A CLOUD ARCHITECTURE FOR REDUCING COSTS IN LOCAL PARALLEL AND DISTRIBUTED VIRTUALIZED CLOUD ENVIRONMENTS

by

JEFFREY MICHAEL GALLOWAY

Dr. SUSAN VRBSKY, COMMITTEE CHAIR
Dr. XIAOYAN HONG
Dr. JOHN LUSTH
Dr. RANDY SMITH
Dr. FEI HU

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of The University of Alabama

TUSCALOOSA, ALABAMA

2013
ABSTRACT

Deploying local cloud architectures can be beneficial to organizations that wish to maximize their available computational and storage resources. Many users are reluctant to move their computational and storage needs to a public cloud vendor. While designing scalable local cloud architectures, power requirements should be given adamant attention. This dissertation focuses on several challenging concerns relating to cloud computing architectures, specifically lowering the power requirements of Infrastructure-as-a-Service (IaaS) local cloud architectures. These challenges include power efficient computational resource load consolidating, power efficient persistent cloud storage consolidating, and deploying a local IaaS cloud architecture with limited networking resources.

The design of a load consolidation approach to Infrastructure-as-a-Service cloud architectures that is power efficient is presented in this dissertation. A proposed Power Aware Load Consolidation algorithm, PALC, maintains the state of all compute nodes, and based on utilization percentages, decides the number of compute nodes that should be operating. Results show that PALC provides adequate availability to compute node resources while decreasing the overall power consumed by the local cloud architecture.

Persistent storage is a necessity in cloud computing architectures. Since the goal of this local cloud architecture design is to deploy resources using minimum power consumption, a power aware persistent storage consolidation algorithm is presented in this dissertation. The Power Aware Storage Consolidation algorithm, PASC, dynamically determines the number or active persistent
storage nodes based on the number of active users. This algorithm, combined with the PALC algorithm will significantly decrease the power consumed by the local cloud architecture.

Realizing the implications of deploying a local cloud system in an environment with limited networking resources (IP addresses), a solution is needed to allow users to connect with only one public IP address. Users will be able to access cloud resources through a simple web interface and maintenance of the cloud will be contained with private networking resources. Also introduced is the ability to scale to have multiple geographically distributed clusters in the local cloud using only one IP address per cluster. This dissertation provides a comprehensive solution for deploying a local cloud architecture that is cost efficient to maintain.
## LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3DES</td>
<td>Triple Data Encryption Standard</td>
</tr>
<tr>
<td>3G</td>
<td>3rd Generation</td>
</tr>
<tr>
<td>4G</td>
<td>4th Generation</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>AMD-V</td>
<td>AMD Virtualization Extensions</td>
</tr>
<tr>
<td>APAC</td>
<td>Asia Pacific</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARPANET</td>
<td>Advanced Research Projects Agency Network</td>
</tr>
<tr>
<td>ATX</td>
<td>Advanced Technology Extended</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>CC</td>
<td>Cluster Controller</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Network</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>Compact Disc Read Only Memory</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CIFS</td>
<td>Common Internet File System</td>
</tr>
<tr>
<td>CLC</td>
<td>Cloud Controller</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>$CPU_{util}$</td>
<td>CPU Utilization (%)</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Repeat</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Control Protocol</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>$DP_r$</td>
<td>Dirty Page Rate</td>
</tr>
<tr>
<td>DRY</td>
<td>Don’t Repeat Yourself</td>
</tr>
<tr>
<td>DSL</td>
<td>Digital Subscriber Line</td>
</tr>
<tr>
<td>$E_w$</td>
<td>Power Consumed During VM Migration</td>
</tr>
<tr>
<td>EBS</td>
<td>Elastic Block Storage Controller</td>
</tr>
<tr>
<td>EC2</td>
<td>Elastic Compute Cloud</td>
</tr>
<tr>
<td>EioP</td>
<td>Ethernet over IP</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>Eucalyptus</td>
<td>Elastic Utility Computing Architecture Linking Your Programs to Useful Systems</td>
</tr>
<tr>
<td>FAH</td>
<td>Folding at Home</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>Gbps</td>
<td>Gigabits per second</td>
</tr>
<tr>
<td>GFS</td>
<td>Google File System</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HaaS</td>
<td>Hardware as a Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>HDD</td>
<td>Hard Disk Drive</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper-text Markup Language</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>ID</td>
<td>Identity</td>
</tr>
<tr>
<td>IMFS</td>
<td>Internet Media File System</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPv4</td>
<td>Internet Protocol Version 4</td>
</tr>
<tr>
<td>IPv6</td>
<td>Internet Protocol Version 6</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>KVM</td>
<td>Kernel-based Virtual Machine</td>
</tr>
<tr>
<td>kWh</td>
<td>KiloWatt-hours</td>
</tr>
<tr>
<td>$L_{ab}$</td>
<td>Link Bandwidth</td>
</tr>
<tr>
<td>L20</td>
<td>20 large virtual machines</td>
</tr>
<tr>
<td>L30</td>
<td>30 large virtual machines</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LTS</td>
<td>Long Term Support</td>
</tr>
<tr>
<td>M</td>
<td>Mean Time Between Two Migration of the Same Virtual Machines</td>
</tr>
<tr>
<td>$M_{tms}$</td>
<td>Total Time for Migration</td>
</tr>
<tr>
<td>MB/s</td>
<td>Megabytes per second</td>
</tr>
<tr>
<td>Mbps</td>
<td>Megabits per second</td>
</tr>
<tr>
<td>$MeM_{mig}$</td>
<td>Total amount of Memory transmitted from source to destination</td>
</tr>
</tbody>
</table>
MySQL  My Structured Query Language

$N_u$  Network utilization

NAS  Network Attached Storage

NAT  Network Address Translation

NC  Node Controller

NFS  Network File System

NIST  National Institute of Standards and Technology

OCCI  Open Cloud Computing Interface

OSI  Open Systems Interconnection

$P_i$  Increase in Process Execution Time During VM Migration

$P_{th}$  Decrease in Process Throughput During VM Migration

PaaS  Platform as a Service

PALC  Power Aware Load Consolidation

PASC  Power Aware Storage Consolidation

PC  Personal Computer

PDA  Personal Data Assistant

Pflops  Peta Floating Point Operations per Second

PHP-CLI  PHP-Command Line Interface

PHP  PHP: Hypertext Preprocessor

QOE  Quality of Experience

QOS  Quality of Service

R20  20 random sized virtual machines
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R30</td>
<td>30 random sized virtual machines</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RDP</td>
<td>Remote Desktop Protocol</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RPM</td>
<td>Revolutions Per Minute</td>
</tr>
<tr>
<td>RRLB</td>
<td>Round Robin Load Balancer</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest, Shamir, Adleman (encryption algorithm)</td>
</tr>
<tr>
<td>S20</td>
<td>20 small virtual machines</td>
</tr>
<tr>
<td>S3</td>
<td>Simple Storage Service</td>
</tr>
<tr>
<td>S30</td>
<td>30 small virtual machines</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SATA</td>
<td>Serial Advanced Technology Attachment</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SDN</td>
<td>Storage Delivery Network</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSD</td>
<td>Solid State Disk</td>
</tr>
<tr>
<td>SSH</td>
<td>Secure Shell</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TB</td>
<td>Terabyte</td>
</tr>
<tr>
<td>UEC</td>
<td>Ubuntu Enterprise Cloud</td>
</tr>
<tr>
<td>UFW</td>
<td>Uncomplicated Firewall</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
</tbody>
</table>
US      United States
USD     United States Dollar
$V_{ins}$ Total Inaccessible Time for VM
VM      Virtual Machine
$VM_{mem}$ Virtual Machine Memory Size
VPN     Virtual Private Network
VT      Virtualization Extensions
W       Watt
WAN     Wide Area Network
Wh      Watt-hour
WS3     Walrus Storage Controller
XHTML   Extensible Hyper-text Markup Language
XL20    20 extra large virtual machines
XL30    30 extra large virtual machines
XML     Extensible Markup Language
XSLT    Extensible Stylesheet Language Transformations
DEDICATION

To my mother, father, and sister.

&

To the pioneers of computer science that paved the way before me.

“We can only see a short distance ahead, but we can see plenty there that needs to be done.”

— Alan Turing, Father of Computer Science

“If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility.”

— John McCarthy, Father of Cloud Computing
ACKNOWLEDGMENTS

My appreciation goes to my professor and adviser, Dr. Susan Vrbsky. I am very thankful for Dr. Vrbsky’s involvement in my academic and professional life for the past three years. During my time here Dr. Vrbsky helped me become a better computer science researcher and teacher. When I applied to the Ph.D. program, Dr. Vrbsky was the first to contact me about joining a research group that would evolve into the Cloud and Cluster Computing Group here at the University of Alabama. As I was progressing through the program, Dr. Vrbsky was able to send me to many great conferences around the United States. Even though most of our interaction involved research topics, I also learn much about the aspects of working in the academic environment. Those insights are invaluable and would be difficult to find anywhere else. As I progress through my career, I plan on remaining in contact with the person that most influenced the direction of my life. Thank you, Dr. Vrbsky, for motivating and helping me through these years of research and teaching.

I would like to express my gratitude to Dr. Xiaoyan Hong. Dr. Hong served as my advisor during my time here as a master’s student. My undergraduate focus was in computer networking so it was obvious that I would take her as an adviser. During my time as a master’s student, Dr. Hong introduced me to many computer network tools and applications. As I moved to local cloud computing infrastructure research, computer networking is still a major factor to master. Dr. Hong has always been a great source of information when I faced tough networking problems.

To Dr. Randy Smith, who was my first contact arriving at the University of Alabama. Thank you for teaching the Software Engineering course and Human Computer Interaction course, both
of which helped me prevail in my studies of cloud computing. I will always remember the first remark of noticing me in Houser Hall with my orange and blue backpack.

To Dr. John Lusth, my teaching advisor. I am glad to have had the opportunity to teach the Introduction to Programming class here for so many semesters. This has undoubtedly been one of the best experiences I had during my time as a Ph.D. student. I believe my approach in teaching students has become significantly improved by Dr. Lusth. Also, for some unknown reason, I now believe that every text editor and web page should respond to VIM commands.

To my other committee members, I appreciate your cooperation and support during in my time here as a Ph.D. student. Dr. Jeffrey Carver, thank you for giving me the insight of applying an empirical approach to my research of cloud computing. Dr. Marcus Brown, thank you for always providing a positive aspect on any situation we discussed. Dr. Yang Xiao, thank you for your insights on wireless networks and security.

To David Engle, thank you for sharing your knowledge in systems administration. This is an interesting field to me, which is shown by the fact that this dissertation is inherently an applied systems research.

To Kathy DeGraw and Jamie Thompson, thank you for everything that you do to make this department what it is! We would all surely fall without your support.

I am also privileged to be associated with the current cloud and cluster computing group at UA: Gabriel Loewen, Jeffrey Robinson, Ashfakul Islam, and Kazi Zunnurhain. I really enjoyed working with this group and will always remember my time in this lab. Just be sure to make good use of the decommissioned “Fluffy”!

To others in the Ph.D. program, Dr. Ferosh Jacob, Brian Eddy, thank you for your discussions about computer science and software engineering topics, and approaches we should make
in relaying these topics to students in the classes we teach here at UA. Also, I am indebted to Dr. Jacob for his vast knowledge on \LaTeX!

I would also like to think Darrell Wright, department head at Shelton State, for his unique insight on teaching students about computer science. I will always remember my time at Shelton, and Mr. Wrights very unique personality with vast knowledge on many subjects and thousands of jokes!

Finally, I would like to thank the financial support provided by the Department of Computer Science at UA. Thank you for the opportunity to pursue my Ph.D. in the area of local cloud computing and strengthen my abilities as a computer science instructor.
## CONTENTS

ABSTRACT ................................................................. ii

LIST OF ABBREVIATIONS ........................................ iv

DEDICATION ........................................................... x

ACKNOWLEDGMENTS ................................................ xi

LIST OF TABLES ...................................................... xxi

LIST OF FIGURES ................................................... xxii

1 INTRODUCTION .................................................... 1

1.1 Types of Clouds ................................................ 4

1.2 Implementation Considerations of a Local Cloud ......... 4

1.3 Local Data Center Design ..................................... 6

1.4 Outline .......................................................... 7

2 BACKGROUND AND LITERATURE REVIEW ................. 9

2.1 Cloud Architecture ............................................ 11

2.1.1 Client Layer ............................................... 11

2.1.2 Application Layer ....................................... 15

2.1.3 Platform Layer ........................................... 18

2.1.4 Infrastructure Layer ..................................... 19

2.1.5 Hardware Layer .......................................... 20

2.2 Cloud Access .................................................. 20
2.2.1 Cloud Platforms ................................................. 20
2.2.2 Web Application Frameworks ............................. 21
2.2.3 Cloud Hosting Services ..................................... 23
2.2.4 Cloud Applications ......................................... 24
2.2.5 Cloud APIs .................................................. 24
2.2.6 Web Browsers ............................................... 25

2.3 Cloud Storage .................................................. 26
  2.3.1 Storage as a Service ................................. 26
  2.3.2 Storage Security ...................................... 27
  2.3.3 Cloud Storage Reliability ......................... 29
  2.3.4 Advantages of Cloud Storage ...................... 29
  2.3.5 Disadvantages of Cloud Storage ................. 32
  2.3.6 Amazon S3 ............................................ 33
  2.3.7 Nirvanix Storage Delivery Network .............. 35
  2.3.8 Google Bigtable Datastore ........................ 36
  2.3.9 Apple Mobile Me .................................... 37

2.4 Local Clouds .................................................. 37
  2.4.1 Eucalyptus ............................................. 38
  2.4.2 Eucalyptus Implementation ........................ 41

2.5 Power Consumption in Cloud Architecture .................. 42

2.6 Interfacing and Deploying the Cost Effective Cloud Architecture .................. 45

2.7 Current Research in Power Management for Persistent Storage in Cloud Architectures .................. 49
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Average Number of Power Cycles Per Day.</td>
<td>58</td>
</tr>
<tr>
<td>3.2</td>
<td>PALC Terminology.</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>Virtual Machine Configuration Options.</td>
<td>65</td>
</tr>
<tr>
<td>3.4</td>
<td>Power Savings of PALC as Compared to RRLB.</td>
<td>76</td>
</tr>
<tr>
<td>3.5</td>
<td>Average Power Used by PALC as Compared to RRLB Over All Job Schedules and Available Compute Nodes.</td>
<td>76</td>
</tr>
<tr>
<td>4.1</td>
<td>Virtual Machine Live Migration Cost.</td>
<td>82</td>
</tr>
<tr>
<td>4.2</td>
<td>Virtual Machine Descriptions.</td>
<td>88</td>
</tr>
<tr>
<td>4.3</td>
<td>Virtual Machine Performance Evaluations.</td>
<td>95</td>
</tr>
<tr>
<td>5.1</td>
<td>PASC Terminology.</td>
<td>135</td>
</tr>
<tr>
<td>5.2</td>
<td>PASC Power Consumption Percentage vs Always on Storage Controllers (Typical Schedule).</td>
<td>144</td>
</tr>
<tr>
<td>5.3</td>
<td>PASC Power Consumption vs Always on Storage Controllers (Random Schedule).</td>
<td>146</td>
</tr>
<tr>
<td>6.1</td>
<td>EoIP Setup.</td>
<td>164</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

1.1 Local, Public, and Hybrid Clouds. .................................................. 5
2.1 Cloud Architectural Layers. ............................................................ 11
2.2 SaaS Application Being Served to a Client Device. ............................ 15
2.3 S3 Bucket with Versioning Enabled. ................................................. 34
2.4 Local Cloud Architecture of Fluffy. .................................................. 42
3.1 Eucalyptus Architecture. ............................................................... 56
3.2 PALC Algorithm. ........................................................................... 60
3.3 Power Requirements for Individual Compute Nodes. ............................ 62
3.4 Simulator Design. .......................................................................... 64
3.5 Virtual Machine Request Schedule. .................................................. 67
3.6 Comparison of Small VM Requests. .................................................. 68
3.7 Comparison of Large VM Requests. ................................................... 69
3.8 Comparison of Extra-Large VM Requests. ......................................... 70
3.9 Comparison of Incrementally Sized VM Requests. .............................. 71
3.10 Comparison of VM Requests. ........................................................... 72
3.11 Comparison of Round Robin VM Requests. ....................................... 75
3.12 Power Requirements Comparison Between PALC and RRLB. .......... 77
4.1 Overview of Live Migration Performing Iterative Memory Transfers. .... 81
4.2 Overview of Virtual Machine Live Migration Costs. ........................... 83
4.3 Performance of Migrating the Base Instance (Lower is better) .......................... 90
4.4 Performance of Migrating the Web Instance While Idle (Lower is better) ........ 91
4.5 Performance of Migrating the Web Instance (Lower is better) ......................... 93
4.6 Performance of Migrating the Database Instance (Lower is better) .................. 94
4.7 Performance of Migrating the Pi Computation Instances (Lower is better) ........ 94
4.8 Base VM CPU Utilization .................................................................................. 96
4.9 Base VM RAM Utilization .................................................................................. 97
4.10 Web1 Response Time (Lower is better) ............................................................. 98
4.11 Web2 Response Time (Lower is better) ............................................................. 99
4.12 Web3 Response Time (Lower is better) ............................................................. 100
4.13 Web Server Performance Comparison ............................................................. 102
4.14 Performance of DB1 Database Virtual Machine ................................................. 103
4.15 Performance of DB2 Database Virtual Machine ................................................. 104
4.16 Performance of DB3 Database Virtual Machine ................................................. 105
4.17 Performance Comparison of Database Virtual Machines ................................... 106
4.18 Performance of Pi1 Pi Computation Virtual Machine ....................................... 107
4.19 Performance of Pi2 Pi Computation Virtual Machine ....................................... 108
4.20 Performance of Pi3 Pi Computation Virtual Machine ....................................... 109
4.21 Performance Comparison of Pi Computation Virtual Machines ....................... 110
4.22 Base Performance Comparison ........................................................................ 111
4.23 Web Server Performance Comparison ............................................................. 113
4.24 Power Consumed While Migrating Web1 ......................................................... 114
4.25 Power Consumed While Migrating Web2........... 115
4.26 Power Consumed While Migrating Web3........... 116
4.27 Comparison of Power Consumed while Migrating Virtual Web Servers. .... 117
4.28 Power Consumed while Migrating DB1 Virtual Database Server.......... 118
4.29 Power Consumed while Migrating DB2 Virtual Database Server.......... 118
4.30 Power Consumed while Migrating DB3 Virtual Database Server.......... 119
4.31 Comparison of Power Consumed while Migrating Virtual Database Servers. .. 120
4.32 Power Consumed while Migrating Pi1 Pi Computation Server............. 121
4.33 Power Consumed while Migrating Pi2 Pi Computation Server............. 122
4.34 Power Consumed while Migrating Pi3 Pi Computation Server............. 123
4.35 Comparing Power Consumed while Migrating the Pi Computation Servers. .. 124
5.1 Topology of the Local Cloud Architecture.................. 131
5.2 PASC Algorithm................................................. 137
5.3 Job Schedule of VM Requests.................................. 142
5.4 Comparing Power Consumed PASC vs Always On (Typical Schedule)... 142
5.5 PASC Power Consumption vs Always on (Typical Schedule)... 143
5.6 Comparing Power Consumed PASC vs Always On (Random Schedule)... 145
5.7 PASC Efficiency vs Always On (Random Schedule)... 146
6.1 Log.txt File Located on Cloud Controller.................. 153
6.2 Example of Log.txt file With Instantiated VM’s.......................... 154
6.3 User Interface Creation/Deletion of VM’s.......................... 156
6.4 Login Screen..................................................... 157
6.5 Initializing a New VM Instance. ................................. 159
6.6 Running VM Instances. ........................................ 159
6.7 Persistent Storage Information. .............................. 160
6.8 Distributed Cloud Architecture. ............................. 162
6.9 Example of Available VM’s in Different Cloud Clusters. 163
Chapter 1

INTRODUCTION

The general concept of cloud computing can take many forms depending on the functionality the cloud provides to the user. A user could be a customer who rents a virtual server, loads software on it, and maintains it in place of a physical server in their building. A user could also be a computer gamer using Steam [Valve, 2011] to maintain their game purchases in the cloud so they do not have to keep up with discs and determine game processing specifications. It could be a university using services provided by a vendor, such as Google, to maintain the students’ email accounts. It could be iPhone users, Android users, students, teachers, vendors, and customers, anyone who needs information in real time, processed somewhere other than on the local client, and distributed (somewhat) seamlessly to devices of their choice. The first thing we should ask is, “what is cloud computing?” Depending on the perspective of the person asked, cloud computing could have many meanings. The cloud computing model was derived from the combination of many past and current computer and network architectures. Even though there is no current standard definition of cloud computing, the overall consensus leans towards providing users with elastic computational, storage, and network resources through web-based applications with the goal of reduced complexity and an easy to use interface.

This dissertation considers the technology needed to bring the idea of cloud computing to the user using power aware computational and storage load consolidating strategies with limited networking resources. This dissertation also provides an architecture for organizations that wish
to build a cloud to do so using commodity computers and networking equipment. The design of this architecture is inherently heterogeneous, giving it the ability to scale easily. The implementation in this research will focus on providing computing and storage resources to users in terms of virtual machines. This is different than other forms of cloud computing which may give the user a platform for development, or software applications for them to use. Organizations implementing a local cloud architecture can also benefit from using energy efficient persistent storage techniques. Persistent storage is necessary in a cloud environment, just as it is needed in non-cloud architectures. Housing the equipment for implementing a local cloud is also an important concern for an organization. This dissertation will cover best practice small to medium sized cloud designs used to reduce the cost of owning a local cloud architecture. These practices include reducing power consumption on the server level through designs for racks and heat dissipation.

Cloud computing can be broken down into three categories: clients, datacenters, and servers [Chappell, 2008] . Client devices in a cloud computing setup are exactly the same as clients in any other network architecture. These are devices, such as a typical desktop computer, laptop, tablet computer, smart phone, or thin client. Clients, such as smart phones and PDAs, are becoming more popular due to their increase in performance and ease of portability. A datacenter is the physical infrastructure of the cloud provider. This datacenter could be private, such as a datacenter of a company or university, or it could be public, such as a web hosting or data backup company. The final layer is made up of the servers, which are the physical machines used in the datacenter. Due to the distributed nature of a cloud environment, these servers can be geographically dispersed, so long as they are able to communicate with one another using the Internet. Given this advantage, cloud providers have the ability to add physical machines at any datacenter within their infrastructure and cloud subscribers would be oblivious to the change.
Cloud computing, in terms of data throughput, storage, and scalability could be an adversary to conventional supercomputers. Supercomputers are generally thought of as massive machines filling entire buildings with thousands, or hundreds of thousands, of processors working together to complete dauntingly complex computational tasks. These supercomputers are typically not used as virtual process distributors to client machines; rather they are used for weapons simulations, weather forecasting, submarine tracking, molecular biology, fluid dynamics, and cryptology. Cloud computing can also accomplish these goals, at a fraction of the cost to the organization. The Cray XT5 Jaguar owned by Oak Ridge National Laboratory can process data at 1.75 PFlops and cost the organization $104 million to build [Mike Bradley, 2009]. Cloud architectures, on the other hand can also reach and exceed this performance factor. Folding@home (FAH) [University, 2011], a project based at Stanford University, has the ability to reach and exceed 10 Pflops using around 200,000 active clients. The benefit to projects, such as FAH, is that they do not have to buy the many client machines needed to achieve this data throughput. They also are not responsible for maintaining or paying utility costs for these client machines. The drawback though, is that the speed of this type of network is much slower in moving the unprocessed data to the clients and the processed data back to FAH. This is just one instance where an organization may find cloud architectures to be more beneficial than a traditional server/supercomputer setup by not having to maintain the compute nodes. Again, this is not the only type of cloud architecture. There are many different organizations that offer cloud alternatives to traditional architectures. Chapter two will cover some of the leaders in the cloud-computing paradigm, including Amazon, Google, Microsoft, and Eucalyptus.
1.1 Types of Clouds

As a user, there are many options to consider when deploying a cloud-based solution. Public clouds are generally provided by commercial vendors competing with other vendors for market share. Private clouds are created to be used by a single organization. They can be located within that organization or housed off-site. Local clouds are created by organizations or users to provide services to multiple users. Local clouds are found and maintained within an organization. Typically, public cloud vendors market themselves based on resources offered, availability, security, and price. Organizations or users can opt to use a public cloud vendor after considering the potential advantages and disadvantages of having services provided off-site at a collocation facility. Also available are software suites aimed at creating local clouds. The user can create a local cloud that manages resources and would be available just like the public vendor’s clouds, but they would be in total control of the physical resources as well. With a local cloud, users can create applications and policies tailored to their own needs. Security and maintenance would also be governed by the organization, as their cloud will be physically located in their building. As the organization’s cloud specific needs grow, they have the option of creating a hybrid cloud. The organization will expand their local cloud to encompass a public vendor’s cloud resources as well. This happens when a local cloud’s resources are maximized, or when off site data replication is needed. Figure 1.1 illustrates the idea of local, public, and hybrid cloud computing architectures.

