will fail**

In the old days of distributed computing, failure was an exception, and hardware errors were not tolerated well. So companies providing gear for distributed computing made sure their hardware seldom failed. This is achieved by using high quality components, and having backup systems (in some cases backup to backup systems!). So the machines are engineered to withstand component failures, but still keep functioning.

This line of thinking created hardware that is impressive, but EXPENSIVE! On the other hand we are going with commodity hardware. These don't have high end whiz bang components like the main frames mentioned above. So they are going to fail -- and fail often. We need to be prepared for this. How?

The approach we will take is we build the 'intelligence' into the software. So the cluster software will be smart enough to handle hardware failure. The software detects hardware failures and takes corrective actions automatically -- without human intervention. Our software will be smarter!

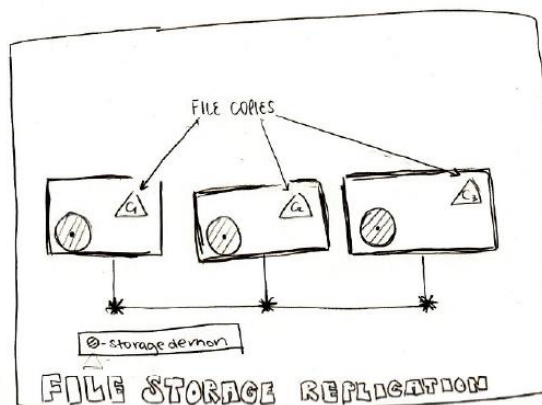
**Hadoop solution : Software is intelligent enough to deal with hardware failure**

- **Problem : hardware failure may lead to data loss**

So now we have a network of machines serving as a storage layer. Data is spread out all over the nodes. What happens when a node fails (and remember, we EXPECT nodes to fail). All the data on that node will become unavailable (or lost). So how do we prevent it?

One approach is to make multiple copies of this data and store them on different machines. So even if one node goes down, other nodes will have the data. This is called 'replication'. The standard replication is 3 copies.

**Hadoop Solution : replicate (duplicate) data**

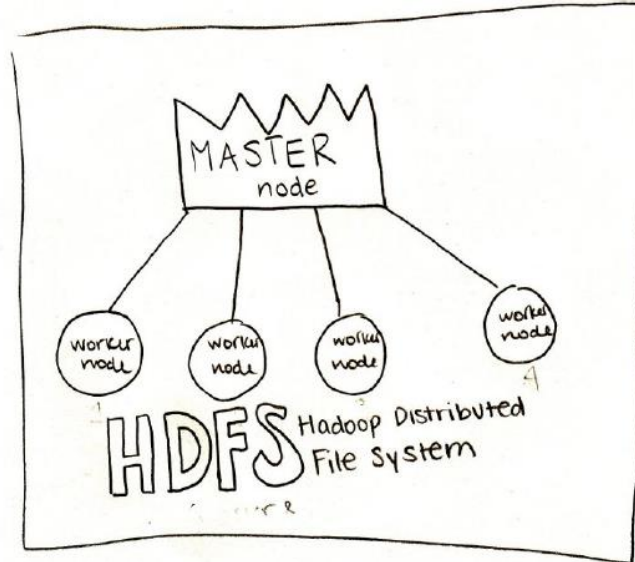


**Fig. 8.9: HDFS File Replication**

- **Problem : how will the distributed nodes co-ordinate among themselves**

Since each machine is part of the 'storage', we will have a 'daemon' running on each machine to manage storage for that machine. These daemons will talk to each other to exchange data. OK, now we have all these nodes storing data, how do we coordinate among them? One approach is to have a MASTER to be the coordinator. While building distributed systems with a centralized coordinator may seem like an odd idea, it is not a bad choice. It simplifies architecture, design and implementation of the system. So now our architecture looks like this. We have a single master node and multiple worker nodes.

**Hadoop solution : There is a master node that co-ordinates all the worker nodes**



**Fig. 8.10: HDFS Master/Worker Design**

## 8.4.2 HDFS Architecture

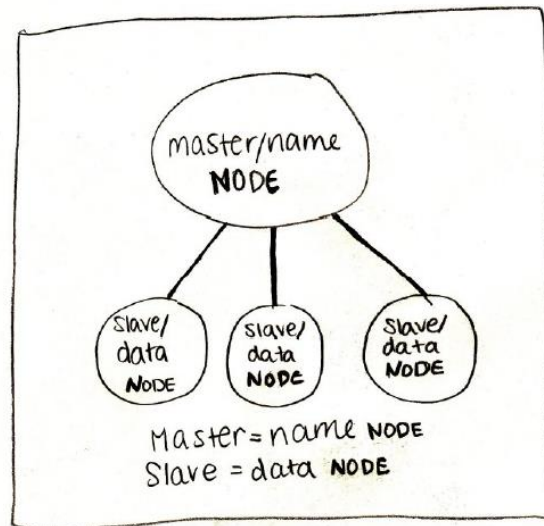
Now we have pretty much arrived at the architecture of HDFS. Let's go over some principles of HDFS. First let's consider the parallels between 'our design' and the actual HDFS design.

### 8.4.2.1 Master / worker design

In an HDFS cluster, there is ONE master node and many worker nodes. The master node is called the Name Node (NN) and the workers are called Data Nodes (DN). Data nodes actually store the data. They are the workhorses.

Name Node is in charge of file system operations (like creating files, user permissions, etc.). Without it, the cluster will be inoperable. No one can write data or read data.

This is called a Single Point of Failure. We will look more into this later.



**Fig. 8.11: HDFS Architecture**

#### 8.4.2.2 Runs on commodity hardware

As we saw Hadoop doesn't need fancy, high end hardware. It is designed to run on commodity hardware. The Hadoop stack is built to deal with hardware failure and the file system will continue to function even if nodes fail.

#### 8.4.2.3 HDFS is resilient (even in case of node failure)

The file system will continue to function even if a node fails. Hadoop accomplishes this by duplicating data across nodes.

#### 8.4.2.4 Data is replicated

So how does Hadoop keep data safe and resilient in case of node failure? Simple, it keeps multiple copies of data around the cluster.

To understand how replication works, let's look at the following scenario. Data segment #2 is replicated 3 times, on data nodes A, B and D. Let's say data node A fails. The data is still accessible from nodes B and D.

#### 8.4.2.5 HDFS is better suited for large files

Generic file systems, say like Linux EXT file systems, will store files of varying size, from a few bytes to few gigabytes. HDFS, however, is designed to store large files. Large as in a few hundred megabytes to a few gigabytes.

Why is this? HDFS was built to work with mechanical disk drives, whose capacity has gone up in recent years. However, seek times haven't improved all that much. So Hadoop tries to minimize disk seeks.

#### 8.4.2.6 Files are write-once only (not updateable)

HDFS supports writing files once (they cannot be updated). This is a stark difference between HDFS and a generic file system (like a Linux file system). Generic file systems allow files to be modified. However, appending to a file is supported. Appending is supported to enable applications like HBase.<sup>4</sup>

### 8.5 Practical Session on MySQL Replication and Partition scheme and HBase Installation

#### 8.5.1 Replication in MySQL

<sup>4</sup> This chapter is compiled based on the book "Hadoop Illuminated", by Mark Kerzner and Sujee Maniyam, available on <https://github.com/hadoop-illuminated/hadoop-book>

## 1. Setting the Replication Master Configuration

Write/Uncomment these lines in my.ini file of master machine

```
[mysqld]
log-bin=mysql-bin
server-id=1
```

## 2. After changing, restart the MySQL master.

## 3. Creating a user for Replication at Master

```
mysql> CREATE USER 'repl'@'%'.mydomain.com IDENTIFIED BY 'slavepass';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%'.mydomain.com';
```

## 4. Make sure that the new user 'repl' has appropriate privileges over appropriate databases.

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'repl'@'localhost';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'repl'@'%';
```

## 5. Obtaining the Replication Master Binary Log Coordinates

```
mysql> FLUSH TABLES WITH READ LOCK;
```

```
mysql > SHOW MASTER STATUS;
```

The File column shows the name of the log file and Position shows the position within the file. Record these values.

## 6. On the master, release the read lock now.

```
mysql> UNLOCK TABLES;
```

## 7. Now, setting the Replication Slave Configuration.

Write/Uncomment the following lines in my.ini file of slave machine

```
[mysqld]
server-id=2
```

## 8. After changing, restart the MySQL slave.

## 9. Setting the Master Configuration on the Slave

Execute the following SQL statement at slave machine.