1.2 Implementation Considerations of a Local Cloud

The idea of moving data to off-site cloud vendors does not appeal to all organizations. In this case, they may wish to build their own local cloud architecture. In doing so, there are many considerations to be made.
Figure 1.1: Local, Public, and Hybrid Clouds.

One consideration is who will be allocated to maintain the local cloud. This is one area where hosting on public clouds has an advantage. Organizations hosting their own cloud will need to maintain proper security levels, software and hardware updates, network resources, and users. Maintenance of these resources is costly and time consuming. Depending on the size of the cloud, organizations may need to hire one or more cloud administrators. These administrators would be responsible for keeping the cloud resources available to the users, making suggestions on when and what to upgrade, security, and backup solutions.

Another consideration is the power consumption of the resources needed to build the local cloud. Since cloud architectures generally increase in size over time, a power aware load consolidating scheme should be applied to reduce unnecessary costs. Organizations increasing the ability of their local cloud should consider the increase in power consumption as the cloud grows. In the default computational resource load balancing algorithms for open source cloud software, power consumption is not a consideration. These default schemes only consider resource availability. By
implementing a load balancing scheme that is power aware, there will be trade-offs. The most profound trade-off is resource availability. This dissertation proposes a cloud computing resource consolidation technique instead of a load balancing technique. Resource Consolidation techniques shown in chapters 3 and 4 take the mentioned trade-offs into consideration. The goal is to reduce power consumption without affecting cloud resource availability.

Since local cloud architectures must provide persistent storage, a power aware persistent storage consolidating scheme can be applied to reduce the cost of maintaining inaccessible data. The resources given to users in the proposed local cloud architecture are in the form of virtual machines. These virtual machines are stateless, meaning a persistent storage option is needed. The only persistent data that needs to be accessible is data belonging to the users that have active virtual machines. All of the other stored data can be moved to a cold file server and switched off to conserve power consumption.

The architecture proposed in this dissertation is limited by scalability factors, such as network bandwidth, computational, and storage costs. Since this is the case, the architecture is well suited for organizations of small to medium size. This design takes into consideration cost of networking resources. In some cases, organizations may wish to scale their cloud across multiple distributed data centers. Consideration for this ability is given in chapter 6.

1.3 Local Data Center Design

A major consideration of hosting a local cloud is the design and layout of the space used to hold the cloud resources. Design considerations can be taken from larger energy efficient data centers [Pedram, 2012], [Bottari and Leite, 2011], [Ruch, Brunschwiler, Paredes, Meijer, and Michel, 2013]. As a small to medium sized organization, power consumption efficiency should be as high as possible. It is shown that only around 52% of the power consumed by a small data center is
actually used by the computation resources (Processing, Networking, Storage, Server Power Supplies) [Buyya, Beloglazov, and Abawajy, 2010]. The other 48% of the power consumption is used to service those resources.

Design considerations of the small local data center should take on a tiered approach, beginning at the lowest level, the compute and storage servers. Both Intel and AMD are driven by the same demand: increase computational efficiency while at the same time decreasing power consumption.

1.4 Outline

The remainder of this dissertation is organized as follows. Chapter 2 introduces relevant literature related to this study and proposes the research challenges. In chapter 3, the power aware computational resource load consolidation algorithm is introduced and evaluated. This chapter focuses on reducing the power consumption in local cloud architectures by switching off unused computational resources. Chapter 4 evaluates the effects of virtual machine live migration. In this chapter, performance of the cloud platform, performance of the virtual machine being migrated, and the power consumed during a live migration is evaluated. Chapter 5 presents a power aware persistent storage consolidation algorithm for local cloud architectures. User’s data will be stored on low power storage nodes while they are inactive. When the user becomes active, their persistent data is moved to an accessible, powered on storage node. In chapter 6, a solution is presented for implementing a local cloud architecture using minimal networking resources. While many small to medium sized organizations would contain their local cloud in a single data center, a distributed architecture is also presented in this chapter. Chapter 7 presents the future work for implementing a power aware local cloud architecture. This includes a more efficient means of transferring data between auxiliary cloud clusters, accessing GPU resources within a virtualized environment, and
accessing persistent data without utilizing cloud computational resources. Chapter 8 gives the conclusions of chapters 3 through 6.
Chapter 2

BACKGROUND AND LITERATURE REVIEW

The concept of delivering computing resources through a global network was conceived in the 1960s. Looking back and understanding the problems that computing and networking had in general will give us reference points as to how cloud architecture(s) began, and what it needs to overcome before it is adopted to replace current computer/network architectures.

In the past computers were clustered together to form a single larger computer. This technique allowed industries to have large computer systems that used proprietary protocols to communicate among themselves and local client devices. The problem was developing a networking protocol that allowed these different industries to communicate with each other. On April 23, 1963 J.C.R Licklider [Licklider, 1963] addressed a group of people affiliated with the Intergalactic Computer Network. His ideas were to have individual computers interconnected and program access from anywhere. The problems mentioned in his memo came down to a common networking language used by all machine languages and compilers. This group eventually led to the development of ARPANET in 1969. Licklider’s ideas sound similar to the current definition of cloud computing.

Grid computing, a computer networking concept that provides seamless access to computing power and data storage capacity distributed over the globe, became a reality in the late 1990s due to the previous work of Ian Foster and Carl Kesselman [Foster, Kesselman, Nick, and Tuecke, Jun], [Foster, Kesselman, and Tuecke, 2001]. Since devices that make up the grid architecture
may be thousands of miles apart, latency can be a problem for fetching and execution of requests from customers. Running a data intensive application with geographically distributed data sources can create a restriction in I/O performance, causing the devices to run inefficiently. This can affect the economic viability of the service, and is still a problem with distributed datacenters in cloud architectures today.

Cloud computing was developing at a slow rate, with Web 2.0 being the current iteration. Bandwidth has been and still is a determining factor for cloud adoption. Finally, in the late 1990s, Internet service providers were able to increase their bandwidth to a point that cloud architectures could be a viable alternative to traditional computer network architectures. Since the late 1990s with the adoption of the two main broadband services, cable and DSL, the bandwidth available to customers has been on a steady rise. Some cloud architectures are heavily dependent on distributed data, given that the customers and their client devices are often mobile. Wireless data plans for mobile devices have also been increasing their bandwidth.

Some of the leaders in cloud computing began in the early 2000s. Amazon Web Services (AWS) began in 2002, which provided a suite of cloud-based services including storage and computation. Amazon then launched its Elastic Compute cloud (EC2) [Amazon, 2011b] in 2006. This combined with Amazon’s Simple Storage Service (S3) allowed customers to rent computer hardware so that they can run their own applications and store data in the cloud. Other cloud leaders include Microsoft, Google, and Apple. Eucalyptus [Eucalyptus, 2011b] provides private cloud solutions that are compatible with AWS, giving users an open source option for creating local clouds. Each of these vendors is developing cloud technologies with different goals in mind. Each has different company backgrounds and different target markets they wish to saturate.
2.1 Cloud Architecture

Many vendors have accepted and implemented the typical cloud-layering scheme, and this scheme is also typically applied in a local cloud environment. This architecture is similar in conception to the OSI (Open Systems Interconnection) [ITU-T, 1994] model in that each layer can be manipulated individually without having to deploy changes (minimal interconnection changes) to other layers of the cloud. Figure 2.1 shows the cloud-layering scheme used by most vendors.

![Diagram of Cloud Architectural Layers](image)

Figure 2.1: Cloud Architectural Layers.

2.1.1 Client Layer

Most customers will access the cloud through interaction with client devices. The cloud vendor must be aware of the different types of client devices to make sure their service will satisfy the customer’s needs. There are many different types of client devices that customers may use, however, most of these devices will fall into one of these three categories: thick clients, thin clients, or mobile clients.
2.1.1.1 Thick Clients

Thick clients, or heavy clients, are typical desktop workstations that have the ability to process resources locally as well as interact with networked resources. These clients are functional even if they do not have a network connection, however, it is only considered to be a client while connected to a server. While connected to a network, the thick client can access multiple server types for information and applications that are not stored on its local hard drive. Thick clients are typically powerful enough to execute multiple client-side (local) applications while still connecting to cloud resources. Another advantage that thick clients have over other clients is their ability to process high quality multimedia, such as Blu-Ray playback and video gaming. The versatility of this type of client makes it a common choice for cloud customers.

Thick clients do have disadvantages when being utilized as the primary interface to the cloud. One disadvantage is that thick clients have to be updated very often. Since these devices use local operating systems, applications, and security features rather than relying on a centralized server to provide these services, each client has to be maintained individually. There are administrative software suites that provide “touchless” updating over a network, but this only works if the client is on and the network is operating correctly. Thick clients also have the disadvantage of being the most energy consuming choice of all clients. Since these devices have many components that give the user the ability to work as a standalone node, the power requirements are greater than thin clients and small battery operated mobile clients. Hardware devices that fall into the thick client category include desktop PCs, workstations, laptops, and netbooks.
2.1.1.2 Thin Clients

Thin clients are computing devices that function as a content absorption device on a network. Unlike thick clients, thin clients are unusable when they are unable to connect to the network assets. These solid-state devices connect over a network to a machine that serves as a processing and storage device. Thin clients contain a minimal set of hardware that provides the user with input devices such as a mouse and keyboard, output for the monitor(s), audio, and a network connection.

Since thin clients are totally dependent on back-end servers to host the instance operating system, applications, and storage, thin clients have longer life spans than traditional thick clients. Thin clients simply give the user the ability to access cloud applications without bringing processing concerns to their local device. These devices are generally cheaper to buy, maintain, and replace than their thick client counterparts.

Another advantage of thin clients is they consume significantly less energy than thick clients. Since most thin clients are solid state, little or no active cooling solutions are needed, thus further decreasing power consumption.

Thin clients do have disadvantages. For one, since powerful servers are used to provide all of the data and applications to the clients, the local node has low computational abilities. Also, if a user needs a specific hardware interface, a thin client would not be the best choice since their interface is limited.

System administrators have the advantage in a thin client setup in that all of the hosting, application, storage capabilities are centrally located on servers. The disadvantage is this creates a single point of failure. If the server is interrupted, all thin clients connected to that server will be
affected. There are software solutions for seamless hardware faults, but this software is expensive and requires more hardware than some customers can afford.

Thin clients are likely only if an organization has an in-house, local cloud. If a client only needs to access cloud-based services or is accessing a virtualized server, then thin clients are a great option over thick clients. There is also a high level of security because no data is stored on the thin client. All of the data resides in the datacenter or in the cloud, so the risk of a physical breach is small.

2.1.1.3 Mobile Clients

These devices range from laptops to PDAs and smartphones. Laptops generally have the same hardware configuration as a desktop PC, and thus, will be considered a thick client. This section will focus on the field of smartphones, which are rapidly becoming more powerful and robust.

Smartphones are being developed specifically with cloud computing in mind. The client device has its own operating system, but is (in most cases) always connected to a network. These devices are optimized for portability, operating on batteries for hours at a time. Applications are installed over the air and charged to the user’s account. Customers can also configure their devices depending on their needs. Smartphones have the ability to give the user data based on their location using a GPS transceiver. With a combination of GPS, Cellular data, Bluetooth, and Wi-Fi, these devices give the customer complete mobility while staying connected to the cloud services they need.

As with thick and thin clients, mobile smartphones also have disadvantages. One disadvantage is the size of the device. The user interface is completely encapsulated in a device that fits in
the customer’s pocket. The small interface combined with limited processing abilities and battery power may prevent the smartphone from being the customer’s primary access to the cloud.

2.1.2 Application Layer

The application layer allows the customer to access the software they desire through the use of a network connection. Software-as-a-Service or ‘SaaS’ [Chappell, 2008], [Weiss, 2007] is a software application delivery model by which an enterprise vendor develops a web-based software application, then hosts and operates that application over the Internet for use by its customers. As shown in Figure 2.2, the vendor hosts all of the programs and data in a central (or distributed) location and provides their customers with access to the data and software through a web interface. The vendor is responsible for updating the software given to the user. They are also responsible for the security and actability issues involved in deploying a SaaS infrastructure.

Figure 2.2: SaaS Application Being Served to a Client Device.

2.1.2.1 Advantages of SaaS

There are definite advantages of using SaaS as opposed to a traditional software package and license. Generally, in the existing software deployment model, customers purchase a software package and a number of licenses to cover their need. The software then becomes the property
of the customer and the vendor as detailed in the license agreement. The vendor in this case would provide support and updates for the software. SaaS, on the other hand is not licensed to be purchased by the customer. Instead of buying the software and installing it on every in-house device, the customer pays for use of the software through a subscription. Since this is the case, the customer only pays for the software they need, when they need it. The customer also has the advantage of not maintaining these types of applications. Vendors will host SaaS applications and provide updates and maintenance. This eliminates the need for the vendor to provide support for multiple hardware and software configurations. It also decreases the likelihood of unlicensed use allowing the vendor to eliminate some of the costs associated with combating piracy. Vendors of SaaS applications are responsible for updating their software (and hardware). The vendor accomplishes this since they are maintaining the software in the cloud instead of on the customer’s local machines. The customer will always have the latest version of SaaS applications whenever they log in because there is only ever one version available. Customers will never have to worry about their software becoming obsolete and unsupported by the vendor. Since the vendor is hosting the software in their datacenters, they are able to quickly make changes if a customer finds an error in the software. Once the error is corrected in the datacenter, all customers that are subscribed to the service will see the changes.

2.1.2.2 SaaS Limitations and Disadvantages

There are disadvantages with SaaS applications as they are compared with traditional in-house software distributions. Although SaaS vendors can easily work in most enterprise settings, there are certain requirements that could make it undesirable for some customers.

The network connection a customer has to a cloud vendor is the most important aspect of the cloud computing architecture. In most cases, it is the customer’s responsibility to establish a
connection to the web so that they are able to use the cloud vendor’s applications. To use a SaaS application, the customer must have a sustainable, high bandwidth network connection. Although this is typically not a problem in a customer’s wired infrastructure, mobile clients are not guaranteed to have a sufficient connection to access SaaS applications. In most cases, SaaS applications are so network intensive, mobile devices are almost useless unless they are in a 3G/4G coverage area. Since it cannot be guaranteed the customer is in a high-speed coverage area, SaaS applications are not always available. If the mobile client is actively using SaaS and moves to a low/no coverage area, the SaaS application could be effectively terminated. Vendors are working on a solution for this situation. If a steady data stream is not always needed, a mobile (or any other) device can store local data in the cache and synchronize when the network connection becomes available again.

SaaS vendors are facing the challenge of load balancing. Technical obstacles include a way to produce an effective, multitenant architecture. Clouds are inherently multitenant, since they allow multiple customers to access single SaaS applications or resources. This is different than single tenant architecture where customers are given individual instances of applications. Advances in virtualization technologies have offset the problem of efficient load balancing, but designing SaaS applications that are effectively delivered to thousands or hundreds of thousands of customers is a difficult task to achieve.

Lastly, one of the most discussed issues in cloud computing is the fact that the cloud vendor completely hosts and maintains the customers data. Vendors are in control of all of their customers’ information. Customers have to be fully aware of the security policies of cloud vendors since they will trust them with all of their data. SaaS vendors typically have very meticulous security audits, but customers are, or should be, aware that their data could be stored on the same physical devices
as their competitors if they both use a cloud architecture solution. Trust will always be an issue between SaaS vendors and customers as long as sensitive data is stored in the cloud [Li and Ping, 2009].

2.1.3 Platform Layer

Platform-as-a-Service, or 'PaaS', is an application delivery model that supplies all of the resources required to build applications without the cost of buying and maintaining the hardware and software needed to host the application [SunMicrosystems, 2010]. PaaS gives the ability to encapsulate a layer of software and provides that as a service to the customer that can be used to build higher-level applications. A vendor that offers PaaS can create a platform by integrating an operating system, middleware, application software, and a development environment. The customer would have access to the encapsulated service by using an API provided by the PaaS vendor. The customer can interact with the platform through the API, and the platform performs actions to maintain and scale itself according to the customer’s needs. Virtual appliances are examples of a PaaS [VMWare, 2011]. The appliance would have its underlying software hidden from the user, and only an API or GUI would be available for configuring and deploying the resources that the appliance provides.

PaaS solutions are development platforms for which the development tool itself is hosted in the cloud and accessed through a browser. Using PaaS, customers can build applications without installing any tools on their local machines and then deploy those applications easily. Using this model instead of traditional developmental platforms, IT departments can focus on competitive innovation instead of maintaining a complex infrastructure. Organizations can redirect resources from maintenance to creating applications that return a profit. Customers of PaaS vendors rely on the fact that the vendors offer virtually limitless processing power that gives them the ability to
build powerful applications in a networked environment and deploy these applications to others whenever they are needed.

Examples of PaaS include the Google Apps Engine [Google, 2011a] and Salesforce’s Force.com [SalesForce, 2011a]. Google’s App Engine allows customers to build and run web applications on Google’s infrastructure. The apps have the ability to scale, as the customer’s needs change. The customer has the ability to develop applications in either the Java or Python environments. Force.com brings a similar approach by allowing customers to develop applications faster and at a lower cost than they could by using traditional software platforms. Force.com gives the ability to create custom databases, user interfaces, web sites, and mobile-targeted applications.

2.1.4 Infrastructure Layer

Infrastructure-as-a-Service, or ‘IaaS’, is the third form of service from a cloud vendor. Where SaaS and PaaS provide applications to customers, IaaS does not. IaaS services offer the hardware, in the form of virtual machines, so that the customer’s organization can do with it as they need [SunMicrosystems, 2010]. IaaS is also sometimes referred to as hardware-as-a-Service, or ‘HaaS’. The cloud vendor maintains all of these resources in their datacenter, and bills the customer for the amount of resources they consume. An organization may analyze the cost benefits of an IaaS vendor and compare that to their current hardware infrastructure. Rather than purchasing servers, software, racks, and paying for the maintenance, the customer will rent these resources from the cloud IaaS vendor. Depending on the IaaS vendor, customers can rent resources, such as server space, network equipment, memory, CPU cycles, and storage space.

The infrastructure can be scaled up or down dynamically depending on the customer’s needs. Also, as an advantage to the cloud vendor, multiple tenants can use the same equipment at the same time by using virtualization software. This software emulates many physical machines to
the customer while executing on fewer server devices, which lowers costs by eliminating machine idling. Since the cloud IaaS vendor maintains all of the physical devices, the customer’s organization can focus more on creating and maintaining a competitive edge rather than funding an IT support staff. Amazon’s Elastic Compute Cloud (EC2) is an example of IaaS provided by a cloud vendor [Amazon, 2011b].

2.1.5 Hardware Layer

The hardware layer of the cloud computing architecture combines all of the physical resources that a cloud vendor owns to provide services to their customers. This includes servers, storage devices, networking equipment, racks, and software.

Most cloud vendors use a distributed datacenter approach to providing services to its customers. Each of these datacenters combines to operate seamlessly to the customer. Each datacenter will accommodate racks filled with servers and storage devices that have the ability to perform very efficiently in a virtualized, networked environment.

2.2 Cloud Access

There are many different ways to access cloud resources. Vendors give customers different types of platforms and development tools to create cloud applications and customers have several options that they can use to access those applications. Local clouds can be accessed using the same tools needed to access a public cloud. Web browsers, frameworks, and API’s exist, and open source software is available to help users connect to local clouds.

2.2.1 Cloud Platforms

A platform delivers the cloud-computing environment to the customer. Many frameworks provide software libraries for common applications, such as web site templates, session manage-
ment, database access, and code reuse. The vendor makes these tools available for the convenience of the customer.

2.2.2 Web Application Frameworks

A web application framework is a generic toolkit for the development of dynamic web applications. These applications manage system processes and data while exposing finished content and services to customers using web browsers and other interactive clients. The advantage of using a web application framework is the overhead that comes with common activities in web development is reduced. These frameworks provide software libraries that are already written to do specific tasks so the developer does not have to write these common tasks every time a web site is developed [Demirkan, Harmon, and Goul, 2011], [Kandaswamy, Fang, Huang, Shirasuna, Marru, and Gannon, 2006].

The early developers wrote hand-coded HTML that was published on web servers. If some aspect of a website needed changing, it had to be done by the IT administrator. Later, dynamic web sites began to appear with the addition of the Common Gateway Interface (CGI) [D. Robinson, 2004], which had the ability to interface external applications with web servers. Eventually, packaged frameworks appeared that combined many software libraries useful for web development into a single software application for developers to use. Two of most prevalent web frameworks used by developers are Django and AJAX.

Django [Django, 2011] is an open-source web application framework written in Python that follows the model-view-controller architectural pattern. It was developed to ease the creation of database driven websites and implements reusability of components. Django utilizes the principle of DRY (Don’t Repeat Yourself) and provides an administrative CRUD (create, read, update, delete) interface that is dynamically generated. Django includes many of the core components a
developer would need to develop, test, and maintain web applications. It includes a lightweight, standalone web server for web site deployment. It also includes an internal dispatcher system that allows an application’s components to communicate using predefined signals and can translate its components into multiple other languages. Google App Engine applications can be developed in the Python language and includes Django.

Asynchronous JavaScript and XML (AJAX) [Murray, 2006] is a group of web development tools used for creating interactive web applications. Web applications that use AJAX can retrieve data from the server asynchronously. Since this is accomplished in the background, it will not interfere with the display and behavior of the current page. AJAX represents a wide range of technologies including XHTML and CSS, the Document Object Model (DOM) for dynamic display of data, XML and XSLT for interchange and manipulation of data, and JavaScript as the client side scripting language that combines these technologies to give the user a rich web experience. An advantage to using this web framework is that by using asynchronous requests, the web browser is more interactive and responds quickly to user inputs. Another advantage is that since the client primarily processes these scripts, they only need to be downloaded once, effectively decreasing the bandwidth to the cloud vendor. Disadvantages to AJAX include the inability to bookmark dynamically created web pages, the web browser requires support for AJAX, and there is no standards body behind AJAX. The disadvantages are relatively minor in respect to the capabilities that the AJAX web framework gives the customer, so it continues to be a forerunner in the web application framework platforms.

There are other full-featured web application frameworks, such as JavaEE [Oracle, 2012], Catalyst [Framework, 2010], Ruby on Rails [Hansson, 2011], and the Zend Framework [Zend,
Each of these platforms has a common goal of rapid and ease of development of web applications.

2.2.3 Cloud Hosting Services

Cloud hosting is a type of Internet hosting service that allows customers and vendors to make their cloud applications accessible using the World Wide Web. These cloud-hosting vendors provide space on servers as well as providing Internet connectivity. The customer will need a cloud hosting service to store data and applications. In terms of providing access to the cloud, the host would be considered the “cloud provider” or “cloud host”. Cloud hosting provides customers with powerful, scalable, and reliable hosting based on clustered load-balanced servers. The cloud host provides subscriptions for billing based on the needs of each customer. Customers have the ability to customize their cloud services and scale based on the growth of their organization. Amazon’s Elastic Compute Cloud and Google’s App Engine [Google, 2011a] are two widely used cloud-hosting services.

The Elastic Compute Cloud (EC2) is a service provided by Amazon that enables customers to have a scalable computing solution in the cloud. It provides the customer with complete control of their computing resources and lets them operate in Amazon’s computing environment. EC2 can create new virtual machines in just minutes, allowing the customer effective scalability, either up or down, as their computing requirements change. Amazon’s EC2 changes the economics of computing by allowing customers to pay only for the capacity they actually use. Amazon initially offered Sun Microsystems’s OpenSolaris [Oracle, 2011] and Solaris Express Community Edition. Later they added many Linux distributions, Microsoft Windows Server 2003 and Microsoft Windows Server 2008. As of 2011, EC2 has over 400 custom images to choose from when creating virtual machines.
Google App Engine provides a platform and hosting solutions for web applications in Google’s datacenters. AppEngine is different from Amazon’s EC2 in terms of services provided to the customer. AppEngine, other than being a cloud host, is a PaaS vendor, where EC2 is an IaaS vendor and cloud host. Currently, AppEngine supports applications developed in Python and Java. It is possible though, with extensions, to develop in other JVM languages such as Jruby [JRuby, 2011], Scala [de Lausanne, 2011], and Jython [Jython, 2011]. Google has implied that future versions of AppEngine will be written to be language independent. Google provides the user with free service, up to a predetermined limit. Any usage over this limit is charged to the customer.

2.2.4 Cloud Applications

Most of the customer’s choice for choosing a cloud provider depends on why they want to access the cloud, and what specific cloud vendors provide in terms of service. Customers are generally looking for a specific set of applications that cover their needs. Some cloud vendors may offer many premade applications ready for use with no development required by the customer. Google is an example of a cloud vendor that provides many applications geared toward productivity. Otherwise, if the customer needs the ability to create custom applications, they will subscribe to vendors that allow customization and creation of new cloud applications.

2.2.5 Cloud APIs

A cloud customer that decides to build their own application will rely on specific APIs in helping them accomplish their goals. There are many different APIs available [Apache, 2011], [Eucalyptus, 2011a], [Rackspace, 2011], [SalesForce, 2011b], [OpenNebula, 2011a] and the ones used by the customer will depend on their programming skills and which cloud vendor they chose. An API is similar to SaaS because software developers do not have to start from the beginning each time they write a program. Rather than building one program that accomplishes everything, the
application can assign specific duties to other applications that are more designed to complete the
task. An API is an interface that defines the way two devices communicate. With APIs, the calls
back and forth between devices are managed by web services. Customers can use the vendor’s
specific APIs by programming new or existing applications to generate the right XML message to
utilize other applications they may need. Usually, when a vendor releases their API, they do so as
a part of a larger software development kit (SDK) that includes the API, programming tools and
platform, and documentation. A customer’s software developer will use an API to run silently in
the background of a cloud application, and not need a direct interface with the user.

2.2.6 Web Browsers

A web browser is a client application used for retrieving information and resources stored
on networked servers located on the Internet. Resources are identified by a Uniform Resource
Identifier (URI) that links the user to the many different types of resources stored on network
servers.

There are many different options for the customer in terms of which web browser they may
use. Web browsers tend to be mostly the same, but with some subtle functional differences. There
are cases when a specific browser must be used, like using Microsoft Internet Explorer when con-
necting to a web version of Microsoft’s Exchange server to get full benefits, but for the most part
the customer should be able to use any browser they want to display cloud applications. The top
five web browsers in the market today are Mozilla Firefox, Microsoft Internet Explorer, Google
Chrome, Apple’s Safari, and Opera. Each of these browsers has advantages and disadvantages
when it comes to specific customer’s needs. For example, Chrome has optimized API’s that al-
low it to more efficiently handle requests from the Google Docs application. Microsoft’s Internet
Explorer would be the best choice when using Microsoft’s web software, such as Live and Bing.
2.3 Cloud Storage

Cloud storage involves storing data with a cloud service provider rather than on a local system. Accessing this data is accomplished the same way as accessing other cloud applications, using a network connection and a client device. There are many advantages to storing data in the cloud. One advantage is the fact that customers have access to their data from any location or client device that has an Internet connection. This is especially convenient for mobile customers who may not have access to workstations or their organization’s wired networking infrastructure.

2.3.1 Storage as a Service

Storage as a service provides persistent space on a cloud vendor’s datacenter to customers who lack the budget to pay for their own storage [Zeng, Zhao, Ou, and Song, 2009]. This cloud service is not new, but given the complexity of current backup, replication, and disaster recovery needs, this type of service has become very popular, especially among small and medium-sized businesses.

The biggest advantage to storage as a service is initial cost reductions for the customer’s organization. Storage is rented from the cloud vendor using a cost-per-gigabyte-stored or cost-per-data-transferred model. This reduces overhead if the customer does not have to budget for storage software and hardware infrastructure. They just pay for how much data they transfer to and store on the cloud vendor’s servers. The customer uses the vendor’s client software to specify the backup set and then transfers that data across a WAN.

There are many different cloud storage vendors. Some vendors are specific in their storage applications, such as email and digital picture storage, where other vendors allow storage of all types of data. The most basic form of cloud storage allows a customer to subscribe to the vendor’s
service and copy files to their servers that store the data for later access [Zeng et al., 2009]. Cloud storage systems utilize hundreds or thousands of data servers. These machines and storage devices require maintenance from the vendor, so it is necessary to store saved data on multiple machines for redundancy. Vendors must use a redundancy plan to assure clients that they could access their information at any given time. Most clients use cloud storage because of the convenience of not having to buy and maintain storage devices, and implement recovery and disaster plans. The following represent some of the largest specialized cloud storage providers. Google Docs allows users to upload documents, spreadsheets, and presentations to Google’s data servers. These files can then be edited using a web browser. Email providers like Gmail, Hotmail, and Yahoo store email messages on their own servers. Flickr and Picasa host millions of digital photos and allow customers to create online photo albums. YouTube hosts millions of user-uploaded video files. GoDaddy and SquareSpace store files and data for many client web sites. Facebook, FourSquare, and Myspace are social networking sites that allow members to post pictures and other data that is stored on the company’s servers.

2.3.2 Storage Security

To ensure the customer’s data is secure, a cloud vendor must use a combination of encryption techniques, authentication processes, and authorization practices. Since more and more organizations are moving their data to the cloud for storage, concerns have been raised on privacy and security practices of cloud storage vendors. One of the most used methods of increasing security is through data encryption [Jensen, Schwenk, Gruschka, and Iacono, 2009]. Encryption is the transformation of plaintext, or any easily readable file format, into cipher text. Encryption is accomplished using a set of mathematical functions, or algorithms, and an encryption “key”. The customer’s data will be stored on the cloud vendor’s servers encrypted with the ability to be read
only using an algorithmic key given to that specific customer. Even though other customer’s data will be stored on the same cloud vendor’s datacenter, each customer can only access the data that they have uploaded to their own account. There are two general types of encryption used for cloud storage security: hardware-based and software-based encryption. Hardware-based encryption is built into a piece of hardware. An example would be pre-encrypted hard drives that cloud vendors can purchase and use. All data stored on these drives is automatically encrypted. Software-based encryption refers to an encryption program installed on the server that encrypts data using software algorithms, such as AES or 3DES.

Security must adapt to the unique technical demands that cloud application present. It must address the new challenges that arise when infrastructure resides across the Internet. The customer cannot rely on their organization’s firewall to manage access to cloud applications because by definition these applications are accessed over the Internet outside of the corporate firewall. Authentication of a user connecting to a cloud resource works differently than authentication of a user connecting to a resource within their local organization. The customer’s organization can rely on multiple layers of authentication. For instance, if the organization is using Microsoft Windows running on a network managed by Active Directory, a user can be authenticated at a node by a user name and password. That user name and password then has deterministic access to resources within the organization’s network. This model does not scale to the cloud because users are not necessarily connected to a corporate LAN. Also, many users who are not affiliated with local organizations are not part of any company’s Active Directory governance. This is further complicated with global enterprises that are widely distrusted with users accessing resources over the Internet not using VPNs. Authentication for cloud computing resources has many challenges other than just allowing access of resources depending on a user name and password pair.
In authorization practices, the client lists the users who have access to different levels of information stored on the cloud vendor’s server. Many corporations have multiple levels of authorization. For example, an entry-level employee may have limited access to data stored on the cloud and the head of the IT department might have complete access to all data stored on the cloud vendor’s server.

2.3.3 Cloud Storage Reliability

Along with Security, a cloud vendor’s other top priority should be reliability. If a cloud storage system is unreliable, it becomes a liability. A customer will not pay a cloud vendor that hosts its services on unstable systems. The cloud vendors address the reliability concern mostly through redundancy. Data is written to the cloud from the customer, and propagated to many of the cloud vendor’s datacenters. If one of the datacenters fails, the customer should be oblivious to this fault and data should be accessible from one of the other redundant datacenters. There are always risks of losing data. Multiple datacenters can fail, datacenters can become compromised exposing the customer’s information, Internet services can fail, etc. The customer has to examine the advantages and disadvantages of using cloud vendors for data storage.

2.3.4 Advantages of Cloud Storage

There are many advantages to storing data in the cloud over local storage. With cloud storage, data resides on the web, located across multiple storage systems rather than at a designated corporation’s hosting site. Cloud vendors balance server loads and move data among various datacenters to ensure the data is close to the customer, making it quickly accessible and reliable. Storing data in the cloud allows customers to better protect their data in the case of a disaster. The customer’s organization may have a backup solution, but if the backups are stored locally and
events like a fire or hurricane destroy the organization, the backups will not be available. Having data stored off-site could be the difference between being down for a few days, or closing for good.

There are numerous common server hardware failures that can result in service outages: hard drives, motherboards, power supplies, network cards, etc. Each of these vulnerabilities has a point solution. IT administrators can buy redundant power supplies, but this approach is expensive and labor intensive. Cloud vendors give customers the advantage of providing hardware redundancy and automatic storage failover. Vendors prevent outages by allowing the customer to keep multiple copies of their data on different datacenters. The vendor knows how to regulate these copies in the event of any hardware failure.

Another problem of hosting storage solutions locally is planned outages. Employees of the company have no access to any of their data if the storage solution is taken offline. Reasons for planned outages are disk drives filling up and needing replacement, upgrading to faster processors, upgrading network capabilities, and patching operating systems and hardware drivers. Cloud vendors take care of these problems for the customer by providing outage-free upgrades. Since customers are not directly dependent on a single server from the cloud vendor, they can take a machine in the data center offline to upgrade it non-disruptively.

A local organization will usually buy more disk space than they need initially to accommodate for future data growth. This means they pay more for the storage long before they need it. If the extra storage is not needed for a long period of time, hardware storage costs tend to drop over time, wasting money for the customer. Cloud vendors solve this problem by giving the customer virtually limitless storage capacities. Customers can drastically reduce over-provisioning in a pay-as-you-go model while also saving on their utility costs in the process. This is attractive
to customers since cloud vendors can provide increases and decreases in storage as their needs change over time. The customer never pays for hardware they do not need.

Customers who store data locally also have the problem of capacity limits. No matter how much storage their IT department packs into a host machine, there is still only one host. Some hosts will allow for multiple network cards and multiple processors, but once all of the slots are full within that host, it has reached its capacity to scale with data growth. Cloud vendors solve this problem with the ability to scale processor power, network bandwidth, and storage capacity in parallel. The Cloud vendor simply adds more storage hosts to the cloud to scale along all the dimensions simultaneously.

Organizations will also have to actively maintain load distribution if they are hosting storage solutions locally. This tends to lead toward workload hot spots. Since a customer’s organization will have multiple clients and servers, unbalanced workloads are inevitable. Some servers will be under worked and others will be overworked. This can be managed with virtualization of servers, but maintaining this setup will be intensive. Cloud vendors allow their servers to balance primary and backup workloads among all of their data centers to achieve maximum performance and storage density, alleviating hot spots.

The last problem with local storage is the management and administration of hosts. Traditional host storage requires the storage of each node to be managed individually. Volumes or partitions are managed on a per host basis. Cloud vendors help the customer’s IT administrator by facilitating exports from a single administrative interface and providing a unified view of storage utilization for all of the company’s needs.
2.3.5 Disadvantages of Cloud Storage

Cloud storage is still in its infancy. Since this is the case, local organizations should not commit all of their data to a cloud storage vendor. Organizations may have difficulty with vendors because they are forced to rewrite solutions for their applications to insure compatibility with storing data online.

Some of the main concerns with choosing to use a cloud storage provider are price and reliability. Customers should learn how the vendor operates in these aspects before choosing them as their provider. The customer should weigh the costs of hosting and maintaining their data locally verses using a cloud solution. Also, customers should be aware that if they do not have a local copy of their data, and the cloud vendor makes a mistake, their data could be potentially lost. The customer should know exactly what their cloud vendor will or will not do in case of data loss or compromise.

Customers should be aware of the inherent danger of vendor outages when storing their data in the cloud. Amazon’s S3, for example, experienced a massive outage in February of 2008 [Amazon, 2008]. Many of their customers lost access to their data during this time. Amazon reported that they have responded to the problem, adding capacity to the authentication system that seemed to be the cause of the outage. They reported that no data was lost since they store data in multiple data centers, but the fact remains that the customers were not able to access their data as they had intended.

Theft is another great concern when storing data in the cloud. The customer should keep in mind that their data could be stolen or viewed by others who are not authorized to see it. Whenever an organization moves data out of its own data center, the security risk always increases since they
do not have full control over the security measures keeping their data safe. Customers should make sure that the vendor takes excellent measures for securing data once it has been transferred to their servers. Also, the customer should ensure that their data is secure while being transferred from their location to the cloud vendor. Using encryption technologies such as SSL can do this.

2.3.6 Amazon S3

S3 is Amazon’s Simple Storage System. This service seemingly gives the customer an infinite cloud storage solution for objects of variable size (1 Byte to 5 GB) [DeCandia, Hastorun, Jampani, Kakulapati, Lakshman, Pilchin, Sivasubramanian, Vosshall, and Vogels, 2007], [Brantner, Florescu, Graf, Kossmann, and Kraska, 2008]. Customers can read and update S3 objects remotely by using a SOAP or REST-based interface. Each object is associated with a bucket, so when the customer creates a new object, they specify into which bucket the new object should be placed. The customer can then query the items in a bucket for data, or the bucket could be used as a means of security. Customers can grant read and write authorization to other users for access to data in specific buckets.

2.3.6.1 S3 Buckets

Buckets are storage elements for objects in S3. Every object that a customer moves to S3 is stored in a bucket. For example, if the object named pictures/gorilla.gif is stored in the jeffreygalloway bucket, then it is addressable using the URL http://jeffreygalloway.s3.amazonaws.com/pictures/gorilla.gif. Buckets have many purposes: they organize the S3 namespace at the highest level, they identify the account responsible for storage and data transfer charges, they play a role in access control, and they serve as the unit of aggregation for usage reporting.

A customer can store objects in buckets that are in a specific region, namely regions that
are geographically close to their location. S3 supports the following regions: US Standard, US Northern California, EU Ireland, and APAC Singapore.

2.3.6.2 S3 Data Protection

S3 provides a robust storage architecture designed for mission-critical and primary data storage. S3 provides high-level durability and availability of objects over a given year. S3 also allows for versioning. Versioning is a way of keeping many instantiations of an object in the same bucket. In one bucket, as shown in Figure 2.4, the customer can have two objects with the same key, but with different version ID’s, such as photo.gif: Version 345678 and photo.gif: Version 456789. Using this feature, customers can easily recover from both unintended user actions and application failures. By default, requests retrieve the most recently written version of an object. The customer can retrieve alternate versions of an object by specifying a version of the object in a request.

![Figure 2.3: S3 Bucket with Versioning Enabled.](image)

Amazon’s S3 is not a free service. They charge customers USD 0.125 to store 1 GB of data per month. By comparison, a 2 TB disk drive from Seagate costs USD 110 today. Assuming a
three year life time of the disk drive, the cost is about USD 0.0015 per GB a month (not including power consumption). Customers should also be aware of the other costs of using S3. There is a cost of USD 0.01 per 10,000 get requests, USD 0.01 per 1,000 put requests, and USD 0.00 to USD 0.12 per GB of outbound network bandwidth consumed per month [Amazon, 2011a].

There are other costs and considerations in data storage that may persuade the customer to use S3 instead of hosting the storage solution themselves. Using S3 as a backup solution would eliminate any mirroring of data or disaster recovery plans for duplication of critical data for the customer’s organization. S3 replicates all data stored for customers to several of its datacenters. Each replica can be read and updated at any time and updates are propagated to replicas asynchronously. If a datacenter fails, the data can be read and updated using a replica at a different datacenter. This approach guarantees full read and write availability that is crucial for web based applications. No client device is ever blocked by system failures or other concurrent clients. This approach also guarantees persistence. The result of an update can only be undone by another update.

In terms of reliability, S3 is designed to provide 99.999999999% durability and 99.99% availability of objects over a given year [Amazon, 2011a]. S3 redundantly stores customer’s data on multiple devices across multiple facilities in an Amazon S3 Region. The service is designed to sustain concurrent device failures by quickly detecting and repairing any lost redundancy. When processing a request to store data, S3 will redundantly store objects across multiple facilities before returning a SUCCESS to the customer. S3 also regularly verifies the integrity of data using checksums.

2.3.7 Nirvanix Storage Delivery Network

Nirvanix [Nirvanix, 2011] uses custom software and file system technologies running on Intel storage servers at five locations, three in the Unites States, one in Germany, and one in Japan.
The Nirvanix Storage Delivery Network (SDN) is a cloud storage solution developed for enterprise use. The SDN stores, delivers and processes requests from users depending on their location to provide the most efficient service possible. SDN allows users access to services with a provided API.

SDN converts a standard server into an infinite capacity network attached storage (NAS) accessible by the user’s applications and integrates into an organization’s existing archive and backup processes. Nirvanix has built a global cluster of storage nodes referred to as SDN that are powered by the Nirvanix Internet Media File System (IMFS). With the ability to store multiple file copies in multiple geographic locations, SDN enables extensive data availability for its customers.

Nirvanix offers a product called CloudNAS for Linux that mounts SDN as a virtual drive that can be accessed by NFS, CIFS, or FTP. After the customer installs the application, IT administrators can apply standard file, directory, or access permissions, and users on the network can then access the Nirvanix-mapped drive from their existing clients. Nirvanix boasts a costs savings of up to 90 percent over managing traditional storage solutions. They also offer encryption services, built-in data disaster recovery and automated data replication to other geographically dispersed nodes, and immediate availability of data to the customer.

2.3.8 Google Bigtable Datastore

BigTable is a database that has the capability of handling numerous users in an on-demand basis. This database is built using the Google File System (GFS) [Ghemawat, Gobioff, and Leung, 2003]. Google started working on this application in 2004 and went public with it in 2008. BigTable was developed with high speed, flexibility, and scalability in mind. A BigTable database can be petabytes in size and span thousands of distributed servers. It is available to customers as a part of the Google App Engine, Google’s cloud computing platform.
2.3.9 Apple Mobile Me

Customers with a MobileMe [Apple, 2010] email account have their data synchronized with all of their devices because the folders, messages, and status indicators look identical. New email messages are pushed instantly to the iPhone over the cellular network or Wi-Fi, eliminating the need to manually check email and wait for downloads. Push also keeps contacts and calendars up to date automatically so when changes are made on one device, all other devices that are connected to MobileMe will be updated as well. MobileMe provides other services, such as locating a lost iPhone or iPad, photo sharing, iDisk which acts as a cloud storage solution, Me.com which stores all of the user’s information in a single accessible account, and 20 GB of cloud storage for hosting all of the customer’s MobileMe data.

MobileMe is available in two versions, as a single user account, or a family pack account. The single user account costs the customer USD 99 per year. The family pack account costs the customer USD 149 per year. The family pack is a one year individual MobileMe subscription plus four family member accounts, each with its own email address, iDisk folder, and 5 GB of online storage space giving the account 40 GB of total online storage.

2.4 Local Clouds

The cloud computing model does not always mean the customer has to traverse the Internet to have access to content. A local cloud removes the service provider component, and allows an organization to manage all of the content themselves in their own datacenter.

Some customers will not make the move to the cloud all at once. Most will start as a hybrid model, only moving some applications to the cloud while keeping most of their data in-house. The best way to become familiar with cloud computing concepts, benefits, and disadvantages without
the outsourcing commitment is to bring the cloud as close as possible by building a local cloud [Toby Velte, 2009].

2.4.1 Eucalyptus

In their current and upcoming releases of Ubuntu, their goal is to have a complete infrastructure layer for the cloud that will let anyone deploy their own cloud, whether it is for their own usage (private cloud), or public consumption. Resource and request distribution architectures are in a time of transition and the benefits of cloud computing promise to replace existing models. Ubuntu Enterprise Cloud (UEC), in conjunction with Eucalyptus and Canonical, is creating a private cloud platform that intends to help companies navigate safely in this time of change.

The architecture of Eucalyptus [Eucalyptus, 2010a], [Eucalyptus, 2011c], [Eucalyptus, 2010b] which is the main component of UEC, has been designed as a modular set of five elements that can easily be scaled [Eucalyptus, 2011c]:

1. Cloud Controller (CLC)

2. Walrus Storage Controller (WS3)

3. Elastic Block Storage Controller (EBS)

4. Cluster Controller (CC)

5. Node Controller (NC)

2.4.1.1 Cloud Controller

The Cloud Controller provides the interface for users to interact with Eucalyptus. This interface uses a standard SOAP (Simple Object Access Protocol) based API modeled after the Amazon EC2 API. The CLC connects with the Cluster Controllers and makes the final decisions
for creating new machine instances. The CLCs hold all of the data connecting users to current instances, the group of available machines to be executed, and a view of the load of the entire system. The CLC allows users to connect to cloud resources using command line or graphical interfaces. The newest version of Eucalyptus (3.2) has a more robust administrative cloud interface allowing users more control over cloud resources.

2.4.1.2 **Walrus Storage Controller**

This application consists of a REST (Representational State Transfer) and SOAP API, which are based on and compatible with Amazon S3. The WSC is a storage location for the virtual machine images that could be instantiated on the UEC. It also has the capabilities of storing data either from a running instance in the local cloud, or from anywhere on the Web. Each time a user instantiates a virtual machine, the image chosen is moved from the walrus storage controller to the node controller that will host the virtual machine. In an effort to reduce network congestion, images that are regularly used are copied to active node controllers.

2.4.1.3 **Elastic Block Storage Controller**

The EBS runs on the same machine(s) as the CC and is configured automatically when the CC is installed. It allows for creation of persistent block devices that can be mounted on instantiated instances of machine images in order to gain access to virtual hard drives. The user can create a file system on top of EBS volumes, or use them in other ways that a block device could be used. EBS also gives UEC the ability to create snapshots of instances and virtual drives, which are stored on WS3. These snapshots can be used for instantiating new EBS volumes and also protect data from corruption and loss. The same snapshot can be used to instantiate as many volumes as the user wants.
2.4.1.4 Cluster Controller

The CC operates as the gateway between the Node Controller and the CLC. This being the case, the CC will require access to both the NC and CLC networks. It receives requests to instantiate machine images from the CLC and then determines which NC will run the machine instance. This decision is based upon status reports that the CC receives from each of the NCs. The CC is also in charge of managing any virtual networks that machine instances run in and routing traffic to and from them.