```
mysql> CHANGE MASTER TO
->     MASTER_HOST='master_host_name',
->     MASTER_USER='replication_user_name',
->     MASTER_PASSWORD='replication_password',
->     MASTER_LOG_FILE='recorded_log_file_name',
->     MASTER_LOG_POS=recorded_log_position;
```

10. Start the slave thread

```
mysql> START SLAVE;
```

11. Now create a table and populate the table at master and observe the replication at slave

12. Some useful commands:

- a. show slave status\G [at slave machine]
- b. show master status\G [at master machine]
- c. show slave hosts [at master machine]

### 8.5.2 Partitioning in MySQL

1. Determine whether your MySQL Server supports partitioning by means of a SHOW VARIABLES statement such as this one:

```
mysql> SHOW VARIABLES LIKE '%partition%';
```

| Variable_name     | Value |
|-------------------|-------|
| have_partitioning | YES   |

1 row in set (0.00 sec)

2. Relationship between Partitioning keys, Primary keys and Unique keys

**All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.**

3. Partitioning Types:

- i) Range
- ii) List
- iii) Hash
- iv) Key

4. Range Partitioning

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT NOT NULL,  
  store_id INT NOT NULL  
)  
PARTITION BY RANGE (store_id) (  
  PARTITION p0 VALUES LESS THAN (6),  
  PARTITION p1 VALUES LESS THAN (11),  
  PARTITION p2 VALUES LESS THAN (16),  
  PARTITION p3 VALUES LESS THAN MAXVALUE  
);
```

## 5. List Partitioning

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT,  
  store_id INT  
)  
PARTITION BY LIST(store_id) (  
  PARTITION pNorth VALUES IN (3,5,6,9,17),  
  PARTITION pEast VALUES IN (1,2,10,11,19,20),  
  PARTITION pWest VALUES IN (4,12,13,14,18),  
  PARTITION pCentral VALUES IN (7,8,15,16)  
);
```

## 6. Hash Partitioning

```
CREATE TABLE employees (  
  id INT NOT NULL,  
  fname VARCHAR(30),  
  lname VARCHAR(30),  
  hired DATE NOT NULL DEFAULT '1970-01-01',  
  separated DATE NOT NULL DEFAULT '9999-12-31',  
  job_code INT,  
  store_id INT  
)  
PARTITION BY HASH( YEAR(hired) )  
PARTITIONS 4;
```

## 7. Key Partitioning

```
CREATE TABLE k1 (  
  id INT NOT NULL PRIMARY KEY,  
  name VARCHAR(20)  
)  
PARTITION BY KEY()  
PARTITIONS 2;
```

## 8. Partitioning Management

## i) changing partition types

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
PARTITION BY RANGE( YEAR(purchased) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (2000),
    PARTITION p3 VALUES LESS THAN (2005)
);
```

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

## ii) dropping a partition

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
-> PARTITION BY RANGE( YEAR(purchased) ) (
-> PARTITION p0 VALUES LESS THAN (1990),
-> PARTITION p1 VALUES LESS THAN (1995),
-> PARTITION p2 VALUES LESS THAN (2000),
-> PARTITION p3 VALUES LESS THAN (2005)
-> );
```

```
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> ALTER TABLE tr DROP PARTITION p2;
```

```
Query OK, 0 rows affected (0.03 sec)
```

## iii) adding a partition

```
CREATE TABLE members (
    id INT,
    fname VARCHAR(25),
    lname VARCHAR(25),
    dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION p0 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990)
);
```

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2000));
```

## iv) some other useful commands

```
SHOW CREATE TABLE <table_name>;
EXPLAIN PARTITIONS <Query Expression>;
SELECT * FROM <table_name> PARTITION (p0,p1); [only supported in MySQL
5.6.X version]
```

### 8.5.3 Hadoop and HBase Installation in Windows 7 using Cygwin

This guide was written for [Cygwin 1.7.7](#), [Hadoop 0.21.0](#) and [HBase 0.20.6](#).

You will additionally need [ZooKeeper 3.3.1](#) in order to get HBase to run properly.

Throughout this guide I will assume that your Cygwin install path will be c:\cygwin and that Hadoop, ZooKeeper and HBase will be installed in c:\cygwin\etc\local (/etc/local/), this is however something you can choose yourself. If you choose to install Cygwin elsewhere, I would recommend to use folder names without whitespaces and other non-regular characters.

The only prerequisite for this quite is that you have Java installed and added to your %PATH% variable (which is usually done automatically).



## 8.5.4 Software

Download each software bundle and put it somewhere where you'll easily find it later.

- Cygwin: [Cygwin 1.7.7](#),
- Hadoop: [Hadoop 0.21.0](#)
- HBase: [HBase 0.20.6](#)
- ZooKeeper: [ZooKeeper 3.3.1](#)

### 8.5.4.1 Cygwin

If you've never used Cygwin (or Linux/Unix/etc), you should perhaps get familiar with those environments first. If you still want to continue, read on.

Throughout the Cygwin section - if you find yourself lost, please follow [Vlad Korolev's](#) guide on how to get Cygwin up and running for Hadoop and make sure to additionally install *tcp\_wrappers* and *diffutils* when choosing packages. Follow steps 2 to 4 in the guide and then continue with the Hadoop installation guide below.

If you're familiar with Cygwin you just need to make sure you have these packages installed:

- openssh
- openssl
- tcp\_wrappers
- diffutils

Additionally you will have to set configure ssh to start as a service, and enable passwordless logins. To do this, fire up a Cygwin terminal window after you've completed the installation and do the following:

```
ssh-host-config
```

When asked about privilege separation answer no

When asked if sshd should be installed as a service answer yes

When asked about the CYGWIN environment variable, enter ntsec

Now go to the Services and Applications toll in Windows, locate the CYGWIN sshd service and start it.

Next Cygwin step is to set up the passwordless login. Go to your Cygwin terminal and enter

```
ssh-keygen
```

Do not use a passphrase and accept all default values. After the process is finished do the following

```
1 cd ~/.ssh \n
```

```
2
```

```
3 cat id_rsa.pub >> authorized_keys
```

This will add your identification key to the set of authorized keys, i.e. those that are allowed to login without entering a password.

Try connecting to localhost to see whether it works

```
ssh localhost
```

Doing this the first time should prompt you with a warning, enter yes and enter. Now try issuing the same command, this time there should be no warning and no need to enter a password.

This concludes the Cygwin installation.

### 8.5.4.2 Hadoop

Since Vlad's guide is made for Hadoop 0.19.0, some of the configuration details specified in his guide do not apply anymore (or have moved to other files), this is an updated version what you'll find in his guide.

First - copy the downloaded tar.gz file to c:\cygwin\usr\local (which corresponds to /usr/local in the Cygwin environment). When this is done, it's time to extract the package, this is done by issuing  
tar xvzf hadoop-0.21.0.tar.gz

This command extracts the content of the downloaded hadoop file into c:\cygwin\usr\local\hadoop-0.21.0 (/usr/local/hadoop-0.21.0). Hadoop requires some configuration, the configuration files are located in c:\cygwin\usr\local\hadoop-0.21.0\conf. The files that need to be altered are:

core-site.xml

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://127.0.0.1:9100</value>
</property>
```

mapred-site.xml

```
<property>
  <name>mapred.job.tracker</name>
  <value>127.0.0.1:9101</value>
</property>
```

and hdfs-site.xml

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.permissions</name>
  <value>>false</value>
</property>
```

Only Hadoop 0.21.0: Next, one line has to be added to the hadoop-config.sh file in hadoop-0.21.-0/bin

```
CLASSPATH=`cygpath -wp "$CLASSPATH"`
```

Add this line before the line containing

```
JAVA_LIBRARY_PATH=
```

The reason for this is that in order for the CLASSPATH to be build with all the Hadoop jars (line ~120 to ~200) the path needs to be in the Cygwin format (/cygdrive/c/cygwin/usr/local/hadoop...), however in order for Java use the classpath, it needs to be in the Windows format (c:\cygwin\usr\local\hadoop..). The line transforms the Cygwin built classpath to one that is understood by Windows.

This should be enough for Hadoop to run, test the installation by issuing these commands in a Cygwin window:

```
cd /usr/local/hadoop-0.21.0
mkdir logs
bin/hadoop namenode -format
```

The last command will take some seconds to finish and should produce about 20 lines of output during the creation of the namenode filesystem. The final step of the Hadoop setup is to start it and test it. To start it issue the following commands in a Cygwin window:

```
cd /usr/local/hadoop-0.21.0
bin/start-dfs.sh
bin/start-mapred.sh
```

Providing no error messages are printed, this should have started Hadoop. This can be checked by opening <http://localhost:9100> and <http://localhost:9101> in a browser. The first link should provide information about the NameNode, make sure that the Live Nodes count is 1. The second link provides information about the cluster. Now it's time to run a little job on the cluster to see whether or not the installation was successful. First, copy some files to the node:

```
cd /usr/local/hadoop-0.21.0
mkdir input
cp conf/*&#42;.xml input
bin/hadoop jar hadoop-&#42;examples.jar grep input output 'dfs[a-z.]+'
cat output/&#42;
```

Provided there were no errors, you've just run your first Hadoop process.

### 8.5.4.3 ZooKeeper

This step, it seems, is only necessary if you're installing the setup on 64 bit Windows. The problem seems to be that the ZooKeeper server which comes bundled with HBase does not work correctly, and thus a standalone one needs to be set up.

Luckily the ZooKeeper install and configuration is quite easy.

First, copy the download `zookeeper-3.3.1-tar.gz` file to your `c:\cygwin\usr\local` directory, open a Cygwin window and issue the following commands:

```
cd /usr/local/
tar xvzf zookeeper-3.3.1.tar.gz
```

ZooKeeper's configuration file (`zoo.cfg`) is located in `/usr/local/zookeeper-3.3.1/conf` (`c:\cygwin\usr\loca\zookeeper-3.3.1\conf`).