Each time a new NC is added to the local cloud cluster, it will register with its CC. The NC will submit its capabilities (number of CPU cores, RAM, Disk Space, etc.) to the CC. Once the NC has registered with the CC, the CC will notify the CLC of the new cloud resources and begin instantiating virtual machines on the new NC.

2.4.1.5 Node Controller

The NCs’ software is installed on the physical servers where the machine instances will hosted. The physical NC machine must have a CPU that will support virtualization. The NC software job is to interact with the operating system and hypervisor running on the node. NC’s primary job is to discover the environment where it will execute in terms of available resources (disk space, number of CPU cores, type of CPU, memory), as well as running VMs that could be started independently of the NC, CC, and CLC.

After the environment has been surveyed, the NC will become idle, waiting for any commands from the CC, such as starting and stopping machine instances. When requested to start a machine image, the NC will:

1. Authenticate a user’s request
2. Download the machine images from WS3

3. Create the requested virtual network interface

4. Start the new machine instance as a virtual machine accessible to the user.

2.4.2 Eucalyptus Implementation

Eucalyptus (Elastic Utility Computing Architecture Linking Your Programs To Useful Systems) is an open-source infrastructure designed to produce customer made clouds in their own organization. Eucalyptus enables the organization to provide IaaS and requires no proprietary hardware to accomplish. This is the leading provider in developing an open cloud standard. The Eucalyptus platform supports AWS, giving the organization the ability to leverage public cloud usage with their private cloud architecture.

Some things to consider when building a private cloud with Eucalyptus are performance of the in-house physical devices, overall network performance, and costs of scalability [Intel, 2012], [Baun and Kunze, Dec]. Eucalyptus is implementable on commodity hardware, but the stability and efficiency of the private cloud implementation will vary greatly depending on the computational performance of the hardware used. As networking is concerned, a local area network will generally perform more efficiently, in terms of throughput and stability, than a connection to a public cloud provider. Finally, an in-house implementation will almost always be limited in its scaling ability due to funding. This is why a hybrid architecture is recommended. If scaling is immediately needed, the customer can just purchase more resources from a public cloud vendor.

During the time taken to research the areas of local cloud computing architectures, the resources used were come to be known as “Fluffy”. “Fluffy” is an infrastructure as a service (IaaS)
cloud which uses the Eucalyptus cloud middleware. Figure 2.4 gives a high level overview of the concept of the “Fluffy” cloud architecture.

![Fluffy Cloud Architecture Diagram](image)

Figure 2.4: Local Cloud Architecture of Fluffy.

2.5 Power Consumption in Cloud Architecture

Recently, the scale and density of cloud computing datacenters has increased tremendously. With this comes an ever increasing power demand. Administrators and cloud designers have become more focused on cloud datacenter power management not only because of the economic standpoint, but also an environmental standpoint. There are many components in the cloud architecture that can be refined to reduce the amount of power consumed for operation. Each computing
node has a power consumption footprint that is required while it is in operation. Depending on the utilization of that node, the power consumption levels will change.

In [AbdelSalam, Maly, Mukkamala, Zubair, and Kaminsky, 2009] the authors propose a mathematical model for power management in a cloud computing environment that is generally used to serve web applications. Using their model, the number of physical servers, along with each server’s CPU frequency, that are needed to perform a set of jobs is determined. Using their mathematical models, only a subset of the total number of physical servers is used, thereby, decreasing the amount of power consumed by the cloud.

The authors of [Mochizuki and Kuribayashi, 2011] propose allocation guidelines in a case where there is a limit to the amount of power consumption available to the cloud resources. Using optimization guidelines, the authors allocate network bandwidth and processing ability based on the amount of power needed to finish job requests. The cloud resources in this architecture are geographically dispersed and are powered by high-density batteries. Cloud resources in one area are confined to the available power in that area, and cannot use power from other areas. Once a request is made for cloud resources, a single area is selected from k areas that have the computational and electrical resources available to handle the request. If none of the distributed cloud computing resources are able to handle the request, then it is rejected.

Physical servers can be placed in different pool sets determined by their current power consumption and the number of requests for resources in the IaaS cloud [Ghosh, Naik, and Trivedi, 2011]. Servers can be placed in hot, warm, or cold pools. Hot pool servers are powered on and running. These servers are immediately ready to host virtual machine requests. Hot servers have pre-instantiated virtual machines that need little or no provisioning delay. Warm pool servers are powered on and in an idle state. Warm servers have the underlying operating system and hypervisor
loaded, but no pre-instantiated virtual machines. This reduces the power consumption of the server as compared to hot servers, but incurs latency when a user requests resources as virtual machines become available. Cold pool servers default to a powered off state. This saves the most energy, but incurs the largest latency when cloud resources are needed. Users will have to wait for cold servers to boot the underlying operating system and hypervisor, network provisioning for the server to take place, and virtual machine instantiation before resources will become available. By varying the number of servers in each pool, a power savings and resource latency trade-off is evaluated.

A green virtual network architecture is given in [Chang and Wu, 2010]. The routing algorithms defined decrease the power consumption of data transmission in cloud environments. The cloud architecture is examined for energy consumption for transferring data among nodes. The green routing algorithm chooses the lowest cost path that stays within the constraints of the data transfer. Traffic load of the cloud is also taken into consideration when picking a path of data transfer.

A green scheduling algorithm which makes use of a neural network based predictor for energy savings in cloud computing architectures is given in [Duy, Sato, and Inoguchi, 2010]. This scheduling algorithm reduces power consumption by shutting down unused cloud servers. The scheduler first determines the dynamic workload on the cloud servers. Only the number of necessary servers is kept on to accommodate this workload. Initially, all servers are in a low power state, or hibernation. If a server is needed for a job request, the green scheduling algorithm will send a “restarting” signal to that server. After a given time for the power up sequence, the server is now in an “on” state ready to receive a user’s request. To reduce latency in the event of a large increase in demand, the authors power on enough servers to accommodate 20% more than the current workload in the cloud.
Virtual machine migration performed in [Corradi, Fanelli, and Foschini, 2011] for the sake of physical cloud compute node consolidation is shown to be a viable solution to increase the overall cloud power efficiency. The goal is to reduce the total number of physical cloud servers needed to handle current requests. The authors use the Xen [Citrix, 2011] hypervisor, which gives the ability to migrate virtual machines between different physical hosts. The authors mention two types of VM migration: regular migration and live migration. In regular migration, the VM has to be paused and copied to another physical server. During this process, it is not accessible. Using live migration, this action is almost transparent to the user. The migration takes place as the user is still active, and only pauses for a short time to complete the transaction.

The authors of [Wu and Wang, 2010] present a control framework of tree distribution for power management in cloud computing architectures. The authors address power management problems by presenting a tree distribution unit as the framework for power budgeting. Instead of having an even distribution of cloud components used to accommodate job requests, the tree structure organizes all of the servers in the cloud by type and priority.

A low power mode is introduced in [Harnik, Naor, and Segall, 2009]. The goal is to reduce the amount of power consumed by cloud storage components with minimal disruptiveness to the overall cloud. Their approach powers down unneeded storage nodes to conserve power consumption. All data items are still accessible, although during low power periods, users may incur latency when accessing data.

2.6 Interfacing and Deploying the Cost Effective Cloud Architecture

There has been much work on developing cloud interfaces. The OCCI (Open Cloud Computing Interface) [Edmonds, Johnston, Metsch, and Mazzaferro, 2010] community is working towards a standard for creating interfaces for several different types of clouds. The OCCI model
is essentially an API which provides developers a standardized set of tools and utilities for managing, deploying, monitoring, and scaling cloud resources. Many implementations of OCCI are in development and work is being done to integrate the OCCI model into popular cloud stacks such as Eucalyptus and OpenStack.

The authors of [Tanahashi, Chen, Marchesin, and Ma, 2010] present a design for online visualization services using cloud computing architectures. This interface is used for visualization of large data sets targeted for use by casual users with a simplistic interface. Rendering of visualizations is done entirely on the cloud, which enables users to view content from a large array of devices such as laptops, cell phones, or other low power devices. Additionally, the interface is web-based and does not require any special browser plugins for visualization, which makes it highly usable on devices which have limited capabilities.

In [Elmroth and Larsson, 2009], the authors present a vendor neutral interface for controlling placement, migration, and monitoring of virtual machines in federated cloud environments. This interface can be used as a layer of abstraction that simplifies management of virtualized service components between competing cloud vendors. There is additional work being done in collaboration with the OCCI community to contribute extensions which will address these concerns and provide a standardization which may be adopted by cloud vendors.

Ubuntu Enterprise Cloud [Simon Wardley, 2009] has a standard interface for administering Eucalyptus cloud resources. This interface is adequate for cloud administrators keeping track of resources and user accounts, and users needing to register for the first time and download security credentials. This interface, however, is not well suited for users wanting to launch virtual machine instances, or access or terminate these instances.

Hybridfox [License, 2012] is a Mozilla Firefox browser add-on that allows users to manage
Amazon EC2 and Eucalyptus cloud computing environments. This is a GUI web interface that allows administrators to manage images, instances, elastic Ips, key-pairs, and persistent storage. It also allows for users to configure the type of instance they want, start and stop instances, and connect to these instances with a mouse click. This is a great interface for users and administrators of EC2 and Eucalyptus, but our design requires an interface that accommodates more complex networking tasks.

The first version of the NIST cloud computing reference architecture is described in [Liu, Tong, Mao, Bohn, Messina, Badger, and Leaf, 2011]. This reference is a vendor neutral model which concentrates on the functionality and interactions of devices and users in the cloud computing paradigm. This reference model is needed since there is no current standard for defining any cloud computing architecture.

The deployment of private cloud architectures can be a long and daunting task. The authors of [Head, Sailer, Shaikh, and Shea, 2010] discuss the many issues of deploying private cloud architectures. The cloud is composed of many inter-dependent components, which makes troubleshooting of errors in the deployment process difficult and time consuming. Users who might not be able to effectively interpret the possible errors may spend a large amount of time to determine the cause and solution. However, deployment of these architectures can be easier to manage with the use of tools, which capture the state of the deployment along with various data in order to pinpoint and troubleshoot possible errors. In [Tianfield, 2011] the author discusses a basic taxonomy of cloud computing architectures. Clouds are divided into two categories, cloud application architectures, and cloud platform architectures. Cloud application architectures are described as providing three services: virtual services (virtual machines, virtual appliances), cloud brokers that assist users in their needs, and business services. Cloud platform architectures determine the type
of resource given to the user, IaaS, PaaS, SaaS. These architectures combine the physical resources and present them virtually to multiple users.

A cloud computing open architecture is presented in [Zhang and Zhou, 2009]. Since there is no standard for cloud computing, research involving resource sharing based on business requirements is performed. The cloud computing open architecture aims to create a reusable way of creating scalable and configurable provisioning platform for cloud computing. It also proposes a set of common and shared services for building cloud platforms and attempts to maximize the potential business aspect of providing these cloud services to users.

An Infrastructure-as-a-Service cloud architecture is described in [Montero, Moreno-Vozmediano, and Llorente, 2011]. They present the cloud operating system (management of the physical and virtual infrastructures) as the key component to this architecture. The cloud operating system is divided into three sections: drivers, core components, and high-level tools. Drivers are used to interface with hypervisors, storage devices, and different network types. The core components are the resources that users request when connecting to the cloud. Users connect using high-level tools used to ease the request of cloud resources. Inter-cloud services help to evolve cloud architectures into a more complex and rich system, which is ultimately conducive to the goals of the IaaS cloud. It is not uncommon for a virtual private network to be used in establishing a connection between a user and cloud services. However, VPN connections are currently not utilized to extend the secure nature of VPN to inter-cloud architectures. The authors of [Dayananda and Kumar, 2012] propose the use of Ipsec VPN to establish secure communication between distributed inter-cloud architectures.

Content distribution networks are gaining a lot of momentum in the cloud. However, there is a lot of speculation as to whether quality of service (QOS), or in this case quality of experience
(QOE) may be guaranteed in a cloud architecture. The authors of [Tran, Mellouk, and Hoceini, 2011] propose a distributed network of replicated servers to ensure quality of experience and quality of service of cloud-based content delivery networks. Several replicated servers are deployed in various geographic locations such that distributed CDN services are sufficiently redundant to provide quality of service and user experience.

A power-aware architecture cloud architecture is presented in [Tu, Kuo, Teng, Wang, and Shiau, 2010] that demonstrates the energy consumption of diskless versus diskfull architectures. The design also uses cpufreqd and xenpm for frequency scaling of the cpu. A smart power monitoring approach is also applied to determine energy consumption and power loss to the architecture.

2.7 Current Research in Power Management for Persistent Storage in Cloud Architectures

There has been a considerable amount of research in power management for cloud data centers. We focus our related work on power management of persistent storage devices for small to medium sized organizations. Our design does not consider the micromanagement of the internal power consumption of each storage node. The power aware consolidation scheme that manages the persistent storage nodes will reside on the cluster controller, which is an administrative node used to handle networking and load balancing requests for compute nodes in Eucalyptus.

The authors of [Narayanan, Donnelly, and Rowstron, 2008] propose a technique called write off-loading. In their design, when persistent data is write-dominated, data blocks are redirected to other storage devices in the data center. Original data locations have their disks spun down, reducing the amount of energy consumed for hosting persistent data. This causes data volumes to be idle for 79% of the time on average. The drawback in this scheme is that if a user needed to read data from a non-off-loaded block, they will incur a latency as the hard disk spins...
up. Spinning down disks and implementing write off-loading decreases energy consumption by 45 to 60% [Narayanan et al., 2008].

Persistent storage cache management schemes are evaluated for energy consumption in [Zhu, David, Devaraj, Li, Zhou, and Cao, 2004]. The authors evaluate cache management for both read accesses and write accesses to disks. OPB, a simple power-aware off-line greedy algorithm is presented, along with a power-aware version of the LRU algorithm. Power consumption is decreased, while in some cases availability is increased. They present limitations in their work that some solutions are only designed for multiple disks instead of individual disks since their study focused on high-end storage systems.

A file system solution to reducing power consumption is given in [Ganesh, Weatherspoon, Balakrishnan, and Birman, 2007]. Authors use log-structured file systems (LFS), and its associated log to precisely determine which disk to access for user stored data. This gives a perfect prediction for data write accesses while also being application independent. They propose that by using LFS, along with a good caching scheme, reads to storage devices will be minimized, and only a small number of the disks have to be powered up to provide full coverage for users read and write requests. They also describe the trade-off of power savings versus introducing latency as the percentage of powered on disks changes.

Investigation of how high scale distributed storage systems can reduce their power consumption during low-utilization time intervals by operating in a low-power mode is examined in [Baliga, Ayre, Hinton, and Tucker, 2011]. They examine methods that do not modify the current placement functions of the persistent storage architecture, as well as a method that obtains full coverage for any placement function using auxiliary nodes.

The authors of [Baliga et al., 2011] present a comprehensive energy consumption analysis
of public and private cloud computing architectures. They show that with low usage, cloud storage systems provide a more efficient persistent data solution than storing data on each individual’s local machine. But as the number of file accesses increase, the power savings becomes minimal and comparable to storing data on local hard drives.

The authors of [Hansen and Jul, 2010] present a design for a distributed storage system called Lithium. Lithium provides a highly scalable, distributed storage architecture based on Byzantine Fault Tolerance systems. Lithium supports commodity hardware and takes into account factors of consolidation to counter the performance limitations with highly distributed data storage systems.

The authors of [Wang and Varman, 2011] present an approach for multiplexing multiple bursty workloads on the same server. This approach takes into account storage consolidation measures, which present challenges in resource management and capacity provisioning. The solution proposed in [Wang and Varman, 2011] multiplexes the concurrent workload bursts on a single server to alleviate the challenges in highly distributed storage systems without decreasing performance.

The authors of [Zhu, Zhu, and Agrawal, 2010] have developed a power-aware resource consolidation algorithm, which uses the dimensionality reduction method to relate resource requirements with power consumption. The algorithm utilizes a distance metric, which is based on interference between resource requirements. The authors show that their algorithm improves power consumption when used in highly data distributed systems for scientific computing purposes.

The authors of [Shen, Subbiah, Gu, and Wilkes, 2011] present a method for resource scaling in the cloud called CloudScale. The CloudScale architecture attempts to predict resource
demand and determine the impact of data and VM migration. In this respect, CloudScale is a predictive architecture in contrast to storage consolidation, which is more of a reactive measure.

The authors of [Verma, Koller, Useche, and Rangaswami, 2010] demonstrate a new power-aware storage consolidation system, which uses Sample-Replicate-Consolidate Mapping (SRCMap). The authors show that SRCMap can seamlessly consolidate data between nodes, after which unused nodes may be powered down to save energy. This process ensures that data is always highly available, while minimizing the power consumption of the system.

2.7.1 Virtual Machine Migration

The ability to transfer virtual machine instances between different physical servers is a powerful tool when applied to managing cloud computing resources. Users of cloud resources interact through virtual machines, which can be clearly abstracted from the physical devices in the cloud. Since this is the case, migration of virtual machines allows easier maintenance of physical resources by moving users away from machines that need inspection. It also allows for potential power savings by consolidating users on as few physical resources as possible. The hypervisor used in these experiments was KVM, which comes as a standard package with the Linux kernel. Virtual machine migration is not a specific property to the hypervisor used in these experiments. Live migration is one of the key features in most of the hypervisors available today. The process of virtual machine migration involves copying the entire state of the guest operating system on physical server A and transferring it to physical server B. When performing a virtual machine migration, a 64-bit host operating system can accommodate either a 32-bit or 64-bit guest operating system. On the other hand, a 32-bit host operating system will only accommodate a 32-bit guest operating system. KVM will not allow the user to perform the migrations if the guest and host operating systems are incompatible.
There are two types of virtual machine migrations, off-line migration and live migration. Off-line or “cold” migration involves saving the state and shutting down the guest operating system before transferring it to another physical server. Once the transfer is complete the virtual machine can be booted on the new server and the old copy can be removed from the original server.

Performing a virtual machine live migration is more complex than transferring a virtual machine using the off-line mode. First, a migrate request is offered on the server hosting the virtual machine. Once this request is processed, the host server connects to the destination server for initial setup. Next all memory pages are transferred to the destination server. The original host server continues to transfer memory pages until the virtual machine’s state converges on both servers. If the migration is successful, the virtual machine is shut down on the original host server. The virtual machine is now transferred to the destination server. The destination server will send a notification packet to update the location of the virtual machine. If the migration of the virtual machine was not successful to the destination server, the original server will send a notification packet as well to update the location of the virtual machine. In this case, the virtual machine is still being executed on the original host.
Chapter 3

POWER EFFICIENT COMPUTATIONAL RESOURCE LOAD CONSOLIDATION

3.1 Introduction

Local cloud implementations are becoming popular due to the fact that many organizations are reluctant to move their data to a commercialized cloud vendor. There are debates on whether moving data to the public cloud would benefit small organizations. Beyond the question of benefit to the organizations utilizing public clouds, there are also issues with trust, security and legality. Some organizations may not trust a third party with their information and/or software. Other organizations may not be comfortable allowing a third party to be responsible for the security of the cloud. Finally there are many organizations that work with data which they cannot legally store off site due to restrictions on the data. In cases such as these, the organization can opt to implement a cloud “in-house”.

There are several different implementations of open source cloud software that organizations can utilize when deploying their own private cloud. Some possible solutions are OpenNebula [OpenNebula, 2011b] or Nimbus [Nimbus, 2012]. However, the most common open source local cloud stack, and the one we will discuss primarily in this dissertation, is Eucalyptus [Eucalyptus, 2011b], provided by Eucalyptus Systems, Inc. This is an Infrastructure-as-a-Service (IaaS) cloud implementation that ultimately gives users virtual machines to undefined job types. A typical Eucalyptus cloud is composed of a front-end cloud controller, a cluster controller for administering compute nodes, a virtual machine image repository, persistent storage controller(s), and many com-
pute nodes. This architecture is built for ease of scalability and availability, but does not address the problem of the amount of power a typical architecture like this consumes.

Organizations that wish to build local clouds do so using commodity hardware. This may mean that the cloud is made up of several different hardware set ups. Even when a cloud is initially built using one type of hardware, the nature of a cloud often means it will be expanded by adding new and different hardware throughout the course of its lifetime. In terms of scalability, the amount of compute nodes will generally increase rapidly over time. Given this heterogeneous nature, the nodes used will have different energy consumption footprints. The administrative cloud components (cloud, cluster, and storage controllers) need to be continuously operating for users to access the resources provided by the compute nodes. This is not true of the compute nodes. Depending on the amount of requests given by users, it is not necessary to have all compute nodes operating at any given time.

Current research in the area of cloud computing load balancing focuses on the availability of resources. The specific load balancing approach depends on the type of resource offered. Since requests can be more specifically defined while using Software-as-a-Service (SaaS), the load balancing techniques used in this case may not be applicable to clouds offering IaaS architectures.

In this chapter, we propose a load consolidation algorithm that could be applied to the cluster controller of a local cloud that is power aware. This load consolidator maintains the utilization of all compute nodes and distributes virtual machines in a way that is power efficient. The goal of this algorithm is to maintain availability to compute nodes while reducing the total power consumed by the cloud.

The strategy to reduce wasted power in a local cloud architecture is to turn off compute nodes when they are not being used. A resource consolidation approach will be used to decrease
the amount of power used in the local cloud architecture. The default load balancing algorithm, round robin, does not account for power usage when compute nodes are idle. Compute nodes are placed in an idle state while they are not hosting any guest virtual machines in their hypervisor. Therefore there is room for reducing power consumption without affecting cloud resource availability. Compute nodes will default to a powered down mode, costing no energy while no
computational resources are being requested. The proposed Power Aware Load Consolidation (PALC) algorithm, described in the next section, resides on the cluster controller administrative node.

3.1.1 The Effect of Power Cycling on Cloud Resources

The strategy presented in this dissertation relies on cycling the power on computational resources instead of transitioning these resources into a low power state. Research was performed to determine if this has negative effects on the life cycle of the cloud components. Generally, components that are motor driven (hard drives, fans, CD-ROMs) will fail before solid state devices. Since local cloud resources are comprised of commodity devices, no extra redundancy was given to decrease failure rates in the experiment. Since there are several single points of failure, any failed component in a cloud node will result in loss of that specific cloud node.

The given mean time to failure is around 50,000 power cycles for standard mechanical SATA hard drives [Seagate, 2009]. We found specifications for Seagate and Western Digital 7200 rpm hard drives that give power cycles for their devices in this range. As for other components, we could not find manufacturer specifications on mean time to failure due to power cycles. As there were no research studies, that we could find, which gauge the limits of hardware due to power cycling, we decided to perform our own experiment with a small commodity system.

A small computer, “Turtle” [Michael Galloway, 2013], was built using commodity components valued at approximately $120 to test the effects of constant power cycling. Turtle consists of a Biostar Viotech 3200+ motherboard, 320GB Western Digital SATA hard disk drive, and a Dioblotek DA series 250W ATX power supply. The components chosen to build Turtle were deliberately low cost. This would be helpful in validating our argument that power cycling does not have an adverse effect on the life cycle of expensive components.
Table 3.1: Average Number of Power Cycles Per Day.

<table>
<thead>
<tr>
<th>Years</th>
<th>Average number of power cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>323</td>
</tr>
<tr>
<td>3</td>
<td>107</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
</tr>
</tbody>
</table>

Turtle has a heartbeat which lasts around two minutes. This heartbeat consists of a governing server powering on Turtle by a wake on LAN cron job every two minutes. Next Turtle boots Ubuntu Server 11.04 and a MySQL client process. Turtle then connects to the governing server and posts a timestamp to its MySQL server. Turtle then stays on for about one minute and powers off. The number of power cycles is determined by counting the number of timestamps in the MySQL database on the governing server. In effect, Turtle was continually powered on and off every two to three minutes.

We began the Turtle experiment on September 22, 2011. The system continued to power cycle every two to three minutes until August 20, 2012. This date was recorded as the death of Turtle with approximately 118,000 heartbeats. We use the word approximately because we manually restarted Turtle a few times to keep it running during the last month. Contrary to our expectations, the motherboard was the first component to fail. The fan failed, causing the motherboard to overheat and shutdown.

After eleven months of continuous two minute heartbeats, Turtle had over 118,000 successful power cycles. This gives an average of around 323 power cycles per day over the course of one year. Given that most organization will keep computational resources 3 -5 years, Table 3.1 calculates average number of power cycles per year up to five years. This experiment indicates that power cycling should have little to no effect on the lifecycle of the commodity cloud components.
The Turtle experiment provided us with anecdotal evidence to demonstrate the feasibility of our load and power consolidation approaches for power savings in local cloud computing architectures.

### 3.2 Power Aware Load Consolidation Algorithm

This algorithm is intended for organizations wanting to implement small to medium-sized local clouds. It should scale to larger-sized clouds because one of the main contributions of the cluster controller is load balancing compute nodes. Design of the infrastructure of the cloud computing architecture is important when determining scalability. When determining our local architecture, all compute nodes were confined to a local networking switch. This is the same for the persistent storage nodes, and the administrative nodes. This gives ease of scalability by isolating network traffic between common devices. Table 3.2 explains the variables used in the PALC algorithm.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Compute node</td>
</tr>
<tr>
<td>$n_j$</td>
<td>$j^{th}$ compute node utilization</td>
</tr>
<tr>
<td>m</td>
<td>Total set of available compute nodes</td>
</tr>
<tr>
<td>$D_t$</td>
<td>Deterministic threshold of compute node utilization</td>
</tr>
<tr>
<td>vm</td>
<td>Virtual machine</td>
</tr>
<tr>
<td>$vm_i$</td>
<td>$i^{th}$ virtual machine</td>
</tr>
</tbody>
</table>

All of the computation included in this algorithm is maintained in the cluster controller. The cluster controller maintains the utilization state of each active compute node and makes decisions on where to instantiate new virtual machines. A deterministic threshold is given, $D_t$, compute node utilization, to determine when a powered down compute node will become active. When a user requests computational resources, that request is handled by “powered on” compute nodes. Once all of the available compute nodes have reached $D_t$ utilization, a new compute node will power
on and become available. If a compute node terminates its last virtual machine, and not all other compute nodes are at $D_t$ utilization, that compute node will power off.

![Algorithm PALC](image)

**Algorithm PALC**

**Consolidate:**
for all active compute nodes $j \in [m]$ do:
    $n_j \leftarrow$ current utilization of compute node $j$
end for
if all $n_j > D_t$ utilization //all available nodes are active
    boot vm on most underutilized compute node
end if
else:
    boot vm on most utilized compute node
end else

**Upscale:**
if each $n > D_t$ utilization:
    if number of active compute nodes $< m$:
        boot next available compute node
    end if
end if

**Downscale:**
if $vm_i$ idle $> 6$ hours or user initiated shutdown:
    shutdown $vm_i$
end if
if active compute node has no active $vm$:
    shutdown active compute node
end if

Figure 3.2: PALC Algorithm.

The PALC algorithm Figure 3.2 has three basic sections. The balancing section is responsible for determining where virtual machines will be instantiated. It does this by first gathering the utilization percentage of each active compute node. In the case that all compute nodes $n$ are above $D_t$ utilization, PALC instantiates a new virtual machine on the compute node with the lowest utilization number. Otherwise, a new virtual machine (VM) is booted on the compute node with the
highest utilization (if it can accommodate the size of the VM). The threshold of $D_t$ utilization was chosen to have the most efficient number of compute nodes operating compared to the number of virtual machines deployed at any given time. It is worth mentioning in the case where all compute nodes are over 75% utilization, all of the available compute nodes are in operation.

The upscale section of the algorithm is used to power on additional compute nodes (as long as there are more available compute nodes). It does this if all currently active compute nodes have utilization over $D_t$.

The downscale section is responsible for powering down idle compute nodes. If the compute node is using less than 25% of its resources (no active virtual machines), PALC sends a shutdown command to that node.

3.3 Simulator Design and Setup

3.3.1 Compute Nodes

Next, we conducted experiments to determine the performance of the PALC algorithm. The performance metric used is the power consumed in terms of Watts.

In order to determine the power consumption to be used in our experiments, power consumption was measured from 15 commodity computers that could be used as compute nodes in a local cloud setup. A Watt-meter was used to gather data on these machines at idle and 100% utilization.

After the power profiling data was gathered from the different machines, they were averaged to give an approximate representation of the power requirement of a typical desktop machine. A scaling factor was then used to determine the power consumption at intermediate intervals between idle and 100% utilization. Figure 3.3 illustrates the power requirements needed by a single compute node in our cloud setup. The average power consumed by the commodity hardware tested
at <5% utilization was 76.16 Watts. These machines consume an average of 121.5 Watts at 100% utilization, giving a range of 45.35 Watts. The power consumption scale was calculated by dividing the difference, 45.35 Watts, by 100 to find the energy used at any utilization level between 0% and 100%.

![Figure 3.3: Power Requirements for Individual Compute Nodes.](image)

In our experiment, compute node utilization is composed of CPU and RAM used by virtual machines. Compute nodes are uniform dual-core CPU, 2 GB RAM machines. Although hard disk usage is included in our power measurements, hard disk space is not used in the computation of the utilization, since storage space far exceeds the need for the number of virtual machines that can be instantiated on these machines. KVM (Kernel-based Virtual Machine) [61] allows for 8 virtual machines per physical core. This experiment will allow for 6 virtual machines per core, giving 12 maximum virtual machines per compute node.

Accommodations were made for the underlying compute node operating system. Each compute node uses the Ubuntu 10.10 Server 64-bit operating system. The host operating system
and Eucalyptus software (including the KVM hypervisor) requires approximately 21.25% of the resources in the compute node. This approximation was computed by observing the actual memory usage of these software components on our cloud nodes. The underlying operating system was determined to need 1/8th of the CPU resources and 30% of the RAM in a 2 GB system.

3.3.2 Watt-Meter

The Watt-Meter is used to calculate the power consumption of the compute nodes over a given time period. This meter calculates the power consumption for each node in parallel and gives an output of the current load in Watts, the average load in Kilowatt-hours (kWh) and the total power consumed during a given time span. Figure 3.4 shows the simulator design in detail. The job scheduler has the ability to request instantiation and termination virtual machines. It sends these requests to the cloud controller, which submits the request to the PALC process hosted on the cloud cluster controller. The PALC then uses its deterministic algorithm to place a new virtual machine on a specific compute node. During the simulation, a Watt-meter is recording the power usage of all of the compute nodes.

3.3.3 Virtual Machine Configuration

As given in the Eucalyptus local cloud implementation, we allow the user to pick from five virtual machine sizes. The virtual machines will use Ubuntu 10.04 64-bit Server Edition with no graphical user interface. Since this is an IaaS cloud architecture, it is generally not known how the virtual machine will be used. In our study, we will assume a 100% utilization of the resources acquired by the virtual machines as long as they are operating. This utilization percentage ensures that the threshold $D_t$ never over-accommodates compute nodes due to idle or under utilized virtual machines.

In this design, a virtual machine will consume resources only on a single compute node,
although it is possible for one virtual machine to span multiple compute nodes. Table 3.3 gives the configurations options for the virtual machines in our cloud.

An IaaS cloud simulator was written for ease of deployment of multiple load consolidation
Table 3.3: Virtual Machine Configuration Options.

<table>
<thead>
<tr>
<th>VM Types</th>
<th>CPU</th>
<th>RAM</th>
<th>Disk</th>
<th>Utilization per Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>1</td>
<td>128</td>
<td>10</td>
<td>6.3%</td>
</tr>
<tr>
<td>m1.large</td>
<td>2</td>
<td>512</td>
<td>20</td>
<td>25%</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>2</td>
<td>1024</td>
<td>20</td>
<td>37.5%</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>4</td>
<td>1536</td>
<td>30</td>
<td>62.5%</td>
</tr>
</tbody>
</table>

techniques used for comparison. This software was written and executed on an actual compute node built specifically for our cloud.

3.3.4 Fluffy - A Local Cloud Implementation

A local cloud implementation, “Fluffy”, is used for running experiments. Fluffy is a constant progression in an attempt to build a private local cloud using low cost hardware. This cloud is built using the Open Eucalyptus cloud management software and Ubuntu Linux Server 11.04.

The major components of this cloud have been stated in chapter 2. This cloud has one cloud controller, one cluster controller for its single cluster of compute nodes, one walrus image controller, three persistent storage controllers, and five compute nodes used to host virtual machines.

There are four dual core compute nodes and one quad core compute node. Even though KVM allows for 8 virtual machines per CPU core, Fluffy is limited to 4 virtual machines per core. This gives it the ability to host 48 small-sized Linux virtual machines.

The local cloud architecture has been created with maximum bandwidth allocation in mind. The network uses multiple switches to isolate traffic to specific regions of the cloud. The persistent storage nodes are connected to a single switch. This keeps data movement between storage nodes isolated so it does not compete for backbone cloud bandwidth. The compute nodes are also connected to a single switch. This also helps reduce the amount of traffic on the local cloud backbone.
Last, the administrative nodes are connected to a third switch. Each of these switches are connected to the single Linksys WRT54GL router. Each of the switches used in the cloud is 1 Gbps. The router contains a 4-port 100 Mbps switch and a routing fabric capable of sustaining around 50 Mbps from the single WAN port to the LAN ports.

3.3.4.1 Job Scheduler

The job scheduler is used to simulate requests from users for virtual machine instances. Unlike clusters and grids, clouds have sporadic job schedules. A schedule was chosen that closely resembles requests for cloud resources.

Jobs in this experiment come as requests from users for virtual machines. There are five different sized virtual machines and users are not limited in the amount of resources each virtual machine allocates at launch, or the number of virtual machines they can request. If the user requests a virtual machine with specific resource allocations that cannot be accommodated by any compute node an error is returned.

3.4 Experiment

The experiment consisted of two similar job schedules. The first schedule consists of requests for up to 20 virtual machines. The second schedule requests up to 30 virtual machines. The job schedule distribution we used is common to requests of resources over a 24 hour period. Figure 3.5 shows the request schedule used in our experiment. As shown in the figure, the number of virtual machines requested peaks at 1pm for both 20 virtual machine and 30 virtual machine requests. The number of active virtual machines declines until 6pm. At that time, the number of active virtual machines increase linearly until 8pm. During the time period from 8pm to 10pm, the number of active virtual machines declines until there are no active virtual machines hosted on the compute nodes.
3.5 Evaluation and Results

The following figures give the results of our experiment runs. As stated previously, only the compute nodes that are powered on are used to determine the total power consumed over the 24 hour period. The load consolidator PALC is used to balance the virtual machines across the compute nodes in a way that conserves the most power.

3.5.1 Comparing Small Virtual Machine Requests using PALC

Figure 3.6 gives the average kWh over a 24 hour period for a job schedule composed of small virtual machines. While requesting 20 virtual machines of this size, the power consumed remains constant at 1.04 kWh. This is due to the fact that the cloud can handle this number of requests using only two compute nodes, so the increase in the number of available nodes had no
effect. The remaining compute nodes in each run remain powered down. When requesting 30 small virtual machines, the cloud can accommodate this workload using only three compute nodes. The power consumed for requesting 30 small virtual machines is consistently 2.1 kWh.

3.5.2 Comparing Large Virtual Machine Requests using PALC

Figure 3.7 shows the comparison of large sized virtual machine requests. Requesting 20 virtual machines of this size consumed 2.27 kWh with five compute nodes. The cloud could only accommodate 15 requests of this size while having only five compute nodes. This means the remaining 5 requests were refused. The power increased to 2.95 kWh when the number of compute nodes increased to 10, 15, and 20. The cloud needed seven compute nodes to accommodate all 20 requests. The power consumed for 30 requests of large virtual machines was 4.93 kWh while having five compute nodes available. When the cloud had 10, 15, and 20 compute nodes available, the power consumption increased to 6.57 kWh. The power consumption remains the same since 30 requests of this size can be launched with 10 compute nodes.
3.5.3 Comparing Extra-Large Virtual Machine Requests using PALC

Figure 3.8 shows the power consumed when requesting virtual machines of extra-large size. While requesting 20 virtual machines of this size and a maximum compute node number of five, the power consumed was 3.5 kWh. When the maximum number of compute nodes was increased to 10, the power consumed was 5.88 kWh. At a maximum of 15 compute nodes, the power consumed was 7.3 kWh. When a maximum of 20 compute nodes, the power consumed increased to 7.78 kWh.

As assumed, the cloud compute nodes consumed the most power when the job scheduler demands 30 extra-large virtual machines over a 24 hour period. When the cloud is constrained to five compute nodes, it consumed 5.9 kWh and refused requests. With 10 compute nodes, the power consumption was 10.2 kWh. With 15 compute nodes, the power increased to 13.53 kWh.
Lastly, the highest recorded power consumption occurred when 30 extra-large virtual machines were requested with 20 compute nodes. The power consumed during this test was 15.9 kWh.

3.5.4 Comparing Incrementally Sized Virtual Machine Requests using PALC

Figure 3.9 shows the comparison of incrementally sized virtual machine requests over the 24 hour period. Each new virtual machine request in this run is larger than the previous, returning to the smallest size once an extra-large request is made. With a 20 virtual machine request schedule and having a maximum of five compute nodes, the cloud consumed 2.86 kWh. When the maximum number of compute nodes was increased to 10, 15, and 20, the power consumed was 3.37 kWh. With a 30 virtual machine request schedule and a maximum of five compute nodes, 5.1 kWh of power was consumed. When the maximum number of compute nodes was increased to 10, the power consumed was 7.36 kWh. Lastly, when the number of compute nodes increased to 15 and 20, the power consumed was 7.64 kWh.
3.5.5 Comparison of Virtual Machine Requests using PALC

The figure 3.10 give the comparison of the different sized virtual machine requests using the PALC algorithm. This power comparison is again based on the virtual machine job schedule given in Figure 3.5. As expected, the small virtual machine requests consume the least amount of power over the 24 hour period. The linear regression model for the small virtual machines with maximum of 20 instantiations can be shown as:

\[ y = 6E - 17x + 1.0384 \]

While requesting a maximum of 30 small virtual machines, the compute nodes again show a linear regression of power usage. This can be shown as:

\[ y = 1E - 16x + 2.1044 \]
When moving to the large virtual machine types, the power consumption increased over deploying only small virtual machine types. The power consumed by instantiating up to 20 large virtual machines can be shown as:

\[ y = 0.0131x + 2.7275 \]

The amount of power consumed becomes noticeably greater when deploying a maximum of 30 large virtual machines. To show this, the polynomic regression is given:

\[ y = 0.0022x^3 - 0.0984x^2 + 1.4219x + 0.0108 \]

Deploying the extra large virtual machines consumed the most power compared to the other experiments. The power consumption of deploying a maximum of 20 extra large virtual machines can be shown as:
\[ y = -0.019x^2 + 0.7609x + 0.1712 \]

Deploying a maximum of 30 extra large virtual machines consumes the most power, since the virtual machines consume the most resources compared to the other virtual machine types. To show the power consumed, the polynomic regression is shown as:

\[ y = -0.0192x^2 + 1.1473x + 0.642 \]

While deploying the incrementally sized virtual machines, the power requirements for 20 maximum virtual machines was slightly higher than deploying 20 large virtual machines. This is due to the accommodation of deploying extra large virtual machines. The power consumption can be shown as:

\[ y = -0.0051x^2 + 0.1587x + 2.2157 \]

The deployment of 30 incrementally sized virtual machines requires more energy than all other sizes, except deploying 30 extra large virtual machines. The power consumption can be shown as:

\[ y = 0.0023x^3 - 0.1078x^2 + 1.6724x - 0.8571 \]

3.5.6 Comparison of PALC and Round Robin Load Balancer

The results obtained by using our PALC algorithm would obviously bring higher power savings over conventional load balancing algorithms, such as the “round robin” approach used by Eucalyptus. In the round robin approach, the cluster controller assigns virtual machine requests to compute nodes sequentially. This effectively balances the load across the compute nodes, but
leaves these nodes in an “on” state with usually low utilization. Figure 3.11 shows the power consumption of the round robin load balancing algorithm (RRLB).

The major performance difference between our PALC algorithm and the round robin approach is that compute nodes that are idle using PALC are powered down. The round robin approach always keeps all compute nodes powered on, no matter the number of active virtual machines. The round robin approach is effective in load balancing across the available compute nodes, but such a relatively simple load balancer consumes a large amount of unnecessary power.

In comparison, when the cloud has 5 compute nodes and 20 small virtual machines are requested, PALC consumes 11% of the energy consumed for round robin with the same parameters. When requesting 20 small virtual machines while having 20 available compute nodes, PALC only uses 2.8% of the energy consumed by round robin. Using the requests for extra-large virtual machines and five available compute nodes, PALC uses 29.5% of the energy consumed by round robin. Requesting 20 extra-large virtual machines with 20 compute nodes available, PALC consumes 20.6% of the energy used by round robin.

3.6 Discussion

Given the results in the previous section, a percentage breakdown of the efficiency of PALC as compared to RRLB is given in Table 3.4. In Table 3.4, S20 indicates 20 small virtual machines, S30 indicates 30 small virtual machines, L20 is 20 large virtual machines, L30 is 30 large virtual machines, XL20 is 20 extra-large virtual machines, XL30 is 30 extra-large virtual machines, R20 is 20 incrementally sized virtual machines and R30 is 30 incrementally sized virtual machines. Subsequent savings are shown as PALC is compared to RRLB executing the same job schedules.

While using 20 compute nodes and requesting 20 small virtual machines, PALC had the largest advantage over RRLB with a savings of 97.2% power consumed. PALC had the smallest
advantage over RRLB while requesting 30 random-sized virtual machines with access to 5 compute nodes. In this case, PALC saved 54.8% of the power used by RRLB to perform the same job.

Table 3.4 illustrates the power consumed by PALC as compared to RRLB while executing the same jobs. While requesting 20 small virtual machines, the amount of power consumed by PALC ranges from 2.8% - 11% of the power consumed by RRLB. This means PALC has a power savings of 89% - 97.2% compared to RRLB. When the number of small virtual machines is increased to 30, the power used by PALC ranges from 5.66% - 21.76% of the power used by RRLB. This is a power savings of 78.24% - 94.33%. The increase in power consumption by both PALC and RRLB comes from the fact that more physical compute nodes are needed to accommodate
Table 3.4: Power Savings of PALC as Compared to RRLB.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>S20</th>
<th>S30</th>
<th>L20</th>
<th>L30</th>
<th>XL20</th>
<th>XL30</th>
<th>R20</th>
<th>R30</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>89</td>
<td>78.2</td>
<td>73.8</td>
<td>55.8</td>
<td>71.5</td>
<td>58.4</td>
<td>72.6</td>
<td>54.8</td>
</tr>
<tr>
<td>10</td>
<td>94.4</td>
<td>88.8</td>
<td>84.9</td>
<td>67.7</td>
<td>72.6</td>
<td>56.7</td>
<td>82.8</td>
<td>64</td>
</tr>
<tr>
<td>15</td>
<td>96.2</td>
<td>92.5</td>
<td>89.7</td>
<td>77.7</td>
<td>76.1</td>
<td>58.4</td>
<td>88.3</td>
<td>74.2</td>
</tr>
<tr>
<td>20</td>
<td>97.2</td>
<td>94.3</td>
<td>92.2</td>
<td>80.4</td>
<td>80.4</td>
<td>61.8</td>
<td>91.1</td>
<td>80.3</td>
</tr>
</tbody>
</table>

Table 3.5: Average Power Used by PALC as Compared to RRLB Over All Job Schedules and Available Compute Nodes.

<table>
<thead>
<tr>
<th>S20</th>
<th>S30</th>
<th>L20</th>
<th>L30</th>
<th>XL20</th>
<th>XL30</th>
<th>R20</th>
<th>R30</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>9</td>
<td>12</td>
<td>24.8</td>
<td>23.5</td>
<td>40.7</td>
<td>13.4</td>
<td>27.8</td>
</tr>
</tbody>
</table>

30 small virtual machines. The power used by PALC when requesting 20 large virtual machines ranges from 7.8% - 26.24% of the power used by RRLB. Requesting 30 large virtual machines uses only 17.03% - 44.17% of the power used by RRLB. When the job schedule requests 20 extra-large virtual machines the power used by PALC is only 18.59% - 28.46% of the power used by RRLB. Requesting 30 extra-large virtual machines brings the power savings of PALC over round robin to around 57%. It should be noted here that not all requests for extra-large virtual machines can be accommodated. Each extra-large virtual machine that is instantiated on a physical compute node requires over 62% of that node’s resources. Combined with the 21% utilization of the compute node by the underlying operating system and hypervisor, each extra-large virtual machine request brings a compute node’s utilization to over 80%.

This table demonstrates that even in the worst case, PALC consumes 55% less power than RRLB while handling the same job schedule. As the local cloud architecture scales in size, idle compute nodes will consume a larger amount of wasted power.

Table 3.5 shows the average power consumed over all combinations of available compute nodes. On average, while requesting 20 small virtual machines, PALC has a savings of 95.6%. The
power aware load balancer has the least amount of average savings when requesting 30 extra-large virtual machines.

The average power consumption decreases with PALC when compared to RRLB over all experiments is 79.9%. This is a significant savings in power consumed while keeping the availability of the cloud resources as high as possible.

As with any load balancing algorithm, if efficiency is given towards becoming power aware, there will be trade-offs in other areas. Since it is the case that if a customer requests a virtual machine that cannot be accommodated for in the active compute nodes, they will experience latency as a new compute node powers on. However, PALC is designed to minimize power transition state changes. In the proposed power aware architecture, accommodations are given to reduce the total power consumed by the cloud architecture, thereby reducing the monetary costs of operation. The
default state for compute nodes in this architecture is to be in a powered off state. These compute nodes become active depending on the number of virtual machine requests given by the cloud controller. If a user requests a virtual machine, and that virtual machine will fit on an already active compute node, there will be no overhead latency other than waiting for that virtual machine to boot. In the other case, if a user requests a virtual machine that in turn creates a non active compute node to become active, the user will have to wait for the physical machine to boot the host operating system before the virtual machine can be deployed and booted.

The results from the data analysis of the load balancers revealed that making them power aware positively affects the cost of operation as clouds scale to size. As shown in the analysis, the PALC scheme reduces the energy consumption of compute nodes by 79.9% on average compared to RRLB.
Chapter 4

POWER AWARE LOAD CONSOLIDATION - WITH VIRTUAL MACHINE MIGRATION

4.1 Introduction

Live migration allows resources from one physical server to be moved to another with little or no interruption in the processes of the guest operating system. The process involved in performing a live migration includes copying the guest virtual machine memory state and CPU register state from a hypervisor on one server to another. In this chapter, virtual machine migration is evaluated in terms of performance of the virtual machine while being migrated, performance of the cloud architecture while migrating the virtual machine, and the power costs of performing a live virtual machine migration.