Open the file and paste the following content into it, overwriting the original config:

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper/data
# the port at which the clients will connect
clientPort=2181
```

Make sure to create the `/tmp/zookeeper/data` directory and make it writable for everyone (`chmod 777`). ZooKeeper is started by typing:

```
cd /user/local/zookeeper-3.3.1
bin/zkServer.sh start
```

Make sure to test if ZooKeeper is running correctly by connecting to it

```
bin/zkCli.sh -server 127.0.0.1:2181
```

This should connect you to ZooKeeper, you can type *help* to see what commands are available, however the only one you need to care about is *quit*.

#### 8.5.4.4 HBase

Start by copying hbase-0.20.6.tar.gz to c:\cygwin\usr\local and extracting it by issuing

```
tar xvzf hbase-0.20.6.tar.gz
```

in a Cygwin terminal.

Now it's time to create a symlink to your JRE directory in /usr/local/. Do this by typing:

```
ln -s /cygdrive/c/Program\ Files/Java/<jre name> /usr/local/<jre name>
```

in a Cygwin terminal. <jre name> will most likely be jre6, but be sure to double check this before making the link.

HBase's configuration files are located in /usr/local/hbase-0.20.6/conf/ (C:\cygwin\usr\local\hbase-0.20.6\conf), and to get HBase up and running we need to edit hbase-env.sh and hbase-default.xml.

In the hbase-env.sh the JAVA\_HOME, HBASE\_IDENT\_STRING and HBASE\_MANAGES\_ZK variables have to be set, this is done by editing the lines containing the variable names to read:

```
export JAVA_HOME=/usr/local/jre6
export HBASE_IDENT_STRING=$HOSTNAME
export HBASE_MANAGES_ZK=false
```

The last variable tells HBase not to use the bundled ZooKeeper server, as we've already installed a stand alone one.

Next, the hbase-default.xml file has to be edited, the two properties that need to be set are *hbase.rootdir* and *hbase.tmp.dir*

```
<property>
<name>hbase.rootdir</name>
<value>file:///C:/cygwin/tmp/hbase/data</value>
<description>The directory shared by region servers.
Should be fully-qualified to include the filesystem to use.
E.g: hdfs://NAMENODE_SERVER:PORT/HBASE_ROOTDIR
</description>
</property>
<property>
<name>hbase.tmp.dir</name>
<value>C:/cygwin/tmp/hbase/tmp</value>
<description>Temporary directory on the local filesystem.</description>
</property>
```

Make sure that both directories exist and are writeable by all users (chmod 777).

The command for starting HBase is:

```
cd /user/local/hbase-0.20.6
```

```
bin/start-hbase.sh
```

This section is very similar to what's found on the HBase [wiki](#), the difference is the standalone ZooKeeper config.

#### Start your cluster

Having done all these steps, it's time to start up the cluster.

The startup procedure should follow this order:

1. ZooKeeper
2. Hadoop
3. HBase

So what you do is:

ZooKeeper:

```
cd /usr/local/zookeeper-3.3.1  
bin/zkServer.sh start
```

Hadoop:

```
cd /usr/local/hadoop-0.21.0  
bin/start-dfs.sh  
bin/start-mapred.sh
```

HBase:

```
cd /usr/bin/hbase-0.20.6  
bin/start-hbase.sh
```

**CHAPTER 9**  
**CLOUD RESEARCH CHALLENGES**

.

## 9. CLOUD RESEARCH CHALLENGES

### 9.1 Introduction

Though the concept of “clouds” is not new, it is undisputable that they have proven a major commercial success over recent years and will play a large part in the ICT domain over the next 10 years or more, as future systems will exploit the capabilities of managed services and resource provisioning further. Clouds are of particular commercial interest not only with the growing tendency to outsource IT so as to reduce management overhead and to extend existing, limited IT infrastructures, but even more importantly, they reduce the entrance barrier for new service providers to offer their respective capabilities to a wide market with a minimum of entry costs and infrastructure requirements – in fact, the special capabilities of cloud infrastructures allow providers to experiment with novel service types whilst reducing the risk of wasting resources.

Cloud systems are not to be misunderstood as just another form of resource provisioning infrastructure and in fact, as this report shows, multiple opportunities arise from the principles for cloud infrastructures that will enable further types of applications, reduced development and provisioning time of different services. Cloud computing has particular characteristics that distinguish it from classical resource and service provisioning environments: (1) it is (more-or-less) infinitely scalable; (2) it provides one or more of an infrastructure for platforms, a platform for applications or applications (via services) themselves; (3) thus clouds can be used for every purpose from disaster recovery/business continuity through to a fully outsourced ICT service for an organization; (4) clouds shift the costs for a business opportunity from Capital Expenditure (CAPEX) to Operational Expenditure (OPEX) which allows finer control of expenditure and avoids costly asset acquisition and maintenance reducing the entry threshold barrier; (5) currently the major cloud providers had already invested in large scale infrastructure and now offer a cloud service to exploit it; (6) as a consequence the cloud offerings are heterogeneous and without agreed interfaces; (7) cloud providers essentially provide data centers for outsourcing; (8) there are concerns over security if a business places its valuable knowledge, information and data on an external service; (9) there are concerns over availability and business continuity – with some recent examples of failures; (10) there are concerns over data shipping over anticipated broadband speeds.

The concept of cloud computing is linked intimately with those of IaaS (Infrastructure as a Service); PaaS (Platform as a Service), SaaS (Software as a Service) and collectively \*aaS (Everything as a Service) all of which imply a service-oriented architecture.

#### Open Research Issues

Cloud Technologies and models have not yet reached their full potential and many of the capabilities associated with clouds are not yet developed and researched to a degree that allows their exploitation to the full degree. We can distinguish technological gaps on the one hand, that need to be closed in order to realize cloud infrastructures that fulfill the specific cloud characteristics and non-technological issues on the other hand that in particular reduce uptake and viability of cloud systems. To the technological aspects belong in particular issues related to (1) scale and elastic scalability, which is not only currently restricted to horizontal scale out, but also inefficient as it tends to resource over usage due to limited scale down capabilities and full replication of instances rather than only of essential segments. (2) Trust, security and privacy always pose issues in any internet provided service, but due to the specific nature of clouds, additional aspects related e.g. to multi-tenancy arise and control over data location etc. arise. What is more, clouds simplify malicious use of resources, e.g. for hacking purposes, but also for sensitive calculations (such as weapon design) etc. (3) Handling data in clouds is still complicated - in particular as data size and diversity grows, pure replication is no viable approach, leading to consistency and efficiency issues. Also, the lacking control over data location and missing provenance poses security and legalistic issues. (4) Programming models are currently not aligned to highly scalable applications and thus do not exploit the capabilities of clouds, whilst they should also simplify development. Along the same line, developers, providers and users should be able to control and restrict distribution and scaling behavior. This relates to (5) systems

development and management which is currently still executed mostly manually, thus contributing to substantial efficiency and bottleneck issues.

On the other hand, non-technological issues play a major role in realizing these technological aspects and in ensuring viability of the infrastructures in the first instance. To these belong in particular (1) economic aspects which cover knowledge about when, why, how to use which cloud system how this impacts on the original infrastructure (provider) –long-term experience is lacking in all these areas; and (2) legalistic issues which come as a consequence from the dynamic (location) handling of the clouds, their scalability and the partially unclear legislative issues in the internet. This covers in particular issues related to intellectual property rights and data protection. In addition, (3) aspects related to green IT need to be elaborated further, as the cloud offers principally “green capabilities” by reducing unnecessary power consumption, given that good scaling behavior and good economic models are in place.

## 9.2 The advent of the “clouds”

The increased degree of connectivity and the increasing amount of data has led many providers and in particular data centers to employ larger infrastructures with dynamic load and access balancing. By distributing and replicating data across servers on demand, resource utilization has been significantly improved. Similarly web server hosts replicate images of relevant customers who requested a certain degree of accessibility across multiple servers and route requests according to traffic load. However, it was only when Amazon published these internal resources and their management mechanisms for use by customers that the term “cloud” was publicly associated with such elastic infrastructures – especially with “on demand” access to IT resources in mind. In the meantime, many providers have rebranded their infrastructures to “clouds”, even though this had little consequences on the way they provided their capabilities.

It may be noted in this context that the term “cloud” dates back to the 90s in reference to the capability of dynamic traffic switching to balance utilization (“telecom clouds”) and to indicate that the telecoms infrastructure is virtualized – the end user does not know or care over which channels her data is routed (see IETF meeting minutes [1]). Microsoft adopted the term 2001 in a public presentation about the .NET framework to refer to the infrastructure of computers that make up the internet [2]. According to Wikipedia, the underlying concept of cloud computing can be dated even further back to a public speech given by John McCarthy 1961 where he predicts that computer time-sharing may lead to the provisioning of computing resources and applications as a utility [3].

Concept and even technological approaches behind “cloud computing” can thus not be considered a novelty as such and in particular data centers already employed methods to maintain scalability and reliability to ensure availability of their hosted data. What is more, cloud systems are, unlike e.g. grid computing, not driven by research first and then being taken up by industry, but instead originates directly from commercial requirements and solutions. It is hence not surprising, that the term “cloud computing” and its current understanding only really became popular with Amazon’s publication of the Elastic Compute Cloud EC2 in 2006 [4], giving rise to a small boom of “cloud offerings” which mostly consisted in a rebranding of their existent in-house solutions and techniques, as well as a potential exposition of these capabilities to consumers. Multiple new “cloud” domains and providers have thus arisen and it is not surprising, that the term has found multiple related, yet different meanings. In particular, the scope of areas and capabilities that so-called clouds are applied for differs thereby strongly. The most typical representatives for cloud related functionalities can currently be found in the following areas: (1) data centers trying to maintain high scalability and increase availability; (2) web server farms automating and stabilizing their servers, respectively the user’s website; (3) in house attempts to balance resources over the business solutions; (4) external ASP-type offerings. It must be made clear in this context that “Clouds” do generally not refer to a specific technology or framework, but rather to a set of combined technologies, respectively a paradigm / concept. The “Grid” and Service Oriented Architectures are often confused as being identical with clouds due to this primarily conceptual understanding. Likewise, current “cloud providers” typically



build upon proprietary technology sets and approaches based on their in-house solutions -only little efforts have been undertaken so far, to build up a generic framework / middleware supporting all the features related to clouds.

It's only been in 2004 that multi-core processing became available for common desktop machines, when Intel finally abandoned the development of a 4 GHz processor and switched to multi-core development instead [5]. Implicitly even more mainstream developers and users investigate the specific advantages and problems of not only horizontal, but also vertical scalability. Additionally, with the "Prosumer" [6] movement, as well as the growing demand to lower management cost and the carbon footprint make outsourcing more and more interesting for the market. It is to be expected that the cloud paradigm will find further uptake in the future – not only as a means to manage the infrastructure of providers, but also to provide smaller entities with the capabilities of a larger infrastructure that they cannot afford to own themselves. At the same time, the cloud paradigm will allow for a set of enhanced capabilities and services not feasible before.

### 9.2.1 What is a "Cloud"

Various definitions and interpretations of "clouds" and / or "cloud computing" exist. With particular respect to the various usage scopes the term is employed to, we will try to give a representative (as opposed to complete) set of definitions as recommendation towards future usage in the cloud computing related research space.

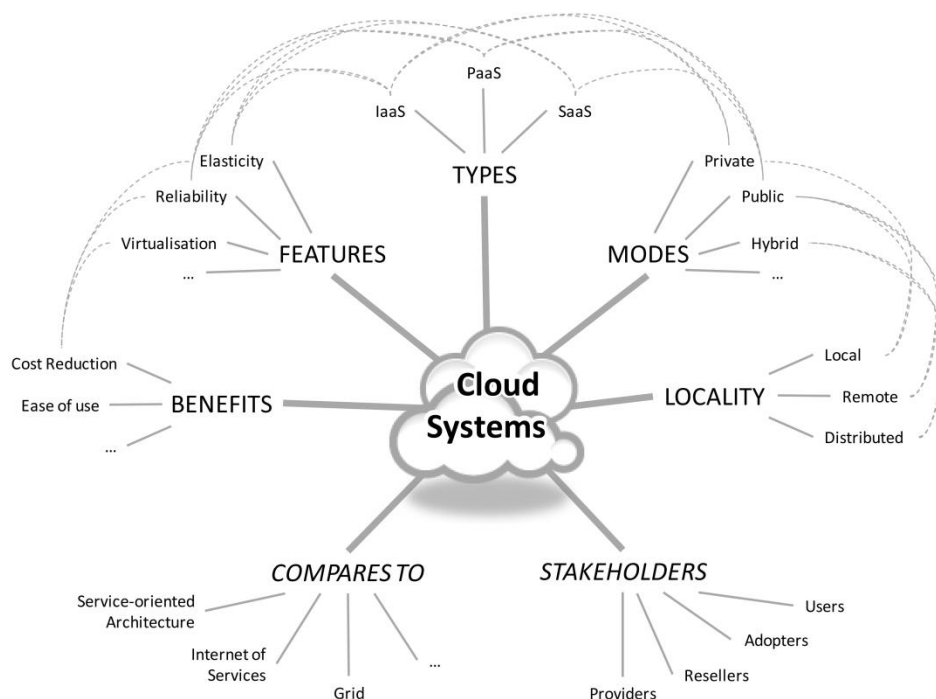


Fig. 9.1: Non-Exhaustive View on the Main Aspects Forming a Cloud System

### 9.2.2 Terminology

In its broadest form, we can define a 'cloud' is an elastic execution environment of resources involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality (of service).

In other words, clouds as we understand them in the context of this document are primarily platforms that allow execution in various forms across multiple resources (and potentially across enterprise boundaries). Resources are put to use according to actual current requirements observing overarching

requirement definitions – implicitly, elasticity includes both up- and downward scalability of resources and data, but also load-balancing of data throughput.

As shall be elaborated, future cloud systems should also be able to maintain a pre-specified level of quality, respectively boundary conditions (including performance, energy consumption, etc.) and should allow integration of resources across organizational boundaries, integrating multiple stakeholders.

Noticeably, the actual details of the capabilities differ slightly depending on how the cloud is employed: since clouds relate to a usage concept, rather than a technology, it has been applied to different areas, as described in the introductory part of this document. We therefore need to distinguish what kinds of capabilities are provided by different cloud systems:

### 9.2.2.1 Types of Clouds

Cloud providers typically centre on one type of cloud functionality provisioning: Infrastructure, Platform or Software / Application, though there is potentially no restriction to offer multiple types at the same time, which can often be observed in PaaS (Platform as a Service) providers which offer specific applications too, such as Google App Engine in combination with Google Docs. Due this combinatorial capability, these types are also often referred to as “components” (see e.g. [7]).

Literature and publications typically differ slightly in the terminologies applied. This is mostly due to the fact that some application areas overlap and are therefore difficult to distinguish. As an example, platforms typically have to provide access to resources indirectly, and thus are sometimes confused with infrastructures. Additionally, more popular terms have been introduced in less technologically centered publications.

The following list identifies the main types of clouds (currently in use):

**(Cloud) Infrastructure as a Service (IaaS)** also referred to as *Resource Clouds*, provide (managed and scalable) resources as services to the user – in other words, they basically provide enhanced virtualization capabilities. Accordingly, different resources may be provided via a service interface:

*Data & Storage Clouds* deal with reliable access to data of potentially dynamic size, weighing resource usage with access requirements and / or quality definition.

Examples: Amazon S3, SQL Azure.

*Compute Clouds* provide access to computational resources, i.e. CPUs. So far, such low-level resources cannot really be exploited on their own, so that they are typically exposed as part of a “virtualized environment” (not to be mixed with PaaS below), i.e. hypervisors. Compute Cloud