Migrating a virtual machine is an efficient way of moving a guest operating system and all of the processes associated with it between physical hosts. Having this ability gives cloud administrators the ability to distribute cloud requests depending on current situations. One strategy of virtual machine live migration gives the ability to balance resources across the cloud's physical servers. Computational load balancing is an approach where virtual machine performance is given priority at the expense of power consumption. Computational load balancing also alleviates the hot spots of utilization on compute nodes. Another strategy of live migration is the ability to consolidate virtual machines on as few physical resources as possible, giving the advantage to reducing the power consumed by the cloud. Both of these strategies have advantages and disadvantages, allowing a dynamic solution of switching between both approaches a viable solution. Virtual ma-
chine live migration also allows cloud administrators to move virtual machines away from specific servers if they need maintenance. In this chapter, the focus of virtual machine live migration will be to apply a consolidation technique to reduce the power consumed by the cloud architecture. The experiments on virtual machine live migration later in this chapter will give insight on how the cloud architecture responds and handles live migrations, and how virtual machines perform while they are in a migration phase.

To perform a virtual machine live migration, a technique called pre-copy [Clark, Fraser, Hand, Hansen, Jul, Limpach, Pratt, and Warfield, 2005], [Strunk, 2012] is used. This live migration strategy is similar across the different hypervisors that accommodate virtual machine migration. Also, the strategy of performing a live migration is different than the strategy of performing an off line migration. The off line migration pauses the virtual machine on the originating server while it copies the memory contents to the destination server. This approach is noticeable by the end user, and is the reason it is not selected as the migration technique in this chapter. Using a live virtual machine migration, the memory and cpu state transfer can be generalized into three phases as shown in Figure 4.1:

1. **Push Phase:** The virtual machine is still being hosted by server A while its memory is iteratively copied to server B. This is an iterative process consisting of multiple rounds of data transfer since pages that have been sent to server B may have been updated on server A. To ensure consistency, these dirty pages must be resent.

2. **Pre-Copy Termination Phase:** The push phase requires a stop condition for the termination phase since it is an iterative process. These stop conditions depend on the system architecture...
and hypervisor used. Some stop conditions include the number of performed memory push iterations, total memory transferred, number of dirty pages, etc.

3. **Pull-and-Terminate Phase:** During this stage the hypervisor suspends the virtual machine on server A. The remaining dirty pages are transferred to the hypervisor on server B. Once this migration is complete, the hypervisor on server B resumes the virtual machine.

![Figure 4.1: Overview of Live Migration Performing Iterative Memory Transfers.](image)

4.2 Virtual Machine Live Migration Costs

Our approach to virtual machine consolidation involves live migration of these virtual machines. In order to reduce the overall costs of maintaining these virtual resources, upfront costs will be incurred by consolidating the virtual machines. It has been shown [Voorsluys, Broberg, Venugopal, and Buyya, 2009], [Wu and Zhao, 2011] that performance degradations are common when performing a live migration on a large scale. Studies have also been performed to show an increase in power consumption on sending and receiving physical servers during a virtual machine live migration [Huang, Gao, Wang, and Qi, 2011].
The first issue with live migration is how long it actually takes to move the virtual machine from server A to server B. This will depend on several factors including network bandwidth, CPU utilization, and Disk I/O. The most common performance aspects of moving a virtual machine includes the migration time and virtual machine inaccessibility time. Another issue with virtual machine migration is how the migration effects running services on the guest operating system. The iterative process of transmitting memory pages between physical hosts will consume a considerable amount of network bandwidth. We take this into consideration by isolating compute node resources, allowing live virtual machine migrations to affect only the bandwidth between compute nodes. Lastly, the energy consumption of both server A and server B may increase during a virtual machine migration. This is due to the fact that more resources are needed during the migration than are otherwise needed. Table 4.1 gives the parameters and costs that may be included when performing a virtual machine live migration.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_{tms}$</td>
<td>Total time for migration (ms)</td>
</tr>
<tr>
<td>$V_{tms}$</td>
<td>Total inaccessible time for vm (ms)</td>
</tr>
<tr>
<td>$E_w$</td>
<td>Power consumed during vm migration (W)</td>
</tr>
<tr>
<td>$MeM_{mig}$</td>
<td>Total amount of Memory transmitted from source to destination</td>
</tr>
<tr>
<td>$P_t$</td>
<td>Increase in process execution time during vm migration (%)</td>
</tr>
<tr>
<td>$P_{th}$</td>
<td>Decrease in process throughput during vm migration (%)</td>
</tr>
<tr>
<td>$CPU_{util}$</td>
<td>CPU utilization (%)</td>
</tr>
<tr>
<td>$N_u$</td>
<td>Network utilization (bps)</td>
</tr>
<tr>
<td>$VM_{mem}$</td>
<td>Virtual machine size (Mb)</td>
</tr>
<tr>
<td>$DP_r$</td>
<td>Dirty page rate (pages/sec)</td>
</tr>
<tr>
<td>$L_{ab}$</td>
<td>Link bandwidth (bps)</td>
</tr>
<tr>
<td>$n$</td>
<td>Number of memory migration iterations before pull-and-terminate phase begins</td>
</tr>
<tr>
<td>$M$</td>
<td>Mean time between two migrations of the same virtual machine (s)</td>
</tr>
</tbody>
</table>
4.2.1 Live Migration Costs

This section covers the costs associated with virtual machine live migration costs. As shown in Figure 4.2 the most common costs include the performance of the cloud architecture during the virtual machine migration, the decrease in performance of the virtual machine while being moved, and the power consumed to perform the migration.

![Figure 4.2: Overview of Virtual Machine Live Migration Costs.](image)

The performance of the migration is generally expressed in how long it takes the local cloud architecture to move the virtual machine from server A to server B. The total time refers to the amount of time it takes from the beginning of the migration process on the source server until the virtual machine is accessible on the destination server. Many studies have come to the conclusion that the total migration time $M_{tms}$ is mostly affected by the amount of memory $MeM_{mig}$ that was sent from source to receiving server, and how much network bandwidth $L_{ab}$ is available. This time is deterministic based on the fact that it increases linearly with the ratio of $MeM_{mig}$ related to $L_{ab}$. This total time can be calculated as:

$$M_{tms} = \frac{MeM_{mig}}{L_{ab}}$$

Table 4.1 lists the notion used in the analysis of virtual machine live migration throughout
this chapter. When performing a virtual machine live migration, the hypervisor on server A sends the entire RAM contents of the virtual machine to server B. During this time, server A is still hosting the virtual machine and pages can become dirty. This makes an iterative approach to migration necessary. The total amount of memory moved from server A to server B is determined by how much memory is migrated in each round $i$, and the total number of rounds $n$ before the pull-and-terminate begins. The total amount of data transmitted from server A to server B during the virtual machine migration can be calculated as:

$$MeM_{mig} = \sum_{i=1}^{n+1} MeM_{migi}$$

The authors of [Akoush, Sohan, Rice, Moore, and Hopper, 2010] have determined the minimum and maximum bounds for the time it takes to perform a virtual machine live migration. These bounds are deterministic, based on the resource utilization of the virtual machine that is to be migrated. For instance, a virtual machine that is idle (no running processes or I/O) will not be modifying its memory state on the source server. In this case, the number of migration iterations would be one, since the source server can send the memory contents of that virtual machine with no dirty pages to send in the second iteration of the migration process. Otherwise, in the worst case, the virtual machine may be in a very active state. For example, the virtual machine may be dirtying every memory page during the migration process on the source server. In this case, the migration iteration process must reach a termination phase, and stop the virtual machine on the source server and resume it on the destination server. As shown in [Akoush et al., 2010], the minimum and maximum bounds of performing a virtual machine live migration can be shown as:

$$\frac{VM_{mem}}{L_{abh}} \leq M_{tms} \leq \frac{VM_{mem\cdot(n+1)}}{L_{abh}}$$
The total time for migration is bound by the time it takes to transfer the entire memory set of the virtual machine once and the time it takes to transfer the virtual machine n times. The link bandwidth and size of the virtual machine are determining factors in the time it takes to perform a virtual machine live migration.

Another important aspect of virtual machine live migration performance is the duration in which the virtual machine is inaccessible. When the hypervisor on server A stops the virtual machine, this starts the time in which that virtual machine is unavailable. This time is concluded when server B starts the virtual machine. During the Pull-and-Terminate phase, server B will have to fetch the last set of dirty pages from server A. The inaccessibility time of the virtual machine is related to the dirty page rate $DP_r$, page size, $l$, the average network transmission rate $L_{ab}$, and the time taken of the last Push Phase, $T_n$. The authors of [Liu, Xu, Jin, Gong, and Liao, 2011a] define the total inaccessible time, $V_{ims}$, as:

$$V_{ims} = \frac{DP_r \cdot T_n \cdot l}{L_{ab}}$$

The inaccessibility time of the virtual machine is bound by the amount of dirty pages that have to be copied from server A to server B during the Pull-and-Terminate phase [Akoush et al., 2010]. The minimum and maximum bounds (ms) that a virtual machine may be inaccessible can be expressed as:

$$0 \leq V_{ims} \leq \frac{VM_{mem}}{L_{ab}}$$

In the case that the virtual machine is idle, it will be immediately available on server B after the migration. This is due to the fact that there will be no dirty pages to copy from server A before starting the virtual machine on server B. In the worst case, the entire memory contents will have to
be copied from server A to server B during the Pull-and-Terminate phase, due to high activity of
the virtual machine leaving all memory pages dirty.

4.2.2 Virtual Machine Performance During Migration

Even though the performance of the cloud infrastructure is often analyzed, the performance
of the virtual machine that is being migrated is not often evaluated. The migration of the virtual
machine will have a negative impact on the performance of that virtual machine. This is due to
the fact that the resources on the source server are more utilized during the migration process. The
network throughput to the virtual machine is also affected due to moving memory pages between
servers. In the experiments given later in this chapter, the affect of performing a live migration on
the virtual machine will be discussed. It is also true that a very active virtual machine will affect
the performance of the migration, due to high network utilization.

4.2.3 Power Consumption During Migration

The energy overhead due to performing a virtual machine live migration can be analyzed as
being proportional to the amount of memory copied from server A to server B. The energy model
assumes homogeneous hosts, where the energy consumed by the sending server is equivalent to
the energy consumed by the receiving server. The following energy model [Liu et al., 2011a] is
proposed:

\[ E_w = \alpha \times MeM_{mig} + \beta \]

The described energy model is consistent with all migrations being performed on homoge-
neous servers. The energy consumed in the switching fabric is difficult to quantify, so the energy
model only describes the power consumed by the servers. The parameters \( \alpha \) and \( \beta \) were derived
using a linear regression technique from [Liu et al., 2011a] to be \( \alpha = 0.512 \) and \( \beta = 20.165 \). \( \alpha \) de-
scribes the amount of energy required while performing the virtual machine migration and $\beta$ gives the amount of data transferred. This gives us the slope-intercept formed equation for determining $E_w$ based on the amount of memory transmitted from server A to server B, $MeM_{mig}$:

$$E_w = 0.512 \times MeM_{mig} + 20.165 \quad [\text{Liu et al., 2011a}]$$

The amount of power consumed by the servers performing the live migration consists of static and dynamic sinks [Strunk, 2012]. The static power consumption is due to the inefficiencies of the physical server (leakage current of transistors, heat generated, cooling appliances, etc.). The dynamic power sinks include resources such as CPU, RAM, and Disk utilization. To further increase our knowledge of the power consumption during a virtual machine migration, $E_w$, we can also obtain this information by determining the power consumed before the live migration begins ($E_{before}$) and subtracting that from the power consumed during the live migration ($E_{during}$):

$$E_w = E_{during} - E_{before} \quad [\text{Strunk, 2012}]$$

4.3 Experiment Setup

In this experiment, aspects of performing a virtual machine live migration will be evaluated. As stated previously, there are three major aspects of virtual machine migration that are of concern. These are migration performance of the local cloud architecture in terms of migration time, performance of the virtual machine as it is being moved from server A to server B, and the amount of energy consumed by the cloud architecture while performing the virtual machine live migration. Each of these aspects will be evaluated independently of each other, combining the results to suggest a better understanding of the costs and benefits of performing a virtual machine live migration. With the results of these experiments, the power aware load consolidation (PALC)
Table 4.2: Virtual Machine Descriptions.

<table>
<thead>
<tr>
<th>VM ID</th>
<th>VM Type</th>
<th>CPU Cores</th>
<th>RAM Size</th>
<th>Disk Size</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base1</td>
<td>Non-utilized Base VM</td>
<td>1</td>
<td>256MB</td>
<td>2GB</td>
<td>192.168.1.10</td>
</tr>
<tr>
<td>Base2</td>
<td>Non-utilized Base VM</td>
<td>1</td>
<td>512MB</td>
<td>5GB</td>
<td>192.168.1.11</td>
</tr>
<tr>
<td>Base3</td>
<td>Non-utilized Base VM</td>
<td>2</td>
<td>1024MB</td>
<td>10GB</td>
<td>192.168.1.12</td>
</tr>
<tr>
<td>Web1</td>
<td>Apache Web Server</td>
<td>1</td>
<td>256MB</td>
<td>2GB</td>
<td>192.168.1.13</td>
</tr>
<tr>
<td>Web2</td>
<td>Apache Web Server</td>
<td>1</td>
<td>512MB</td>
<td>5GB</td>
<td>192.168.1.14</td>
</tr>
<tr>
<td>Web3</td>
<td>Apache Web Server</td>
<td>2</td>
<td>1024MB</td>
<td>10GB</td>
<td>192.168.1.15</td>
</tr>
<tr>
<td>DB1</td>
<td>MySQL Database Server</td>
<td>1</td>
<td>256MB</td>
<td>2GB</td>
<td>192.168.1.16</td>
</tr>
<tr>
<td>DB2</td>
<td>MySQL Database Server</td>
<td>1</td>
<td>512MB</td>
<td>5GB</td>
<td>192.168.1.17</td>
</tr>
<tr>
<td>DB3</td>
<td>MySQL Database Server</td>
<td>2</td>
<td>1024MB</td>
<td>10GB</td>
<td>192.168.1.18</td>
</tr>
<tr>
<td>Pi1</td>
<td>Computing Pi</td>
<td>1</td>
<td>256MB</td>
<td>2GB</td>
<td>192.168.1.19</td>
</tr>
<tr>
<td>Pi2</td>
<td>Computing Pi</td>
<td>1</td>
<td>512MB</td>
<td>5GB</td>
<td>192.168.1.20</td>
</tr>
<tr>
<td>Pi3</td>
<td>Computing Pi</td>
<td>2</td>
<td>1024MB</td>
<td>10GB</td>
<td>192.168.1.21</td>
</tr>
</tbody>
</table>

The algorithm will be adjusted to accommodate migration of virtual machines in order to decrease the overall power consumption of the local cloud architecture.

The devices used in these experiments include Dell Optiplex 755 virtual machine hosts connected by a 1Gbps switch. Ubuntu 12.10 with kvm and virsh installed are used to perform live migrations. Each of the virtual machine images are located on an image server, with directories containing the images mounted on each of the virtual machine hosts. This is necessary to perform live migrations of the virtual machines without transferring the virtual machine’s base image. Each virtual machine type has three sizes associated with it when being deployed. The first size is deployed with 256MB of RAM, 2GB of local disk space, and one virtual CPU core. The second size is deployed with 512MB RAM, 5GB of local disk space, and one virtual CPU core. The last, and largest size, is deployed with 1024MB RAM, 10GB of local disk space, and two virtual CPU cores.

There are four different types of virtual machines used in determining the various performance and costs associated with a live migration. These types include a base instance virtual
machine, a virtual web server running Apache and PHP, a virtual database server running MySQL, and a virtual cluster computing Pi to the $n^{th}$ place.

The first type is a base case. This is a non-utilized Ubuntu server virtual machine with parameters shown in Table 4.2. This virtual machine has the only the default processes running that are installed with Ubuntu server 12.10, and SSH. Performance measurements for this virtual machine consists of evaluating CPU and RAM resources during the migration and comparing them to these resources consumed while not being moved to another server.

The second virtual machine type is the Apache Web Server. When accessed, this web server will respond with the PHP info script which displays the specific installation details of PHP. To test the performance of this virtual web server, a program, http_load is used. http_load runs multiple http fetches in parallel in order to test the durability and throughput of a web server. The results of http_load will be compared to performing the same tests on the virtual web server while not performing a live migration.

The third virtual machine type is the MySQL Database Server. To test the performance of the MySQL database, and inherently the virtual machine, a program, sysbench is used. Sysbench is a benchmarking suite that gathers performance information on CPU, File I/O, and the MySQL database while under heavy load. To test the MySQL database, a table with 100,000 rows of data is created. Sysbench evaluates the performance as transactions per second of the MySQL database. These performance evaluations are then compared with a virtual MySQL database server while not performing a live migration.

The last virtual machine type uses a computationally intensive algorithm computing Pi to the $n^{th}$ place. To do this, a jython program implementing the Chudnovsky algorithm is used. Performance is evaluated as the time taken to compute Pi to the $n^{th}$ place. This performance
evaluation is then compared to the virtual machine performing the same calculations while not
performing a live migration.

4.4 Results of Live Migration

The following subsections give the results of performing a live migration in terms of virtual
machine performance, cloud physical machine performance, and power consumption.

4.4.1 Virtual Machine Live Migration Performance

This section describes the performance of the physical resources used to host and migrate
virtual machines. When instantiating a live migration, a stand alone server is used to ensure the
migration between the physical servers is not altered. Each migration evaluation is performed 10
times, and the averages are given in the figures below.

4.4.1.1 Performance of Migrating the Base Instance

The base instances are evaluated for physical cloud resource performance. These instances
have no active processes except the ones required for the operating system.

![Figure 4.3: Performance of Migrating the Base Instance (Lower is better).](image)
Figure 4.3 gives the performance of live migrating the base instances. The smallest virtual machine, base1, has a live migration time of 9.98 seconds. The base2 virtual machine has a similar migration performance of 10.51 seconds. The base3 virtual machine includes 1GB of RAM and two virtual CPU cores. This instance is migrated on average in 26.99 seconds.

4.4.1.2 Performance of Migrating the Web Server Instance

To evaluate the performance of the physical cloud resources while performing a live migration of the virtual web server, the migration performance is examined while there are no HTTP requests to the virtual machine. As shown in figure 4.4, the Web1 and Web2 virtual machines have similar migration performance of around 12 seconds. The larger Web3 virtual machine takes just under 14 seconds to be migrated while handling no HTTP requests. These base evaluations can be compared to the cloud’s performance when migrating these virtual machines while they are handling various HTTP requests.

![Figure 4.4: Performance of Migrating the Web Instance While Idle (Lower is better).](image-url)
Figure 4.5 gives the performance of the cloud infrastructure as it migrates the Apache web server virtual machine while accommodating HTTP requests. To saturate the web service on the virtual machine, the number of concurrent HTTP requests is increased until network saturation occurs. The Web2 and Web3 virtual machines show similar trends as the number of concurrent HTTP requests are increased. Network saturation of the 1Gbps backbone becomes apparent when the number of concurrent HTTP requests reaches 100.

The Web1 virtual machine is constrained by the amount of resources allocated before booting. This virtual machine has the lowest memory footprint, giving it the ability to be migrated easily, but has severe limitations on its productivity. The tests of 10, 25, and 50 concurrent HTTP requests are show in figure 4.5. All other concurrent HTTP request tests fail using Web1. We note here that even when the Web1 virtual machine fails to migrate, the original virtual machine continued to function, so the user still has access to the web server.

4.4.1.3 Performance of Migrating the MySQL Database Instance

Performance of the physical cloud resources while migrating the virtual database servers is given in figure 4.6.

The performance of migrating the DB1 and DB2 virtual machines are similar. In the best case, the DB1 virtual machine is migrated in 10.25 seconds. In the worst case, it takes 18.27 seconds to migrate DB1. The performance of the cloud while migrating DB1 can be shown as:

Cloud Performance of Migrating DB1: \( y = -2E^{-5}x^3 + 0.0043x^2 + 10.552 \)

While migrating the DB2 virtual machine, the best case was 8.36 seconds. In the worst case, the cloud resources took 16.89 seconds to migrate the DB2 virtual machine. The performance of the cloud while migrating DB2 can be shown as:
Figure 4.5: Performance of Migrating the Web Instance (Lower is better).

Cloud Performance of Migrating DB2: \(y = -3E^{-5}x^3 + 0.0049x^2 + 8.6984\)

Performing a migration on the DB3 virtual machine is more stable than that of DB1 and DB2. This virtual machine has a lower performance on the cloud resources during a migration due to its large amount of allocated resources. In the best case, DB3 was migrated in 22.32 seconds. In the worst case it took 23.93 seconds to migrate DB3. The performance of the cloud while migrating DB3 can be shown as:

Cloud Performance of Migrating DB3: \(y = -0.0123x + 22.422\)

4.4.1.4 Performance of Migrating the Pi Computation Instance

The migration performance of the pi computation virtual machines is given in figure 4.7.
Figure 4.6: Performance of Migrating the Database Instance (Lower is better).

Figure 4.7: Performance of Migrating the Pi Computation Instances (Lower is better).
The Pi2 virtual machine shows a better performance while being migrated when compared to the Pi3 virtual machine. This is due to the lower amount of physical resources given to it when launched. The performance of the cloud while migrating the Pi2 virtual machine can be shown as:

\[ y = 1E - 25x^5 - 7E - 20x^4 + 1E - 14x^3 - 1E - 9x^2 + 6E - 5x + 11.088 \]

Migrating the Pi3 virtual machine can be shown as:

\[ y = -6E - 25x^5 + 2E - 19x^4 - 4E - 14x^3 + 2E - 9x^2 - 3E - 5x + 12.493 \]

4.4.2 Virtual Machine Performance During Live Migration

In this section, the performance of the virtual machine that is being migrated will be evaluated. The performance evaluations parameters evaluated are listed in Table 4.3 and vary depending on which virtual machine type is being evaluated. Performing a live migration will never have a positive effect on the virtual machine that is being moved. In the best case, the effect will be negligible and unnoticed by the user.

<table>
<thead>
<tr>
<th>VM Type</th>
<th>How Evaluated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base VM</td>
<td>CPU and RAM Utilization</td>
</tr>
<tr>
<td>Apache Web Server</td>
<td>HTTP Requests Response Time</td>
</tr>
<tr>
<td>MySQL Database Server</td>
<td>Query Responses per Unit Time</td>
</tr>
<tr>
<td>Computing Pi</td>
<td>Time Taken to compute Pi to ( n^{th} ) Place</td>
</tr>
</tbody>
</table>

4.4.2.1 Performance of Base Instance - CPU Utilization

The first virtual machine to be evaluated is the base image machine. This virtual machine only varies in the amount of CPU cores, RAM, and disk space allocated when launched. Using the sar command in the sysstat package, the average CPU and RAM utilization was obtained before and during a live migration. Figure 4.8 shows the CPU utilization of all three base virtual machines
where CPU UtilA is the CPU utilization of the idle instance while not in a migration. CPU UtilB shows the utilization while the instance is in a migration. As shown in figure 4.8, the average CPU utilization for the base image virtual machines is 0% while not in the migration process and so is not visible in the figure. During the live migration the average CPU utilization is still negligible at less than 0.6% in all instances.

![Figure 4.8: Base VM CPU Utilization.](image)

The RAM utilization of the base virtual machine images is given in figure 4.9. Mem UtilA in the figure gives the utilization while not in a migration. Mem UtilB shows the memory utilization while in a migration. The base1 instance is given 256MB RAM when launched, and consumes 20.88% of this when fully booted. The RAM utilization is not affected while performing a live migration, during which the instance still consumes 20.88%. The second base image, base2, is launched with 512MB of RAM. When booted, this instance consumes 10.71% of the allocated RAM. During the live migration, this instance consumed 10.79% of the allocated memory. The
last base image, base3, is launched with 1GB of RAM. It consumes 6.47% when booted and is also not affected at all during the live migration.

4.4.2.2 Performance of Web Server - Response Time

The second virtual machine to be examined is the virtual Apache web server. The web server was selected as a test case because of the high amount of network traffic associated with it. To test the performance of this virtual machine, a program, http_load is used. This interface allows for dynamic concurrent HTTP requests and a specified number of HTTP fetches. While examining the performance of these web instances, a set number of 50 concurrent HTTP requests was used to simulate access to the web server by 50 users. http_load is executed on a stand alone virtual machine to ensure that its process is not intrusive to the Web1, Web2, and Web3 virtual machines. It rates performance metrics in number of HTTP 200 responses per second. Each performance metric was executed 10 times, and the average was taken. The results for the Web1 virtual machine are given in Figure 4.10.
Figure 4.10: Web1 Response Time (Lower is better).

While fetching HTTP requests from Web1 when not in a migration, the response time increases linearly as the number of fetches increases. During the 100 HTTP request iteration, the average response time is 0.39 seconds. The average response time for 2000 HTTP fetches is 10.39. Given the data points as number of HTTP fetches given, the simple linear regression can be given as:

Web1 Non-Migration Stage \( y = 0.0052x - 0.1859 \)

This regression line is deterministic until the network bandwidth has been saturated or the number of concurrent HTTP requests has fully utilized the Web1 virtual machine. This formula is helpful in determining the performance of the Apache web server.

When the Web1 virtual machine is being migrated, the web server’s response time increases significantly. The major contributing factor to this decrease in performance is the virtual machines available RAM. Web1 is instantiated with 256MB of RAM. Handling 50 concurrent HTTP requests
while being migrated creates a need to use the virtual machines swap space, in turn decreasing its performance. Based on figure 4.13 the polynomic regression can be expressed as:

Web1 Migration Stage \((y = 8E - 12x^4 + 3E - 8x^3 + 4E - 5x^2 + 0.0061x + 1.4915)\)

![Figure 4.11: Web2 Response Time (Lower is better).](image)

The Web2 virtual machine is instantiated with 512MB of RAM. This gives an increase in performance by reducing the response time to concurrent HTTP requests while not in a migration. Similar to the Web1 virtual machine, the highest performance comes when the number of HTTP fetches equals 100, serving those requests in 0.39 seconds. This virtual machine has a maximum response time to 2000 HTTP requests of 7.9 seconds. The simple linear regression, comparing number of fetches to response time, is expressed as:

Web2 Non-Migration Stage \((y = 0.0039x - 0.0006)\)

While the Web2 virtual machine is being migrated, the web server has an HTTP request response time similar to the Web1 virtual machine. Again, as the number of concurrent HTTP fetch
requests is 50, the total number of fetches per second is bound by the total network bandwidth. The polynomic regression for the response time compared to HTTP requests for Web2 is represented as:

\[
    \text{Web2 Migration Stage} \quad (y = -8E - 6x^2 + 0.0315x - 3.933)
\]

![Graph showing response time](image)

Figure 4.12: Web3 Response Time (Lower is better).

The Web3 virtual machine is launched with 1GB of RAM and two virtual CPU cores. Deploying with this amount of resources was determined to have a positive effect on the performance of the web server. As shown in figure 4.12, the Web3 virtual machine has a response time to HTTP requests similar to the Web2 virtual machine. As expected, the performance shows a linear trend depending on the number of HTTP fetches requested. The simple linear regression which describes the performance of Web3 while not in a migration is expressed as:

\[
    \text{Web3 Non-Migration Stage} \quad (y = 0.0034x - 0.00089)
\]
The Web3 performance is interesting in that it performs worse than the Web1 and Web2 while responding to 2000 HTTP requests while in a migration. The average response time to this number of requests, with 50 concurrent connections, is 31.8 seconds. It is also worth noting that many of the tests using http_load requesting 2000 HTTP fetches caused the Web1 virtual machine live migration to fail. Due to the larger amount of resources, the Web3 virtual machine does have a better response time in all other cases. The performance linear regression of the Web3 virtual machine while in a migration is expressed as:

\[ y = 0.0167x - 1.8356 \]

The failed test cases shows the effects of stability of larger virtual machines while they are in a live migration. The HTTP requests saturate the network for a longer period of time, also causing RAM pages in the Web3 virtual machine to be continuously updated. This prolonged HTTP request also degrades the performance of the cloud resources during the live migration. In some cases, the cloud was unable to perform a live migration of the Web3 virtual machine. In those cases, the virtual machine is never stopped on the source physical server, and the user experiences no connection interruption.

Figure 4.13 give the performance comparison of Web1, Web2, and Web3 while they are stationary and while they are in a live migration process. While stationary, all of the virtual machines show a linear degrade of performance as the number of HTTP requests increases. The Web2 and Web3 virtual machines are identical in their performance, due to the availability of resources given when they were launched. The Web1 virtual machine performs slightly worse, being constrained by its small amount of available resources.

All of the virtual machines performed significantly worse while in a live migration com-
pared to being stationary. The Web1 and Web2 virtual machines have performance similar to each other. The Web3 virtual machine has a linear performance regression, and out performs Web1 and Web2 for all tests except 2000 HTTP requests.

4.4.2.3 Performance of MySQL Database Instance - Transactions Per Second

The third virtual machine to be examined is the virtual MySQL database server. The database server was selected as a test case because of the high amount of disk I/O associated with it. To test the performance of this virtual machine, a program, sysbench is used. Sysbench allows for modifications of the table size to be tested in the MySQL database, as well as the number of concurrent requests to that table (threads). In these experiments, the table size is 100,000, and the concurrent requests are modified by increasing the thread count. The results of these experiments are given below.
Figure 4.14: Performance of DB1 Database Virtual Machine.

The DB1 database is instantiated with 245MB of RAM, 2GB of local disk space and one CPU core. As shown in figure 4.14, the DB1 virtual machine shows a linear increase in performance as the number of concurrent requests to the MySQL database decreases. While stationary, the DB1 database server has the highest performance of 626.53 transactions per second serving only one concurrent connection. In the worst case while being stationary, DB1 has a performance of 462.19 transactions per second serving 128 concurrent requests. This is mostly due to the allocation of only one CPU core when DB1 was launched. On average the DB1 virtual machine has a performance of 566.22 transactions per second across all test cases while being stationary.

While the DB1 virtual machine is in a migration, it has a best case performance of 590.89 transactions per second, serving only one concurrent request thread. In the worst case while migrating, DB1 handles 436.6 transactions per second, serving 128 concurrent request threads. On
average the DB1 virtual machine has a performance of 519.31 transactions per second during a live migration across all test cases.

Figure 4.15: Performance of DB2 Database Virtual Machine.

Figure 4.15 gives the performance of the DB2 MySQL database virtual machine. This machine is launched with 512MB of RAM, 5GB of local disk space and one CPU core. DB2 shows a trend in performance that is similar to DB1, as the number of concurrent requests to the database increase, the number of transactions per second decreases. In the best case while the DB2 virtual machine is stable, it has a performance of 627.32 transactions per second. The worst case while stable is 452.02 transactions per second. The limiting factor here again is the number of CPU cores allocated when the virtual machine is instantiated. The average performance is 542 transactions per second for the DB2 database virtual machine while during a migration across all test cases.

When the DB2 virtual machine is in a migration, it has a best case performance of 582.38
database transactions per second, serving only one concurrent request thread. In the worst case while migrating, DB2 has a performance of 406.18 transactions per second, serving 128 concurrent request threads. During a live migration, the DB2 virtual machine has an average performance of 497.5 transactions per second across all test cases.

![Figure 4.16: Performance of DB3 Database Virtual Machine.](image)

Figure 4.16 gives the performance of the DB3 MySQL database virtual machine. This machine is launched with 1024MB of RAM, 10GB of local disk space and two CPU cores. The DB3 database server shows similar performance to the DB1 and DB2 virtual machines when the number of concurrent requests is one. Since this virtual machine is deployed with two CPU cores, it has a higher performance when handling increasing concurrent requests. In the best case while the DB3 virtual machine is stable, it has a performance of 885.79 transactions per second, using two concurrent request threads. The worst case while stable is 621.49 transactions per second,
serving only one request thread. The average performance for the DB3 database virtual machine is 731.37 transactions per second across all test cases while not in a migration phase.

When the DB3 virtual machine is in a migration, it has a best case performance of 754.29 database transactions per second, serving two concurrent request threads. In the worst case while migrating, DB3 has a performance of 572.67 transactions per second, serving only one concurrent request thread. While in a migration, the average performance for the DB3 virtual machine is 661.24 transactions per second across all test cases.

Figure 4.17: Performance Comparison of Database Virtual Machines.

A comparison of the virtual machines is given in figure 4.17. The DB1 and DB2 show similar performance trends while stationary and during a live migration. The performance of DB1 is slightly better than that of DB2.

The DB3 virtual machine has a higher performance than DB1 and DB2 after the initial test case of one concurrent request thread. This is due to DB3 having two CPU cores when launched.

106
After the test case of two concurrent MySQL request threads, the performance of DB3 is degraded, but still has higher performance than the DB1 and DB2 virtual machines.

4.4.2.4 Performance of Pi Computation Instance - Response Time

The last virtual machine to be examined is the Pi Computation instance. This type of virtual machine was selected as a test case because of the high CPU utilization of the algorithm that computes Pi with very high precision. To test the performance of this virtual machine, a jython program was created to compute Pi to a specific place. The range for this program was from the $10,000^{th}$ place to the $150,000^{th}$ place. To alleviate any stability issues in the performance of the host or guest virtual machine, each test was performed eight times sequentially, and the average was taken. The results of these experiments are given below.

![Figure 4.18: Performance of Pi Computation Virtual Machine.](image)

Figure 4.18 shows the performance of the Pi1 virtual machine computing Pi to the $n^{th}$ place while stationary and during a migration. As expected with a high CPU utilization experiment, the virtual machines performance is degraded as the $n^{th}$ place of Pi computed becomes larger.
While stationary, Pi1 takes just 0.24 seconds to compute the 10,000th place of Pi. To compute the 150,000th place of Pi, the virtual machine took 28.54 seconds.

When performing these experiments while migrating the Pi1 virtual machine, connection to the machine was lost each time the live migration finished. The Pi1 virtual machine would migrate to the destination server, but would become unresponsive to current SSH commands, and would not allow any further SSH login attempts. These errors could be the fault of low memory constraints on the virtual machine when it was launched.

The performance of Pi2 virtual machine is shown in figure 4.19. This virtual machine takes 0.16 seconds to compute Pi to the 10,000th place, a slight performance increase over Pi1. To compute Pi to the 150,000th place, Pi2 took 28.57 seconds. The performance of this virtual machine was degraded only slightly during a live migration. While in the migration, Pi2 took 0.17 seconds to compute the 10,000th place of Pi, and took 28.55 seconds to compute the 150,000th place of Pi.
place. Also it is worth mentioning here that there were no issues with the stability of the migration of this virtual machine, or the Pi3 virtual machine for the remainder of the experiments.

![Figure 4.20: Performance of Pi3 Pi Computation Virtual Machine.](image)

Figure 4.20 gives the performance of the Pi3 virtual machine. This virtual machine performs slightly better than the Pi1 and Pi2 virtual machines when computing the \( n^{TH} \) value of \( \pi \). When Pi3 is stationary, it took 0.12 seconds to compute \( \pi \) to the 10,000\(^{th} \) place. To compute \( \pi \) to the 150,000\(^{th} \) place, Pi3 took 26.86 seconds.

During a live migration the Pi3 virtual machine took 0.13 seconds to compute \( \pi \) to the 10,000\(^{th} \) place. It took 26.86 seconds to compute the 150,000\(^{th} \) place of \( \pi \) while migrating.

The Pi3 virtual machine was launched with two CPU cores, 1024MB of RAM and 10GB of persistent disk space. This gives the Pi3 virtual machine an average performance increase of 6.8% compared to Pi2 and 11.2% compared to Pi1 while stationary. During a migration, Pi3 has an average performance increase of 6% over the Pi2 virtual machine.
The figure 4.21 gives a comparison of the Pi virtual machines while computing Pi. Since this virtual machine tests the CPU utilization while stationary and during a migration, the performance is not greatly affected. The performance of the Pi virtual machines can be generalized as:

\[ y = 1E - 9x^2 - 7E - 6x + 0.1072 \]

4.4.3 Power Consumption of Performing a Live Migration

To evaluate the amount of power, \( E_w \), consumed by migrating the four types of virtual machines, the [struck add ref] method is used. This formula is defined as:

\[ E_w = E_{during} - E_{before} \]

A Watt-meter is used to determine the power, in Watts, the physical servers are using while hosting the virtual machines. These values are then compared to the amount of power consumed.
when a live migration is being performed. The servers used in these experiments were Dell Opti-plex 755 machines running Ubuntu 12.10 Server. During the live migration, the Watt-mater is connected to server A. Since all of the compute nodes are identical Dell machines, it can be assumed that the power consumed is identical when any of these machines are performing the live migration. The migrations are performed 10 times, taking the average power consumption in Watts as the result.

4.4.3.1 Power Consumption of Migrating the Base Instance

When performing the base virtual machine live migration power consumption, each of the three base virtual machines is examined. The base1 and base2 virtual machines are similar except for the amount of RAM and Disk space allocated when launched. The base3 virtual machine consumes the highest amount of resources and has two virtual CPU cores. Figure 4.22 gives the comparison of the amount of power consumed for the base virtual machines.

![Figure 4.22: Base Performance Comparison.](image)

Figure 4.22: Base Performance Comparison.
As shown in figure 4.22 the base1 and base2 virtual machines consume similar amounts of power while stable, 55 and 55.1 Watts respectively. Migrating the base3 instance consumes slightly more power while stable at 55.4 Watts.

The difference in the power consumption of migrating the base virtual machines is negligible as well. Migrating the base1 and base2 virtual machines requires 58.4 Watts while migrating the base3 virtual machine requires 59.1 Watts.

Comparing the power consumption of hosting the base virtual machines when stationary to when they are migrating gives the costs of performing a live migration. The cost of migrating the base1 and base2 virtual machines are similar, a 6% increase in power consumption during the migration. The cost of migrating the base3 image is a 6.3% increase in power consumption during the migration.

4.4.3.2 Power Consumption of Migrating the Web Server Instance

This section examines the power consumption of migrating the three web server virtual machines. To obtain a base power consumption for each virtual machine, a Watt-meter is used. The results are shown in figure 4.23.

The base power consumption to host the Web1 and Web2 virtual machines is similar at 58.2 Watts. The Web3 virtual machine consumes slightly more power while stationary at 59.1 Watts. These power consumption values were taken while hosting each of the web server virtual machines independently while also not having any external HTTP requests.

Figure 4.24 shows the amount of power consumed by hosting the Web1 virtual machine while stationary and during a migration. These power consumption values, and the rest of the power consumption values in this section, were taken while the web server virtual machine was
stationary and during a migration. The web server virtual machines were also loaded with an increasing number of concurrent HTTP requests.

The Web1 virtual machine was able to respond to all of the increasing number of HTTP requests while stationary. The power consumption of this virtual machine is between 75.5 Watts and 76 Watts in all cases while stationary. When the Web1 web server is in a migration, it responded to the 10, 25, and 50 concurrent HTTP request experiments, and failed to migrate on all other experiments. Its power consumption while responding to these requests was around 77.15 Watts. This gives a cost of migration, as a percent increase in power consumption, of 2% to migrate the Web1 virtual machine.

Figure 4.25 gives the power consumed for hosting the Web2 virtual machine while stationary and during a live migration. This web server was able to respond to all HTTP request experiments while being stationary and during a migration. While stationary, the Web2 virtual ma-
Figure 4.24: Power Consumed While Migrating Web1.

The Web2 virtual machine consumed around 77.6 Watts for the first three stationary HTTP request experiments. During the remaining experiments, the Web2 virtual machine consumed slightly above 75.5 Watts while stationary. While in a migration, Web2 consumed 74.5 Watts, 76.5 Watts, and 77.7 Watts for 10, 25, and 50 concurrent HTTP requests, respectively. The power consumption was lower for the first three experiments, which could be due to stability issues with the cloud host operating system and network. While stationary, Web2 levels out around 75.6 Watts for the remainder of the concurrent HTTP experiments. While in a migration, Web2 fluctuates slightly between 76.5 Watts and 78 Watts for the remaining concurrent HTTP experiments. Examining figure 4.25, the cost of migrating Web2 is less than a 0.5% increase in power consumption.

Figure 4.26 gives the power consumed for hosting the Web3 virtual machine while sta-
Figure 4.25: Power Consumed While Migrating Web2.

This web server was about to respond to all HTTP request experiments while being stationary and during a migration. While in a migration, the Web3 virtual machine increases in power consumption from 82.1 Watts to 84.3 Watts for the first four HTTP experiments. When requesting 250 concurrent HTTP requests, the virtual machine consumes 83.2 Watts. The remainder of the experiments have a power consumption of around 83.4 Watts during a migration. While stationary, Web3 increases similarly in power consumption as it does during a migration for the first three HTTP experiments. The power consumption of Web3 is then higher while stationary than during a migration for 100 and 250 concurrent HTTP requests. This could be due to stability issues with the host operating system and cloud network.

Figure 4.27 shows the comparisons between the virtual web servers in the amount of power required to host them while responding to HTTP requests. As expected, the Web3 virtual machine requires the most power while stationary and while in a migration. This virtual machine requires
the most resources when booted, and in turn has the highest virtual machine performance, in terms of HTTP requests, but has a trade-off of higher power consumption. The Web1 and Web2 stationary power requirements are similar after 100 concurrent HTTP requests. The power consumed by Web1 is lower while handling HTTP requests while in a migration as compared to Web2, but Web1 was only able to respond to the first three HTTP experiments. This is due to Web1 having a lower amount of resources than Web2 when instantiated.

4.4.3.3 Power Consumption of Migrating the MySQL Database Instance

This section gives the power consumption of the cloud resources when performing a live migration of the database virtual machines. The results are shown in the figures below.

Figure 4.28 shows the amount of power consumed while hosting the DB1 virtual machine.
While stationary, the DB1 virtual machine consumes 74.4 Watts, while responding to one concurrent database request. In the worst case, the DB1 virtual machine consumes 76 Watts when responding to 128 concurrent database requests.

While in a migration, the cloud resources consume 74 Watts in the best case while moving the DB1 virtual machine. In the worst case, the DB1 virtual machine consumes 76.9 Watts while accommodating 128 concurrent database requests.

Figure 4.29 gives the amount of power consumed while hosting the DB2 virtual machine. While stationary, the DB2 virtual machine consumes 74.2 Watts, while responding to one concurrent database request. In the worst case, the DB2 virtual machine consumes 76 Watts when responding to 128 concurrent database requests.

While in a migration, the cloud resources consume 75.4 Watts in the best case while moving
Figure 4.28: Power Consumed while Migrating DB1 Virtual Database Server.

Figure 4.29: Power Consumed while Migrating DB2 Virtual Database Server.
the DB2 virtual machine. In the worst case, the DB2 virtual machine consumes 76.5 Watts while accommodating 128 concurrent database requests.

Figure 4.30: Power Consumed while Migrating DB3 Virtual Database Server.

Figure 4.30 shows the amount of power consumed while hosting the DB3 virtual machine. While stationary, the DB3 virtual machine consumes 74.5 Watts, while responding to one concurrent database request. In the worst case, the DB3 virtual machine consumes 81.6 Watts when responding to 128 concurrent database requests. The performance evaluation of DB3 using only one thread is similar to DB1 and DB2 since the test cases do not allocate the virtual machine’s second CPU core.

While in a migration, the cloud resources consume 74.5 Watts in the best case while moving the DB3 virtual machine. In the worst case, the DB3 virtual machine consumes 81.3 Watts while accommodating 128 concurrent database requests. One reason the DB3 virtual machine requires more power than DB1 and DB2 is the amount of resources allocated to it when launched. This
creates a trade-off of performance of the virtual machine and the power consumed by the physical cloud resources.

![Figure 4.31: Comparison of Power Consumed while Migrating Virtual Database Servers.](image)

A comparison of the amount of power needed to host the database virtual machines is given in figure 4.31. The DB1 virtual machine is unstable until the number of concurrent requests reaches 32. After this number of requests, DB1 has a linear increase in power consumption based on the number of concurrent database requests. The DB2 virtual machine has a similar trend compared to DB1 after 32 concurrent database requests. The DB3 virtual machine requires much more power while stationary and while in a migration. This is a trade-off of this virtual machine due to the amount of resources allocated to it when booted.
4.4.3.4  Power Consumption of Migrating the Pi Computation Instance

This section evaluates the power consumed by hosting and migrating the Pi computation virtual machines. Since the stability of the host cloud server fluctuates in power consumption during these tests, the experiments were run eight times. These power fluctuations are due to the jvm in which jython operates being context switched on and off of the CPU. Performance issues with the Pi3 virtual machine was better due to allocating it two CPU cores when booted. The results are given in the figures below.

![Power Consumption Graph](image)

**Figure 4.32: Power Consumed while Migrating Pi1 Pi Computation Server.**

The power consumed by hosting the Pi1 virtual machine is given in figure 4.32. While computing Pi to the $10,000^{th}$ place, the Pi1 virtual machine consumes 70.1 Watts. The most power consumed, 74.9 Watts, is while Pi1 is computing Pi to the $75,000^{th}$ place. For values after 50,000, the Pi1 virtual machine consumes roughly the same amount of power. Fluctuations in power consumption can be due to stability issues with the virtual machine host server. On average,
the Pi1 cluster virtual machine consumes 73.98 Watts over all test cases while not in a migration phase.

During the migration phase, the Pi1 cluster virtual machine becomes unresponsive after it relocates to server B. The connection is lost for the user, and is unrecoverable. Since this is the case, the Pi1 virtual machine is considered as failed after performing a live migration. This is the reason there is no power consumption plot given in 4.32.

![Figure 4.33: Power Consumed while Migrating Pi2 Pi Computation Server.](image)

Figure 4.33 shows the power consumption of the Pi2 virtual machine while stationary and during a live migration. The power consumption while stationary is relatively low when computing the 10,000\(^{th}\) and 25,000\(^{th}\) place at 72.8 Watts. The power consumption increases to just over 74 Watts for the rest of the Pi tests. On average, the Pi2 cluster virtual machine consumes 74.34 Watts over all test cases while stationary.

When the Pi2 virtual machine is migrating, its power consumption increases significantly.
for computing the $10,000^{th}$ and $25,000^{th}$ place to over 74 Watts. In the worst case while migrating, Pi2 consumes 75.4 Watts while computing the $75,000^{th}$ place of Pi. During the live migration phase, the Pi2 cluster virtual machine consumes on average 74.98 Watts over all test cases.

Figure 4.34: Power Consumed while Migrating Pi3 Pi Computation Server.

The power consumed by hosting the Pi3 virtual machine is given in figure 4.34. Pi3, the largest virtual machine hosted by the cloud resources, also consumes, on average, the most power. While stationary, the lowest amount of power consumed was while computing Pi to the $10,000^{th}$ place at 74 Watts. The most power consumed while stationary was computing Pi to the $100,000^{th}$ place at 75.5 Watts. The Pi3 cluster virtual machine consumes 74.92 Watts on average across all test cases while not in a live migration phase.

When the Pi3 virtual machine was in a live migration, the lowest amount of power consumed was while computing Pi to the $100,000^{th}$ place at 76.1 Watts. The virtual machine con-
sumed the most power while computing Pi to the 10,000th place at 77.3 Watts. On average, the Pi3 cluster virtual machine consumes 76.8 Watts over all test cases while in a live migration phase.