Providers therefore typically offer the capability to provide computing resources (i.e. raw access to resources unlike PaaS that offer full software stacks to develop and build applications), typically virtualized, in which to execute cloudified services and applications. IaaS (Infrastructure as a Service) offers additional capabilities over a simple compute service.

Examples: Amazon EC2, Zimory, Elastichosts.

**(Cloud) Platform as a Service (PaaS)**, provide computational resources via a *platform* upon which applications and services can be developed and hosted. PaaS typically makes use of dedicated APIs to control the behavior of a server hosting engine which executes and replicates the execution according to user requests (e.g. access rate). As each provider exposes his / her own API according to the respective key capabilities, applications developed for one specific cloud provider cannot be moved to another cloud host – there are however attempts to extend generic programming models with cloud capabilities (such as MS Azure).

Examples: Force.com, Google App Engine, Windows Azure (Platform).

**(Clouds) Software as a Service (SaaS)**, also sometimes referred to as *Service or Application Clouds* are offering implementations of specific business functions and business processes that are provided with specific cloud capabilities, i.e. they provide applications / services *using* a cloud infrastructure or platform, rather than providing cloud features themselves. Often, kind of standard application software functionality is offered within a cloud.

Examples: Google Docs, Salesforce CRM, SAP Business by Design.

Overall, Cloud Computing is not restricted to Infrastructure / Platform / Software-as-a-Service systems, even though it provides enhanced capabilities which act as (vertical) enablers to these systems. As such, I/P/SaaS can be considered specific “usage patterns” for cloud systems which relate to models already approached by Grid, Web Services etc. Cloud systems are a promising way to implement these models and extend them further.

### 9.2.2.2 Deployment Types (Cloud Usage)

Similar to P/I/SaaS, clouds may be hosted and employed in different fashions, depending on the use case, respectively the business model of the provider. So far, there has been a tendency of clouds to evolve from private, internal solutions (private clouds) to manage the local infrastructure and the amount of requests e.g. to ensure availability of highly requested data. This is due to the fact that data centers initiating cloud capabilities made use of these features for internal purposes before considering selling the capabilities publicly (public clouds). Only now that the providers have gained confidence in publication and exposition of cloud features do the first hybrid solutions emerge. This movement from private via public to combined solutions is often considered a “natural” evolution of such systems, though there is no reason for providers to not start up with hybrid solutions, once the necessary technologies have reached a mature enough position. We can hence distinguish between the following deployment types:

**Private Clouds** are typically owned by the respective enterprise and / or leased. Functionalities are not directly exposed to the customer, though in some cases services with cloud enhanced features may be offered – this is similar to (Cloud) Software as a Service from the customer point of view.

Example: eBay.

**Public Clouds.** Enterprises may use cloud functionality from others, respectively offer their own services to users outside of the company. Providing the user with the actual capability to exploit the cloud features for his/her own purposes also allows other enterprises to outsource their services to such cloud providers, thus reducing costs and effort to build up their own infrastructure. As noted in the context of cloud *types*, the scope of functionalities thereby may differ.

Example: Amazon, Google Apps, Windows Azure.

**Hybrid Clouds.** Though public clouds allow enterprises to outsource parts of their infrastructure to cloud providers, they at the same time would lose control over the resources and the distribution /management of code and data. In some cases, this is not desired by the respective enterprise.

*Hybrid clouds* consist of a mixed employment of *private* and *public cloud* infrastructures so as to achieve a maximum of cost reduction through outsourcing whilst maintaining the desired degree of control over e.g. sensitive data by employing local private clouds.

There are not many hybrid clouds actually in use today, though initial initiatives such as the one by IBM and Juniper already introduce base technologies for their realization [10].

**Community Clouds.** Typically cloud systems are restricted to the local infrastructure, i.e. providers of public clouds offer their own infrastructure to customers. Though the provider could actually resell the infrastructure of another provider, clouds do not *aggregate* infrastructures to build up larger,

cross-boundary structures. In particular smaller SMEs could profit from *community clouds* to which different entities contribute with their respective (smaller) infrastructure. Community clouds can either aggregate public clouds or dedicated resource infrastructures. Community clouds show some overlap with GRIDs technology (see e.g. Reservoir [14]).

**Special Purpose Clouds.** In particular IaaS clouds originating from data centers have a “general purpose” appeal to them, as their according capabilities can be equally used for a wide scope of use cases and customer types. As opposed to this, PaaS clouds tend to provide functionalities more specialized to specific use cases, which should not be confused with “proprietaryness” of the platform: specialization implies providing additional, *use case specific methods*, whilst proprietary data implies that *structure* of data and interface are specific to the *provider*. Specialized functionalities are provided e.g. by the Google App Engine which provides specific capabilities dedicated to distributed document management. Similar to general service provisioning (web based or not), it can be expected that future systems will provide even more specialized capabilities to attract individual user areas, due to competition, customer demand and available expertise.

Special Purpose Clouds are just extensions of “normal” cloud systems to provide additional, dedicated capabilities.

### 9.2.2.3 Cloud Environment Roles

In cloud environments, individual roles can be identified similar to the typical role distribution in Service Oriented Architectures and in particular in (business oriented) Virtual Organizations. As the roles relate strongly to the individual business models it is imperative to have a clear definition of the types of roles involved in order to ensure common understanding.

**(Cloud) Providers** offer *clouds* to the customer – either via dedicated APIs (PaaS), virtual machines and / or direct access to the resources (IaaS). Note that hosts of cloud enhanced services (SaaS) are typically referred to as *Service Providers*, though there may be ambiguity between the terms Service Provider and Cloud Provider.

**(Cloud) Resellers or Aggregators** aggregate cloud platforms from *cloud providers* to either provide a larger resource infrastructure to their customers or to provide enhanced features. This relates to *community clouds* in so far as the cloud aggregators may expose a single interface to a merged cloud infrastructure. They will match the economic benefits of global cloud infrastructures with the understanding of local customer needs by providing highly customized, enhanced offerings to local companies (especially SME’s).

**(Cloud) Adopters or (Software / Services) Vendors** enhance their own services and capabilities by exploiting cloud platforms from *cloud providers* or *cloud resellers*. This enables them to e.g. provide services that scale to dynamic demands – in particular new business entries who cannot estimate the uptake / demand of their services as yet (cf. II.B.1). The cloud enhanced services thus effectively become *software as a service*.

**(Cloud) Consumers or Users** make *direct* use of the cloud capabilities – as opposed to *cloud resellers* and *cloud adopters*, however, not to improve the services and capabilities they offer, but to make use of the direct results, i.e. either to execute complex computations or to host a flexible data set. Note that this involves in particular larger enterprises which outsource their inhouse infrastructure to reduce cost and efforts (see also *hybrid clouds*). Note that future market developments will most likely enable the user to become provider and consumer at the same time, thus following the “Prosumer” concept.

**(Cloud) Tool Providers** do not actually provide cloud capabilities, but supporting tools such as programming environments, virtual machine management etc.

### 9.2.3 Specific Characteristics / Capabilities of Clouds

Since “clouds” do not refer to a specific technology, but to a general provisioning paradigm with enhanced capabilities, it is mandatory to elaborate on these aspects. There is currently a strong tendency to regard clouds as “just a new name for an old idea”, which is mostly due to a confusion between the cloud concepts and the strongly related P/I/SaaS paradigms, but also due to the fact that similar aspects have already been addressed without the dedicated term “cloud” associated with it.

This section specifies the concrete capabilities associated with clouds that are considered *essential* (required in any cloud environment) and *relevant* (ideally supported, but may be restricted to specific use cases). We can thereby distinguish non-functional, economic and technological capabilities addressed, respectively to be addressed by cloud systems.

**Non-functional aspects** represent *qualities* or *properties* of a system, rather than specific technological requirements. Implicitly, they can be realized in multiple fashions and interpreted in different ways which typically leads to strong compatibility and interoperability issues between individual providers as they pursue their own approaches to realize their respective requirements, which strongly differ between providers. Non-functional aspects are one of the key reasons why “clouds” differ so strongly in their interpretation.

**Economic considerations** are one of the key reasons to introduce cloud systems in a business environment in the first instance. The particular interest typically lies in the reduction of cost and effort through outsourcing and / or automation of essential resource management. As has been noted in the first section, relevant aspects thereby to consider relate to the cut-off between loss of control and reduction of effort. With respect to hosting private clouds, the gain through cost reduction has to be carefully balanced with the increased effort to build and run such a system. Obviously, **technological challenges** implicitly arise from the non-functional and economical aspects, when trying to realize them. As opposed to these aspects, technological challenges typically imply a specific realization – even though there may be no standard approach as yet and deviations may hence arise. In addition to these implicit challenges, one can identify additional technological aspects to be addressed by cloud system, partially as a pre-condition to realize some of the high level features, but partially also as they directly relate to specific characteristics of cloud systems.

#### 9.2.3.1 Non-Functional Aspects

The most important non-functional aspects are:

- **Elasticity** is an essential core feature of cloud systems and circumscribes the capability of the underlying infrastructure to adapt to changing, potentially non-functional requirements, for example amount and size of data supported by an application, number of concurrent users etc. One can distinguish between horizontal and vertical scalability, whereby *horizontal scalability* refers to the amount of instances to satisfy e.g. changing amount of requests, and *vertical scalability* refers to the size of the instances themselves and thus implicit to the amount of resources required maintaining the size. Cloud scalability involves both (rapid) up- and down-scaling. Elasticity goes one step further, though, and does also allow the dynamic integration and extraction of physical resources to the infrastructure. Whilst from the application perspective, this is identical to scaling, from the middleware management perspective this poses additional requirements, in particular regarding reliability. In general, it is assumed that changes in the resource infrastructure are announced first to the middleware manager, but with large scale systems it is vital that such changes can be maintained automatically.
- **Reliability** is essential for all cloud systems – in order to support today’s data centre-type applications in a cloud, reliability is considered one of the main features to exploit cloud capabilities. Reliability denotes the capability to ensure constant operation of the system without disruption, i.e. no loss of data, no code reset during execution etc. Reliability is typically achieved through redundant resource utilisation. Interestingly, many of the reliability aspects move from hardware to a software-based solution (Redundancy in the file systems vs.

RAID controllers, stateless front end servers vs. UPS, etc.). Notably, there is a strong relationship between availability and reliability – however, reliability focuses in particular on prevention of loss (of data or execution progress).

- **Quality of Service** support is a relevant capability that is essential in many use cases where specific requirements have to be met by the outsourced services and / or resources. In business cases, basic QoS metrics like response time, throughput etc. must be guaranteed at least, so as to ensure that the quality guarantees of the cloud user are met. *Reliability* is a particular QoS aspect which forms a specific quality requirement.
- **Agility and adaptability** are essential features of cloud systems that strongly relate to the elastic capabilities. It includes on-time reaction to changes in the amount of requests and size of resources, but also adaptation to changes in the environmental conditions that e.g. require different *types* of resources, different *quality* or different *routes*, etc. Implicitly, agility and adaptability require resources (or at least their management) to be autonomic and have to enable them to provide self-\* capabilities.
- **Availability** of services and data is an essential capability of cloud systems and was actually one of the core aspects to give rise to clouds in the first instance. It lies in the ability to introduce redundancy for services and data so failures can be masked transparently. Fault tolerance also requires the ability to introduce new redundancy (e.g. previously failed or fresh nodes) in an online manner non-intrusively (without a significant performance penalty).

With increasing concurrent access, availability is particularly achieved through replication of data /services and distributing them across different resources to achieve load-balancing. This can be regarded as the original essence of scalability in cloud systems.

### 9.2.3.2 Economic Aspects

In order to allow for economic considerations, cloud systems should help in realizing the following aspects:

- **Cost reduction** is one of the first concerns to build up a cloud system that can adapt to changing consumer behavior and reduce cost for infrastructure maintenance and acquisition. *Scalability* and *Pay per Use* are essential aspects of this issue. Notably, setting up a cloud system typically entails additional costs – be it by adapting the business logic to the cloud host specific interfaces or by enhancing the local infrastructure to be “cloud-ready”. See also *return of investment* below.
- **Pay per use.** The capability to build up cost according to the actual consumption of resources is a relevant feature of cloud systems. Pay per use strongly relates to quality of service support, where specific requirements to be met by the system and hence to be paid for can be specified. One of the key economic drivers for the current level of interest in cloud computing is the structural change in this domain. By moving from the usual capital upfront investment model to an operational expense, cloud computing promises to enable especially SME’s and entrepreneurs to accelerate the development and adoption of innovative solutions.
- **Improved time to market** is essential in particular for small to medium enterprises that want to sell their services quickly and easily with little delays caused by acquiring and setting up the infrastructure, in particular in a scope compatible and competitive with larger industries. Larger enterprises need to be able to publish new capabilities with little overhead to remain competitive. Clouds can support this by providing infrastructures, potentially dedicated to specific use cases that take over essential capabilities to support easy provisioning and thus reduce time to market.

- **Return of investment (ROI)** is essential for all investors and cannot always be guaranteed – in fact some cloud systems currently fail this aspect. Employing a cloud system must ensure that the cost and effort vested into it is outweighed by its benefits to be commercially viable – this may entail direct (e.g. more customers) and indirect (e.g. benefits from advertisements) ROI. Outsourcing resources versus increasing the local infrastructure and employing (private) cloud technologies need therefore to be outweighed and critical cut-off points identified.
- **Turning CAPEX into OPEX** is an implicit, and much argued characteristic of cloud systems, as the actual cost benefit (cf. ROI) is not always clear (see e.g.[8]). Capital expenditure (CAPEX) is required to build up a local infrastructure, but with outsourcing computational resources to cloud systems on demand and scalable, a company will actually spend operational expenditure (OPEX) for provisioning of its capabilities, as it will acquire and use the resources according to operational need.
- **“Going Green”** is relevant not only to reduce additional costs of energy consumption, but also to reduce the carbon footprint. Whilst carbon emission by individual machines can be quite well estimated, this information is actually taken little into consideration when scaling systems up. Clouds principally allow reducing the consumption of unused resources (down-scaling). In addition, up-scaling should be carefully balanced not only with cost, but also carbon emission issues. Note that beyond software stack aspects, plenty of Green IT issues are subject to development on the hardware level.

### 9.2.3.3 Technological Aspects

The main technological challenges that can be identified and that are commonly associated with cloud systems are:

- **Virtualization** is an essential technological characteristic of clouds which hides the technological complexity from the user and enables enhanced flexibility (through aggregation, routing and translation). More concretely, virtualization supports the following features:

*Ease of use:* through hiding the complexity of the infrastructure (including management, configuration etc.) virtualization can make it easier for the user to develop new applications, as well as reduces the overhead for controlling the system.

*Infrastructure independency:* in principle, virtualization allows for higher interoperability by making the code platform independent.

*Flexibility and Adaptability:* by exposing a virtual execution environment, the underlying infrastructure can change more flexible according to different conditions and requirements (assigning more resources, etc.).

*Location independence:* services can be accessed independent of the physical location of the user and the resource.

- **Multi-tenancy** is a highly essential issue in cloud systems, where the location of code and / or data is principally unknown and the same resource may be assigned to multiple users (potentially at the same time). This affects infrastructure resources as well as data / applications / services that are hosted on shared resources but need to be made available in multiple isolated instances. Classically, all information is maintained in separate databases or tables, yet in more complicated cases information may be concurrently altered, even though maintained for isolated tenants. Multi-tenancy implies a lot of potential issues, ranging from data protection to legislator issues.

- **Security, Privacy and Compliance** is obviously essential in all systems dealing with potentially sensitive data and code.
- **Data Management** is an essential aspect in particular for storage clouds, where data is flexibly distributed across multiple resources. Implicitly, data consistency needs to be maintained over a wide distribution of *replicated* data sources. At the same time, the system always needs to be aware of the data location (when replicating across data centers) taking latencies and particularly workload into consideration. As size of data may change at any time, data management addresses both horizontal and vertical aspects of scalability. Another crucial aspect of data management is the provided consistency guarantees (eventual vs. strong consistency, transactional isolation vs. no isolation, atomic operations over individual data items vs. multiple data times etc.).
- **APIs and / or Programming Enhancements** are essential to exploit the cloud features: common programming models require that the developer takes care of the scalability and autonomic capabilities him-/herself, whilst a cloud environment provides the features in a fashion that allows the user to leave such management to the system.
- **Metering** of any kind of resource and service consumption is essential in order to offer elastic pricing, charging and billing. It is therefore a pre-condition for the elasticity of clouds.
- **Tools** are generally necessary to support development, adaptation and usage of cloud services.

#### 9.2.4 Related Areas

It has been noted, that the cloud concept is strongly related to many other initiatives in the area of the “Future Internet”, such as Software as a Service and Service Oriented Architecture. New concepts and terminologies often bear the risk that they seemingly supersede preceding work and thus require a “fresh start”, where plenty of the existing results are lost and essential work is repeated unnecessarily. In order to reduce this risk, this section provides a quick summary of the main related areas and their potential impact on further cloud developments.

##### 9.2.4.1 Internet of Services

Service based application provisioning is part of the Future Internet as such and therefore a similar statement applies to cloud and Internet of Services as to cloud and Future Internet. Whilst the cloud concept foresees essential support for service provisioning (making them scalable, providing a simple API for development etc.), its main focus does not primarily rest on service provisioning. Cloud systems are particularly concerned with providing an infrastructure on which any type of service can be executed with enhanced features. Clouds can therefore be regarded as an enabler for enhanced features of large scale service provisioning. Much research was vested into providing base capabilities for service provisioning accordingly, capabilities that overlap with cloud system features can be easily exploited for cloud infrastructures.

##### 9.2.4.2 Internet of Things

It is up to debate whether the Internet of Things is related to cloud systems at all: whilst the internet of things will certainly have to deal with issues related to elasticity, reliability and data management etc., there is an implicit assumption that resources in cloud computing are of a type that can host and / or process data – in particular storage and processors that can form a computational unit (a virtual processing platform). However, specialized clouds may e.g. integrate dedicated sensors to provide enhanced capabilities and the issues related to reliability of data streams etc. are principally independent of the type of data source. Though sensors as yet do not pose essential scalability issues, metering of resources will already require some degree of sensor information integration into the cloud. Clouds may furthermore offer vital support to the internet of things, in order to deal with a flexible amount of data originating from the diversity of sensors and “things”. Similarly, cloud



concepts for scalability and elasticity may be of interest for the internet of things in order to better cope with dynamically scaling data streams.

Overall, the Internet of Things may profit from cloud systems, but there is no direct relationship between the two areas. There are however contact points that should not be disregarded. Data management and interfaces between sensors and cloud systems therefore show commonalities.

#### 9.2.4.3 The Grid

There is an on-going confusion about the relationship between Grids and Clouds [11], sometimes seeing Grids as “on top of” Clouds, vice versa or even identical. More surprising, even elaborate comparisons (such as [9][12][13]) still have different views on what “the Grid” is in the first instance, thus making the comparison cumbersome. Indeed most ambiguities can be quickly resolved if the underlying concept of Grids is examined first: just like Clouds, Grid is primarily a concept rather than a technology thus leading to many potential misunderstandings between individual communities. With respect to research being carried out in the Grid over the last years, it is therefore recommendable to distinguish (at least) between (1) “Resource Grids”, including in particular Grid Computing, and (2) “eBusiness Grids” which centers mainly on distributed Virtual Organizations and is closer related to Service Oriented Architectures (see below). Note that there may be combination between the two, e.g. when capabilities of the eBusiness Grids are applied for commercial resource provisioning, but this has little impact on the assessment below.

**Resource Grids** try to make resource - such as computational devices and storage - locally available in a fashion that is transparent to the user. The main focus thereby lies on *availability* rather than scalability, in particular rather than *dynamic* scalability. In this context we may have to distinguish between HPC Grids, such as EGEE, which select and provide access to (single) HPC resources, as opposed to distributed computing Grids (cf. Service Oriented Architecture below) which also includes P2P like scalability - in other words, the more resources are available, the more code instances are deployed and executed. Replication capabilities may be applied to ensure *reliability*, though this is not an intrinsic capability of in particular computational Grids. Even though such Grid middleware(s) offers manageability interfaces, it typically acts on a layer on top of the actual resources and thus does rarely virtualizes the hardware, but the computing resource as a whole (i.e. not on the IaaS level). Overall, Resource Grids do address similar issues to Cloud Systems, yet typically on a different layer with a different focus - as such, e.g. Grids do generally not cater for horizontal and vertical elasticity. What is more important though is the strong conceptual overlap between the issues addressed by Grid and Clouds which allows re-usage of concepts and architectures, but also of parts of technology (see also SOA below).

Specific shared concepts:

- Virtualization of computation resources, respectively of hardware
- Scalability of amount of resources versus of hardware, code and data
- Reliability through replication and check-pointing
- Interoperability
- Security and Authentication

**eBusiness Grids** share the essential goals with Service Oriented Architecture, though the specific focus rests on integration of existing services so as to build up new functionalities, and to enhance these services with business specific capabilities. The eBusiness (or here “Virtual Organization”) approach derives in particular from the distributed computing aspect of Grids, where parts of the overall logic are located in different sites. The typical Grid middleware thereby focus mostly on achieving reliability in the *overall* execution through on-the-fly replacement and (re)integration. But eBusiness Grids also explore the specific requirements for commercial employment of service consumption and provisioning - even though this is generally considered an aspect more related to Service Oriented Architectures than to Grids.

Again, eBusiness Grids and Cloud Systems share common concepts and thus basic technological approaches. In particular with the underlying SOA based structure, capabilities may be exposed and integrated as stand-alone services, thus supporting the re-use aspect.

Specific shared concepts:

- Pay-per-use / Payment models
- Quality of Service
- Metering
- Availability through self-management

It is worth noting that the comparison here is with deployed Grids. The original Grids concept had a vision of elasticity, virtualization and accessibility [15] [16] not unlike that claimed for the Clouds vision.

#### **9.2.4.4 Service Oriented Architectures**

There is a strong relationship between the “Grid” and Service Oriented Architectures, often leading to confusions where the two terms either are used indistinguishably, or the one as building on top of the other. This arises mostly from the fact that both concepts tend to cover a comparatively wide scope of issues, i.e. the term being used a bit ambiguously. Service Oriented Architecture however typically focuses predominantly on ways of developing, publishing and integrating application logic and / or resources as services. Aspects related to enhancing the provisioning model, e.g. through secure communication channels, QoS guaranteed maintenance of services etc. come in this definition secondary. Again it must be stressed though that the aspects of eBusiness Grids and SOA are used almost interchangeably - in particular since the advent of Web Service technologies such as the .NET Framework and Globus Toolkit 4, where GT4 is typically regarded as Grid related and .NET as a Web Service / SOA framework (even though they share the same main capabilities).

Though providing cloud hosted applications as a service is an implicit aspect of Cloud SaaS provisioning, the cloud concept is principally technology agnostic, but it is generally recommended to build on service-oriented principles. However, in particular with the resource virtualization aspect of cloud systems, most technological aspects will have to be addressed at a lower level than the service layer.

Service Oriented Architectures are therefore of primary interest for a) the type of applications and services the user can build for and host on the cloud system and b) for providing additional high level services and capabilities with which to enhance the base cloud capabilities.

### **9.3 Gaps and Open Areas**

There is no full scale middleware existent which commonly addresses all cloud capabilities. What is more, not all capabilities can as yet be fulfilled to the necessary extend, even though an essential basis has been provided from both commercial and academic side. The current set of capabilities fulfils the requirements to realize simple cloud systems (as was to be expected given their availability on the market). The particular issue of interest thereby is in how far the available support fulfils the expectations towards cloud systems in their various appearances and use cases

The main gaps that can be identified relate to the following aspects:

#### **9.3.1 Technical Gaps**

##### **9.3.1.1 Manageability and Self -\***

Cloud systems focus on intelligent resource management so as to ensure availability of services through their replication and distribution. In principle, this ensures that the amount of resources consumed per service / application reflects the degree of consumption, such as access through users, size of data etc. Whilst most cloud system allow for main features related to elasticity and availability,

the management features are nowhere near optimal. Resource usage – issues are not only relevant for cost reduction, but also for meeting the green agenda and for ensuring availability when resources are limited.

Management features are mostly use-case specific at the moment and generally better at managing scale-up (e.g. when bandwidth usage exceeds a threshold) than at scale-down (mostly because the duration of inactivity is unpredictable). There is little general support in particular for new providers with respect to how to manage resources, when to scale, how to meet the requirements of the user regarding quality of service etc. This also involves self-detection of failures, of resource-shortage, but also of free load etc. and taking according actions – in particular in hybrid environments where management has to act across different resource infrastructures and can generally not be centralized. A major criterion thereby consists in improving the performance of management. Obviously, interoperability plays a major role in distributed management across resource environments, but also the capability to adapt to changes in the environment – this does not only apply to customer requirements, but also to technological restrictions, such as related to relevant libraries (IaaS & SaaS) or engines (PaaS). Adaptability and interoperability are thereby strongly linked to each other.

*Main issues:* efficiency; interoperability; compensating insufficient resources; boundary criteria.

### 9.3.1.2 Data Management

The amount of data available on the web, as well as the throughput produced by applications, sensors etc. increases faster than storage and in particular bandwidth does. There is a strong tendency to host more and more public data sets in cloud infrastructures so that improved means of managing and structuring the size of data will be necessary to deal with future requirements. Hence in particular storage clouds should be able to cater for such means in order to maintain availability of data and thus address quality requirements etc. Not only data size poses a problem for cloud systems, but more importantly consistency maintenance, in particular when scaling up. As data may be shared between tenants partially or completely, i.e. either because the whole database is replicated or indeed a subset is subject to concurrent access (such as state information), maintaining consistency over a potentially unlimited number of data instances becomes more and more important and difficult. One of the main research gaps and efforts in the area is how to provide truly transactional guarantees for software stacks (e.g. multi-tier architectures as SAP NetWeaver, Microsoft .NET or IBM WebSphere) that provides large scalability (100s of nodes) without resorting to data partitioning or relaxed consistency (such as eventual consistency). Clearly ACID 2-phase commit transactions will not work (timing) and compensating transactions will be very complex. Worse, the use of caching on distributed database systems means we have to validate cache coherency. At the moment, segmentation and distribution of data occurs more or less uncontrolled, thus not only leading to efficiency issues and (re)integration problems, but also potentially to clashes with legislation. In order to be *able* to compensate this, further control capabilities over distribution in the infrastructure are required that allow for context analysis (e.g. location) and QoS fulfillment (e.g. connectivity) - an aspect that is hardly addressed by commercial and / or research approaches so far.

As most data in the web is unstructured and heterogeneous due to various data sources, sensible segmentation and usage information requires new forms of annotation. What is more, consistency maintenance strategies may vary between data formats, which can only be compensated by maintaining meta-information about usage and structure. But also with the proprietary structures of individual cloud systems, moving data (and / or services ) between these infrastructures is sometimes complicated, necessitating new standards to improve and guarantee long term interoperability. Work on the “eXternal Data Representation” (XDR) standard for loosely coupled systems will play an important role in this context.

Cloud resources are potentially shared between multiple tenants – this does not only apply to storage (and CPUs), but potentially also to data (where e.g. a database is shared between multiple users) so that not only changes can occur at different locations, but also in a concurrent fashion. This necessitates improved means to deal with multi-tenancy in distributed data systems.

Classical data management systems break down with large numbers of nodes – even if clustered in a cloud. The latency of accessing disks means that classical transaction handling (two-phase commit) is unlikely to be sustainable if it is necessary to maintain an integral part of the system global state. Efficiency efforts (such as caching) compound the problem needing cache coherency across a very large number of nodes. As current clouds typically use centralized Storage Area Networks (e.g. Amazon EBS), unshared local disk (e.g. Amazon AMI) or cluster file-systems (e.g. GFS; but for files, not entire disk images), commodity storage (such as desktop PCs) can currently not be easily integrated into cloud storage, even though Live Mesh already allows for synchronization of local storage in / with the cloud.

In order to address these issues, the actual usage behavior with respect to file and data access in cloud systems need to be assessed more carefully. There are only few of these studies currently available, but the according information would help identifying the typical distribution, access, consistency etc. requirements of the individual use cases.

*Main issues:* data size; interoperability; control; distribution; consistency & multi-tenancy.

### **9.3.2 Privacy and Security**

Strongly related to the issues concerning legislation and data distribution is the concern of data protection and other potential security holes arising from the fact that the resources are shared between multiple tenants and the location of the resources being potentially unknown. In particular sensitive data or protected applications are critical for outsourcing issues. In some use cases, the information that a certain industry is using the infrastructure at all is enough information for industrial espionage.

Whilst essential security aspects are addressed by most tools, additional issues apply through the specifics of cloud systems, in particular related to the replication and distribution of data in potentially worldwide resource infrastructures. Whilst the data should be protected in a form that addresses legislative issues with respect to data location, it should at the same still be manageable by the system.

In addition, the many usages of cloud systems and the variety of cloud types imply different security models and requirements by the user. As such, classical authentication models may be insufficient to distinguish between the Aggregators / Vendors and the actual User, in particular in IaaS cloud systems, where the computational image may host services that are made accessible to users.

In particular in cases of aggregation and resale of cloud systems, the mix of security mechanisms may not only lead to problems of compatibility, but may also lead to the user distrusting the model due to lack of insight.

All in all, new security governance models & processes are required that cater for the specific issues arising from the cloud model (see also [17]).

See in particular Table 3 and Table 6, for issues concerning “Security and Compliance”.

*Main issues:* multi-tenancy, trust, data-encryption, legislation compliance.

### **9.3.3 Federation and Interoperability**

One of the most pressing issues with respect to cloud computing is the current difference between the individual vendor approaches, and the implicit lack of interoperability. Whilst a distributed data environment (IaaS) cannot be easily moved to any platform provider (PaaS) and may even cause problems to be used by a specific service (SaaS), it is also almost impossible to move a service /image / environment between providers on the same level (e.g. from Force.com to Amazon).

This issue is mostly caused by the proprietary data structures employed by each provider individually. History of web service standardization has shown that specifications may easily diverge rather than converge if too many parallel standardization strands are pursued. Therefore, current standardization approaches in the web service domain may prove insufficient to deal with the complexity of the problem, as it tends to be slow and diverging between multiple instances of standardization bodies. Also, interoperability is typically driven stronger by de facto standards, than by other de jure standardization efforts.

In particular cloud computing with the strong industrial drivers and the initial uptake already in place has a strong tendency to impel de-facto standards (see also vendor lock in). Traditionally, US –with an emphasis on software innovation - favor a voluntary, market driven approach to standardization. Europe, with a strong track record in telecom standardization, seems to favor an upfront approach – albeit mostly in hardware related fields. While innovations between domains usually benefit from an early focus on interoperability, the quest for disruptive innovations within domains benefits from a lower focus on interoperability requirements in this early phase. Too early focus on interoperability and standardization issues may therefore be disruptive as e.g. long-term requirements and structures cannot be assessed to their full extend today, and a bad specification may hinder interoperable development accordingly. A particular focus must hence rest on atomic, minimal, composable and adaptable standards.

While nobody is questioning the usefulness and benefit of interoperability, it should also be noted that careful consideration is necessary in which fields and when those steps provide the biggest benefit. New policies and approaches may therefore be needed to ensure convergence and thus achieve real interoperability rather than adding to the issue of divergence.

Federation and Interoperability are issues relevant for many capabilities, but in particular for “Data Management” and “Virtualization”, as well as aspects related to “Cost Reduction” and “Improved Time to Market”.

*Main issues:* proprietary structures / de-facto standards; vendor lock-in.

### **9.3.4 Virtualization, Elasticity and Adaptability**

Though virtualization techniques have improved considerably over recent years, additional issues arise with the advent of cloud systems that have not been fully elaborated before – in particular related to the elasticity of the system (horizontal and vertical up- and down-scaling), interoperability and manageability & control of the resources. Changes in the configuration of the service / data need to be reflected by the setup of the underlying resources (according to their capabilities and capacities), but also changes in the infrastructure need to be exploited by the virtual environment without impacting on the hosted capabilities. For example, if another CPU is added to a virtual machine, the running code should make use of the additional resource without having to be restarted or even adapted. This obviously relates to the issue of programming models and resource control – it should be noted in this context that actual resource integration in virtual machines is less an issue than developing applications that actually exploit such dynamic changes.

To provide efficient elasticity that is capable of respecting the QoS and green requirements new, advanced scheduling mechanisms are required that also take the multi-tenancy aspect into consideration. For example, it may be more sensible to delay execution if resources will be available shortly, so as to avoid the employment of currently powered-down resources etc. Virtualization (and to a degree scheduling) have to take the human factor into consideration thereby: the degree of interaction with cloud systems, as well the increasing connectivity will require that the systems are capable to integrate humans not only as users, but also as an extended resource that can provide services, capabilities and data.

Currently, also little support is available for cross-platform execution and migration which global cloud structures will require (with the exception of specialized “niche” cloud systems). Especially, the movement of (parts of) an application between cloud structures (e.g. from private cloud to public cloud and back) is a key issues that is not supported yet.

All these capabilities will require a stronger “self-\*” awareness of the resources and the virtual environment involved, so as to improve the adaptability to changes in the environment and thus maintain boundary conditions (such as QoS and business policies). And, of course, implicitly new models are to develop according applications and tools that can easily exploit these features.

*Main issues:* elasticity; optimized scheduling; interoperability; resource manageability; rapidly changing workloads.

### **9.3.5 Apis, Programming Models and Resource Control**

Cloud virtual machines tend to be built for fixed resource environments, thus allowing horizontal scalability (instance replication) better than vertical scalability (changes in the resource structure) – however, future systems will have to show more flexibility with this respect to adapt better to requirements, capabilities and of course green issues. In addition, more fine grained control over e.g. distribution of data etc. must be granted to the developer in order to address legislation issues, but also to exploit specific code requirements. Cloud systems will thus face similar issues that HPC has faced before with respect to description of connectivity requirements etc., but also to ensure reliability of execution, which is still a major obstacle in distributed systems. At the same time, the model must be simple enough to be employed by average developers and / or business users. Cloud systems provide enhanced capabilities and features, ranging from dynamically scalable applications and data, over controlled distribution to integration of all types of resources (including humans). In order to exploit these features during development of enhanced applications and services, the according interfaces and features need to be provided in an easy and intuitive fashion for common users, but should also allow for extended control for more advanced users. In order to facilitate such enhanced control features, the cloud system needs to provide new means to manage resources and infrastructure, potentially taking quality of service, the green agenda and other customer specifications into consideration. Development support for new “cloudified” applications has to ensure movability of application (segments) across the network, enabling a more distributed execution and communication model within and between applications.

Since cloud applications are likely to be used by much more tenants and users than non-cloud applications (“long tail”), customizability must be considered from the outset. The issue applies equally to distributed code, as to distributed data. Data is expected to become exceedingly large (see “Data Management” above) - hence an interesting approach in cloud system’s code management consists in moving the software to the data, rather than the other way round, since most code occupies less space than the data they process. However this is intrinsically against the current trend for clouds to be provided in remote data centers with code and data co-existing.

*Main issues:* connectivity; intelligent distribution (code & data); multi-tenancy; enhanced manageability; reliability; ease of use; development and deployment support.

## **9.4 Non-Technical Gaps**

### **9.4.1 Legislation, Government and Policies**

Not only data is subject to specific legislation issues that may depend on the location they are currently hosted in, but also applications and services, in particular regarding their licensing models. Legislation issues arise due to the fact that different countries put forward different laws regarding which kind of data is allowed, but also which data may be hosted where. With the cloud principally hosting data / code anywhere within the distributed infrastructure, i.e. potentially anywhere in the world, new legislative models have to be initiated, and / or new means to handle legislative

constraints during data distribution. Related to that, governance of clouds needs to be more open to the actual user who needs to be able to specify and enforce his / her requirements better, such as data privacy issues, issues caused by business (process) requirements and similar. Governance solution could also help to select only those vendors providing open-source solutions, thus avoid vendor lock in.

Clouds generally benefit from the economic globalization so that providers (and implicitly users) can make use of cheaper resources in other countries etc. Hence, similar issues apply to clouds that apply to the global market and new policies are required to deal with jurisdiction, data sovereignty and support for law enforcement agencies new cross-country regulation have to be enacted etc.

*Main issues:* legislation; governance; licensing; globalization.

#### **9.4.2 Economic Concerns**

In order to provide a cloud infrastructure, a comparatively high amount of resources needs to be available, which implies a considerable high investment for start-up. As it is almost impossible to estimate the uptake and hence the profit of services offered to the customers, it remains difficult to assess the return of investment and hence the sensible amount of investment to maximize the profit. With the cloud outsourcing principle being comparatively new on the market, new knowledge about business models, market situation, how to extract value and under what conditions etc. are required – in other words, new expert systems and best use recommendations are required. This also includes issues related to the “green agenda”, namely policies basing on dedicated benchmarks under what circumstances to reduce resource usage and / or switch between different power settings etc. This implies new scheduling mechanisms that weigh green vs. business (profit & quality) issues. In a cloud environment it would be possible to improve ‘green’ credentials by utilizing more efficient processors and memory. A few large data centers with clouds are likely to be more ‘green’ than millions of smaller but already large data centers. Notably, from a global perspective, sharing resources may be greener than down-powering idle resources, if this reduces their production (and hence the according carbon footprint) in the first instance.

In general, business control is principally possible, yet linkage between the technical and economical perspective is still weak and hence maintenance of e.g. service quality respecting the economical descriptions still requires improvement. An indirect economical issue that will have to be solved through e.g., means for improved interoperability consists in the current tendency towards vendor-lock in. Most vendors want to maintain this status in order to secure their customer base, yet with scope and competition growing in the near future, it is to be expected that even larger vendors will adopt more interoperable approaches. As a side note it should be mentioned that already some major key player are basing their system on more standard based approaches, such as MS Azure.

*Main issues:* extended business knowledge; improved QoS management; Green Agenda; energy proportional computing.

#### **9.4.3 Business Models and Expert Systems**

Extracting value from cloud system employment is not always straight forward, as it depends on the cost and effort to be invested first versus the (potential) gain from the employment of such a system. There is little knowledge so far about when and under what circumstances to move to a (public or private) cloud, respectively when to distribute capabilities in a hybrid cloud. Though outsourcing to clouds can reduce start up time and makes better use of resources due to the elasticity of the infrastructure, the additional effort to move services and large datasets into a new environment, as well as the risk to lose control over the system, makes such a movement a considerable business decision. As long as interoperability is at a stage where no simple movement from local to cloud platforms is possible additional knowledge is required to support such decisions and in the long run allow for autonomic management of outsourcing and reconfiguration decisions.

## 9.5 REFERENCES & SOURCES

- 1 Malis, A. (1993), 'Routing over Large Clouds (ROLC) Charter', part of the 32nd IETF meeting minutes' - available at <http://www.ietf.org/proceedings/32/charters/rolc-charter.html>
- 2 New York Times (2001), 'Internet Critic Takes on Microsoft' - available at <http://www.nytimes.com/2001/04/09/technology/09HAIL.html?ex=1217563200&en=7c46bdefb6a8450a&ei=5070>
- 3 Wikipedia, 'John McCarthy (computer scientist)' - available at [http://en.wikipedia.org/wiki/John\\_McCarthy\\_\(computer\\_scientist\)](http://en.wikipedia.org/wiki/John_McCarthy_(computer_scientist))
- 4 Barr, J. (2006), 'Amazon EC2 Beta' - available at [http://aws.typepad.com/aws/2006/08/amazon\\_ec2\\_beta.html](http://aws.typepad.com/aws/2006/08/amazon_ec2_beta.html)
- 5 Sutter, H. (2005), 'The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software', in *Dr. Dobb's Journal*, **30**(3).
- 6 Wikipedia, 'The Prosumers' - available at <http://en.wikipedia.org/wiki/Prosumer>
- 7 Wikipedia, 'Cloud Computing' - available at [http://en.wikipedia.org/wiki/Cloud\\_computing](http://en.wikipedia.org/wiki/Cloud_computing)
- 8 Golden, B. (2009), 'Capex vs. Opex: Most People Miss the Point About Cloud Economics' -available at [http://www.cio.com/article/484429/Capex\\_vs.\\_Opex\\_Most\\_People\\_Miss\\_the\\_Point\\_About\\_Cloud\\_Economics](http://www.cio.com/article/484429/Capex_vs._Opex_Most_People_Miss_the_Point_About_Cloud_Economics)
- 9 Fellows, W. (2009), 'The State of Play: Grid, Utility, Cloud' - available at [http://old.ogfeurope.eu/uploads/Industry%20Expert%20Group/FELLOWS\\_CloudscapeJan09-WF.pdf](http://old.ogfeurope.eu/uploads/Industry%20Expert%20Group/FELLOWS_CloudscapeJan09-WF.pdf)
- 10 Sims, K. (2009), 'IBM Blue Cloud Initiative Advances Enterprise Cloud Computing' - available at <http://www-03.ibm.com/press/us/en/pressrelease/26642.wss>
- 11 Foster, I. (2008), 'Cloud, Grid, what's in a name?' - available at <http://ianfoster.typepad.com/blog/2008/08/cloud-grid-what.html>
- 12 Members of EGEE-II (2008), 'An egee comparative study: Grids and clouds - evolution or revolution. Technical report, Enabling Grids for E-science Project' - available at <https://edms.cern.ch/document/925013/>.
- 13 Vaquero, L. M.; Rodero-Merino, L.; Caceres, J. & Lindner, M. (2009), 'A break in the clouds: towards a cloud definition', *SIGCOMM Comput. Commun. Rev.* **39**(1), 50--55.
- 14 Rochwerger, R; Caceres, J; Montero, RS; Breitgand, D; Elmroth, E; Galis, A; Levy, E; Llorente, IM; Nagin, K & Wolfsthal, Y (2009), 'The RESERVOIR Model and Architecture for Open Federated Cloud Computing'. *IBM Systems Journal*, September 09
- 15 Foster, I (1998), 'The Grid: Blueprint for a New Computing Infrastructure', Morgan Kaufmann Publishers
- 16 Next Generation GRIDs Expert Group (2003), 'Next Generation GRIDs: European Grid Research 2005-2010', available at [ftp://ftp.cordis.lu/pub/ist/docs/ngg\\_eg\\_final.pdf](ftp://ftp.cordis.lu/pub/ist/docs/ngg_eg_final.pdf)
- 17 Catteddu, D; Hogben, G eds. (2009), 'Cloud Computing - Benefits, risks and recommendations for information security', European Network and Information Security Agency (ENISA) – available at <https://www.enisa.europa.eu/activities/risk-management/files/deliverables/cloud-computing-risk-assessment>

[This document is a shortened version of the report "THE FUTURE OF CLOUD COMPUTING: OPPORTUNITIES FOR EUROPEAN CLOUD COMPUTING BEYOND 2010, Expert Group Report, Public Version 1.0 of European Commission".]