![Figure 4.35: Comparing Power Consumed while Migrating the Pi Computation Servers.](image)

Figure 4.35 gives a comparison of the amount of power consumed while hosting the Pi virtual machines. As the chart shows, all of the virtual machines consume roughly the same amount of power, except the migrating Pi3 virtual machine, for all tests after finding the 75,000th value of Pi. The Pi3 virtual machine, launched with the most amount of physical resources creates a trade-off of power and performance when computing the values of Pi.
Chapter 5

POWER EFFICIENT PERSISTENT CLOUD STORAGE CONSOLIDATION

5.1 Introduction

Organizations have several different implementations of open source cloud software to choose from when deploying their own local cloud. Common open source cloud solutions include OpenNebula [OpenNebula, 2011b], Nimbus [Nimbus, 2012], CloudStack [Apache, 2013], and OpenStack [OpenStack, 2012]. The most commonly used open source local cloud stack is Eucalyptus [Eucalyptus, 2011b] provided by Eucalyptus Systems, Inc. The cloud Infrastructure-as-a-Service (IaaS) architecture described in this chapter is developed using the Eucalyptus cloud stack which gives users virtual resources to undefined job types. The Eucalyptus IaaS architecture is composed of a front-end administrative cloud controller, a cloud cluster controller for administering virtual compute resources, a virtual machine image repository, and many persistent storage nodes and compute nodes. The Eucalyptus cloud stack is designed for ease of scalability and availability of resources, but does not address the problem of the amount of power a typical architecture like this consumes.

The dramatic increase in demand for access to computing resources coincides with an increased awareness of the potential impact on the environment caused by large scale computing. Not only is green computing needed to lessen the impact of large scale computing on the environment, but green computing can also decrease the monetary cost of large scale computing. A large amount of electricity is needed to operate and cool the data centers. For example, a data center with
1000 racks and about 25k square feet would require 10,000,000 watts of energy for its computing infrastructure. Removing the dissipated heat would require an additional 5,000,000 watts [Patel, Sharma, Bash, and Graupner, 2002]. There are approximately 6,000 data centers in the United States, which according to the U.S. Government report in [Kurp, 2008] consumed 1.5% of all electricity produced in 2006 for a total of $4.5 billion. There has been a dramatic growth in overall power consumption by data centers since the year 2000, with the demand for power increasing every year. Storage devices have the fastest growing power consumption rate at 191%.

Research is being done in areas, such as efficiency increases in network data transmissions, disk access to improve energy efficiency and data center location. Some organizations have already adopted the approach of utilizing more energy efficient hardware [Barroso and Holzle, 2007]. Approaches to minimizing disk storage and/or data transmission is being used to lower the costs of data center operation, such as more efficient means of data replication and file replacement strategies [Vrbsky, Lei, Smith, and Byrd, 2010]. A few organizations are relocating their data centers to take advantage of more opportune environments. For example, Microsoft has built data centers in cooler climates and one of their data centers in Washington takes advantage of hydroelectric power to lower its carbon footprint [Kurp, 2008].

The goal of this research is to give small to medium-sized businesses the opportunity to deploy efficient cloud architectures similar to what is offered by large vendors. Cloud computing resources generally take the form of processing and storage resources. This chapter focuses on decreasing the amount of energy required to store persistent data in these cloud environments. Relying on the idea that users only access their persistent storage accounts after requesting computing resources, not all persistent data has to be available at any given time. Instead, the efficient
cloud architecture proposed in this dissertation reduces the power consumption of storage nodes by moving inaccessible data to “cold” storage servers.

There is a need for storage nodes in the proposed efficient cloud architecture since the virtual machines requested by users are inherently stateless. As users request computational resources, their persistent storage will become available as well. This persistent storage is mounted as accessible network storage and is only available to that user while they are actively consuming compute node resources. Users are determined to be inactive once they have released all of their compute node resources (virtual machines).

While maintaining a typical design for local cloud architectures, in the proposed efficient cloud architecture each cluster comprises only three different types of nodes. These are compute nodes, storage nodes, and a cluster controller that hosts load balancing and networking processes. The cluster controller is an administrative component and must always remain available. This is not true for the compute nodes and storage nodes. Organizations could reduce the energy consumed by storage nodes by applying a consolidation scheme that only gives accommodations for accessible data. As described in Chapter 3, the strategy to reduce wasted power in a local cloud architecture is to turn off compute nodes when they are not being used. The default load balancing algorithm, round robin, does not account for power usage when compute nodes are idle. Therefore, there is room for reducing power consumption without dramatically affecting cloud resource availability. Compute nodes will default to a powered down mode, (turned completely off), costing no energy while no computational resources are being requested. Storage nodes in this architecture will also default to a powered down mode, denoted as a “cold node”. As the number of active users increase, cold nodes can be converted to hot nodes, which are powered on and operate like typical network storage devices.
There is a need for storage nodes in the proposed cloud architecture since the virtual machines requested by users are inherently stateless. As users request computational resources, their persistent storage will become available as well. This persistent storage is mounted as accessible network storage and is only available to that user while they are actively consuming compute node resources.

5.2 Problem Statement

Storage devices consume the second most amount of energy in a typical data center, following computational devices [Harnik et al., 2009]. In larger data centers, storage devices can consume up to 40% of the total amount of power needed for operation [EPASTAR, 2007], [Sculz, 2012]. Since computational resources has been given a lot of attention in reducing the amount of power consumed to perform jobs, storage devices may become the dominant consumer of power in the near future.

Persistent storage is a necessary component in a local cloud environment. In the proposed design, the computation resources requested by users do not maintain any information once they are no longer needed. The virtual machines are inherently stateless processes that are maintained by the compute nodes in the local cloud.

While a cloud storage solution is needed, it is not necessary to have all persistent data available at any given time. The amount of persistent data storage that is available depends on the number of users that are logged into the local cloud system at any given time. Persistent storage nodes are located in each auxiliary cluster that makes up the entire cloud network. Since the limiting factor in determining the number of active users relies on the number of compute nodes in a cluster, storage nodes can be transitioned into a “cold” state if the data they hold belongs to an inactive cloud user.
Generally, storage nodes are heterogeneous devices with lower computational capabilities than compute nodes. Emphasis is given to hard disk and network speed on these devices. Since these nodes can accommodate a huge amount of persistent storage, the energy consumption of these nodes is relatively high. In this case, a power aware storage consolidation algorithm can be introduced to decrease the amount of electricity consumed by storage nodes.

While accommodating for overall low power consumption, a full coverage solution of the persistent data is still mandatory. Generally, the proposed architecture finds the minimal subset of persistent storage nodes needed to satisfy data availability to active cloud compute resources. The architecture satisfies the full coverage problem if all of the data items are available to cloud users.

For designs that accommodate electricity conservation, there will be trade-offs. This proposed design hosts inactive users’ data on cold storage nodes which default to a powered off state. This data is transferred to a hot storage node when the user becomes active. This fulfills the need of persistent data storage in the cloud architecture, but introduces a latency of when the user can access their data.

5.3 Cloud Storage Architecture

Storage nodes in the local cloud architecture have emphasis placed on the ability to transfer and store large amounts of data. Using a power aware approach to data storage, these nodes will transfer active user’s data from a cold server to hot servers once these users request at least one virtual machine. Depending on the amount of persistent data each user has determines how long it will take for this data to become available. Once the user releases their computational resources, their data is deemed inaccessible, and is moved back to a cold server. Cold servers are powered off to decrease power consumption. There should be no difference in the design of the hot or cold
storage node. In the case a hot node cannot accommodate all of the persistent storage requests, cold nodes can be converted to hot nodes.

5.3.1 Network Limitations

In the proposed local cloud architecture, the persistent storage nodes should be allocated to a single switch. This allows data transfer between the storage nodes with no interference of other network traffic in the local cloud. This is imperative, since transfer of data between storage nodes is network intensive. This also prevents persistent data transfer from interfering with networking interfaces to and from compute node resources.

Assuming the local cloud architecture is using a 1 Gbps Ethernet LAN, the maximum throughput for this network is 125 MB/s (1,000,000,000/8). Since a switched architecture is used, each port is capable of this speed simultaneously. The switched Ethernet architecture is sufficient for a small to medium-sized cloud architecture due to its ability to auto adjust communication speeds to accommodate different hardware. Other features include device isolation and higher bandwidth between cloud resources.

Figure 5.1 illustrates the networking topology of persistent storage nodes, and how they communicate with the other components of the local cloud architecture. The administrative nodes: cloud controller, cluster controller, and walrus/image controller are confined to a single switch that allows access to the cloud’s web interface to not interfere with compute and storage node communication. The compute nodes are confined to switches depending on their capabilities. Compute nodes are grouped together based on their amount of resources available to alleviate any performance bottlenecks. The storage nodes are also confined to a single switch, and that switch is confined to the switch hosting the compute nodes. This design decision is made to increase the performance of communication between compute and storage nodes in the local cloud architecture.
5.3.2 Hot Storage Node

Storage nodes that are actively hosting persistent data to active users are considered hot storage nodes. Data is transferred to these nodes when new users become active. These nodes remain powered on as long as the number of active users remain high. As the number of active users decreases, hot storage nodes consolidate the active user’s data to fewer nodes to allow powering
down of unneeded nodes. Once a user releases computational resources, their persistent data is transferred to a cold server.

Since the described cloud implementation is of an IaaS architecture, the types of data stored in the cloud cannot be predetermined. Data can be of any type, depending on what the user is doing with the cloud computational resources. The data can range from a few values stored on a page to a very large database.

5.3.3 Cold Storage Node

When users release computational resources in the local cloud, their persistent storage is moved from the hot storage nodes to a cold storage node. These storage nodes stay powered off until a request is made for data stored on them. When a user requests computational resources from the local cloud, the cluster controller sends a wake on LAN packet to the cold server hosting that user’s persistent data. The data is then transferred from the cold server to a hot server. Once the data transfer is complete, the cold server powers off. The default state for persistent storage nodes in the cloud architecture is to be in a “cold” state.

5.4 MySQL Database Design

A MySQL database is used to record information about the persistent storage in the local cloud. This database will reside on the cluster controller. Each time computational resources are requested or released, this database will be updated to handle persistent storage. This database encompasses two tables. The User table is used to determine which user is active, and where their data is located. The Nodes table keeps track of all of the different types of storage nodes in the local cloud. The Users table has six fields:

1. UserID
2. StorageLocation
3. ServerLocation
4. StorageSize
5. BackupLocation
6. Active

The Nodes table has two fields:

1. NodeID
2. Type

5.4.1 UserID

The UserID is copied from the Eucalyptus cloud controller. This field is also used to name the directory of the persistent storage. The UserID of a specific user will never change as long as they are active in the local cloud architecture.

5.4.2 StorageLocation

This field determines where the persistent storage data is located. If the owner of the directory is not active, this data is located on a cold storage node. Once the user becomes active, the record is examined to locate their persistent storage on a cold server and move it to a hot storage node.

5.4.3 ServerLocation

Since the local cloud architecture can accommodate multiple storage nodes, a physical server location field is needed. Once a user becomes active, the persistent storage is located on a physical cold storage node and moved to a hot storage node. When the user releases all compute node resources, this field is updated and their data is moved back to a cold storage node.
5.4.4 StorageSize

This field is needed to make sure the persistent data will fit on the hard drive of the storage node to which it will be transferred. This field is updated once the user requests to terminate their last active virtual machine. This ensures their persistent data will not change in size before being moved.

5.4.5 BackupLocation

Since data corruption and machine failure is inevitable, a persistent storage backup solution is needed in the local cloud architecture. This field is updated when a backup process replicates a user’s data from a cold storage node to a dedicated backup node. When data corruption or physical machine failure occurs, data can be transferred from the backup node to a functioning hot storage node.

5.4.6 Active

This field is used to determine if a user is actively consuming computational resources. If this is the case, then the persistent data is located on a hot storage node. If this field has a false value, then the persistent data is located on a cold storage node. This field is updated when a user requests computational resources and when they release their last virtual machine.

5.4.7 NodeID

Each persistent storage node has a unique identifier. This field is used to distinguish physical nodes from each other. The NodeID will never change as long as the storage node is active in the local cloud architecture.
5.4.8 Type

This field is used to determine the type of storage controller. There are three different types: hot storage node, cold storage node, and backup node. Hot and cold storage nodes can change types during their operation in the local cloud architecture depending on the number of active users. Backup nodes shall never change in type and are used only for data replication from cold storage nodes.

5.5 Power Aware Storage Consolidation Algorithm

This algorithm is intended for organizations wanting to implement small to medium-sized local clouds. It should scale to larger-sized clouds since the data transfer among storage nodes is encapsulated within the switch they are behind. This algorithm will be limited in scalability with respect to accessibility. As the number of active users increases, the amount of network bandwidth needed will also increase. Table 5.1 explains the variables used in the PASC algorithm.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$data_j$</td>
<td>User $j$’s current data set</td>
</tr>
<tr>
<td>$HS_a$</td>
<td>$a^{th}$ Hot storage node</td>
</tr>
<tr>
<td>$Quota_j$</td>
<td>User $j$’s storage quota</td>
</tr>
<tr>
<td>$CS_a$</td>
<td>$a^{th}$ Cold storage node</td>
</tr>
<tr>
<td>$m$</td>
<td>Total set of available storage nodes</td>
</tr>
<tr>
<td>$CS_a$ utilization</td>
<td>Current utilization of $a^{th}$ Cold storage node</td>
</tr>
<tr>
<td>minUtil</td>
<td>Most under utilized cold storage node in set $m$</td>
</tr>
</tbody>
</table>

All of the computation included in this algorithm is maintained in the cluster controller. The cluster controller maintains the utilization state of each active compute node and makes decisions on where to instantiate new virtual machines. The cluster controller also maintains the networking state of all of the computational nodes. Each time a user becomes active or leaves the local cloud,
the MySQL database is accessed in order to give them access to their persistent data or to delete the data. The MySQL database is used to maintain the location of the persistent data in the cloud, which users are currently logged in, what type of backup need to be performed, and which storage nodes are hot and cold.

The PASC algorithm has five basic sections. The Active section is responsible for locating a user’s data on a cold storage node and moving it to a hot storage node. It evaluates the utilization of all of the hot storage nodes to determine which node will host the user’s data. A hot storage node must be able to host the user’s current persistent data plus up to the maximum allowed persistent data for that user. Utilization for hot nodes is calculated as the max quota allowed for all users’ data that it is currently storing compared to its maximum storage capacity. Once a hot storage node has been found, the users’ data will be transferred from a cold storage node to the hot storage node. The data will be deleted from the cold storage node. The data is no longer needed on the cold storage node since the user is actively accessing it from a hot node, and it has been replicated to a backup node. The cold node that was accessed is then powered down.

In the case there are no hot nodes able to accommodate the users’ data, a cold-hot conversion is performed on one of the cold storage nodes.

The In-Active section handles the users’ data when they release their last virtual machine resource from the compute nodes. When the user becomes inactive, the utilization of all of the cold storage nodes is examined to determine which cold node will host the users’ data. Once a cold storage node has been determined, that node is powered on and the users’ data is transferred from the hot storage node to the cold storage node. The users’ data is then deleted from the hot storage node to make room for future active users’ data. Once the data has been written to the cold storage node, the node then powers down.
Algorithm PASC

Active:
if user$_i$ becomes active:
    locate data$_j$ on cold storage node
    HS$_a$ ← current utilization of hot storage node$_a$
    if HS$_a$ can accommodate data$_j$ + (Quota$_j$ − data$_j$):
        power on cold storage node
        transfer data$_j$ → hot storage node$_a$
        delete data$_j$ from cold storage node
        power off cold storage node
    else: //no hot storage node can accommodate data$_j$
        perform cold-hot conversion

In-Active:
if user$_i$ becomes in-active:
    CS$_a$ ← current utilization of cold storage node$_a$
    if CS$_a$ can accommodate data$_j$:
        power on cold storage node$_a$
        transfer data$_j$ → cold storage node$_a$
        delete data$_j$ from hot storage node
        power off cold storage node$_a$

Cold-Hot Conversion:
    CS$_a$ ← current utilization of cold storage node$_a$
    for CS$_a$ ∈ [m] do:
        minUtil ← (if CS$_a$ utilization < minUtil)
        update CS$_a$ with minUtil to hot storage node
        perform In-active on CS$_a$ to transfer unused data to cold storage nodes

Hot-Cold Conversion:
    for CS$_a$ ∈ [m] do:
        if ((CS$_a$ + CS$_a$ utilization) < 75%):
            transfer data CS$_a$ → CS$_a$+1
            delete data from CS$_a$
            update CS$_a$ to cold storage node

Backup:
    if day < 7: incremental backup
    else: full backup

Figure 5.2: PASC Algorithm.
The Cold-Hot Conversion is used to change the state of a cold storage node to a hot storage node in the event that none of the current hot nodes can accommodate a user’s data. The algorithm examines the utilization of each cold storage node as the amount of data stored on its hard drive(s) compared to its total amount of storage. The cold storage node with the lowest utilization will then be converted to a hot storage server. Once the node is converted, the In-Active section of PASC is executed on this node to transfer its data to other cold servers.

A Hot-Cold Conversion is used to change the state of a hot storage node to a cold storage node. This is performed to decrease the amount of power consumed compared to the number of active users in the local cloud. Each hot storage node is evaluated for utilization. If the sum of any two hot storage nodes is less than \( \text{minUtil} \), then the data will be combined to the hot node with the highest utilization percentage. Transferring data to the higher utilized node reduces network traffic and while reaching the same conclusion of converting one hot storage node to a cold storage node and giving it the ability to power down. This produces a cold storage node with a zero percent utilization, since all of the data was transferred to another hot node, then deleted from the converted cold node.

The backup section is included to make sure persistent data on the storage nodes is replicated for fault tolerance. Nodes that are designated as backups are never converted to hot or cold storage nodes. These nodes are powered down until a low traffic state in the cloud is reached. Each night starting at 2am the backup nodes are powered on, along with all of the cold storage nodes with utilization percentages of greater than zero. An incremental backup of persistent data is performed. Once per week, on the weekend, a full backup of the persistent data is performed.
5.6 Data Replication

Cloud users should keep in mind that all computers and their components will eventually fail. There is no way to predict when this will happen. The most common loss of data comes from corruption or physical machine failure.

In the local cloud architecture, there should be dedicated backup persistent storage nodes. These storage nodes are no different than the other hot and cold storage nodes. They should never be used as hot storage nodes. The backup pattern is based on executing an incremental backup of each cold server every night. In this case, only the changes made during the day are involved in the backup process. Files are transferred from cold storage nodes to the backup nodes. Once per week, a full backup is executed where the entire contents of the cold storage nodes is transferred to the backup nodes.

In the case of a storage node failure, data can be transferred from the backup node to another correctly functioning storage node. Since the backup nodes are defaulted to an off state, the user will incur latency while the backup node powers on. Since the data is not replicated to any other location in the cloud, the backup node is the only option for data recovery when a fault occurs. The backup servers also operate using Ubuntu Linux operating systems, which take 20 to 30 seconds to be operational after powering on. Once the node boots, the users’ data is transferred from the backup node to the hot storage node. After this occurs, the user has the ability to manipulate the data until they log out of the system. Once the user logs out, the data is transferred to a cold storage node, and will again be replicated to a backup node.

In this architecture, there are at most two copies of all users’ data. Additional replication can be added at the cost of power consumption and physical hard disk space. Since this architecture
is designed for a small to medium sized organization, backup procedures are probably already in place for other persistent data services. Tape backup of backup nodes can be performed for an additional layer of availability and redundancy if needed. This is also good practice, since tapes can be rotated off-site, in the case of a disaster.

5.7 Discussion of Results

In this section, we evaluate the results of the PASC algorithm along with other modifications to increase data throughput while attempting to decrease power consumption and latency. Since this experiment is implemented on an IaaS cloud, the power consumption model is determined by the number of users that connect to the cloud at any given time. Based on two different job schedules, the power consumed by the PASC algorithm is compared to an always on approach to storing persistent data. Since there will be a trade off of latency in some cases, the time until persistent data is available is evaluated as well.

5.7.1 Job Schedule

There are two schedules used in the experiments to examine PASC. The first job schedule used in our experiments uses similar curves with differences in the number of virtual machines activated over a 24-hour period. This is used to ensure different numbers of storage nodes are tested in each experiment run. Each job schedule increases the number of users by 30, allowing use for a greater number of hot storage nodes. The job schedule represents the typical usage of cloud resources, increasing in the morning until peaking around noon and declining as the day approaches 5pm. The resources increase slightly again in the evening between 6pm and 9pm. The second job schedule uses a random number of virtual machine implementations over the 24 hour period.

The job types for these experiments were divided into three categories. These types were
virtual cluster, database server, and web server. Apache Hadoop was implemented on our IaaS cloud for creating a virtual cluster. As this architecture was created to utilize all of the cloud’s resources, it consumed the most power as well. The second job type scheduled was cloud database servers. The persistent storage was mounted to instances hosting the database management systems. Lastly, webservers were utilized as job types in our IaaS cloud. The virtual cluster vm was chosen because of its high demand on computational resources. The web server, Apache, was chosen due to its demand on networking resources. The Database virtual machine has a high amount of disk I/O. These three types were chosen for their distinct demands on the local cloud computing architecture.

There are a total of four storage nodes either hot or cold, and one backup node. Depending on the number of active users, storage nodes can be converted from hot to cold, or from cold to hot. A cold to hot conversion is performed each time the number of users exceeds 30, 60, and 90.

5.7.2 Storage Node Power Consumption

For the experiment, we used a watt-meter to gather information on the power consumed by the storage nodes. There were two different job schedules used, the typical small local datacenter job schedule in figure 5.3, and a random job schedule. During both schedules, power consumption was recorded at regular intervals.

On average, while the nodes were active and transmitting data used by virtual machines, the storage nodes consumed 68.2 Watts. During each experiment run, we logged the power consumed. This data is presented in figure 5.4.

Based on the first job schedule, accommodating 10 VMs only required the use of one storage controller. The storage controller consumed 556Wh during the run as compared to the 6.5kWh consumed by having all of the storage nodes on all of the time. When requesting 20 VMs,
Figure 5.3: Job Schedule of VM Requests.

Figure 5.4: Comparing Power Consumed PASC vs Always On (Typical Schedule).
PASC again required the use of one storage controller and consumed 852.5Wh. The maximum number of VM requests for one storage node is 30. With this number of users, PASC consumed 988.9Wh over the 24 hour period. The next experiment requested 60 virtual machines. This number saturates two storage controllers and consumes 1.52kWh. Lastly, 120 virtual machines were requested, using four storage controllers. With this number of users PASC consumed 2.3kWh based on the given job schedule.

Figure 5.5: PASC Power Consumption vs Always on (Typical Schedule).

Figure 5.5 shows the percentage of power consumed by the PASC approach compared to an always on approach to persistent storage. These results are given using the typical job schedule while requesting an increasing number of virtual machines of a given type. Comparing the web servers, the persistent storage nodes saved 95% to 79% of the power with the PASC approach compared to the always on storage node approach. When comparing the cluster instances, the
power savings was between 92% and 65%. The database virtual machine saves between 93% and 72%.

Table 5.2 gives a summary comparison of the amount of power the cloud architecture uses comparing PASC to always on storage nodes. In the best case, the web server virtual machine type consumes only 5.1% of the power for storing persistent data. In the worst case, the cluster virtual machine consumes 35.2% of the power for persistent storage compared to leaving the storage nodes in an always on state.

Table 5.2: PASC Power Consumption Percentage vs Always on Storage Controllers (Typical Schedule).

<table>
<thead>
<tr>
<th>Number of VMs</th>
<th>Web</th>
<th>Cluster</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5.1</td>
<td>8.5</td>
<td>6.9</td>
</tr>
<tr>
<td>20</td>
<td>7.8</td>
<td>13</td>
<td>10.5</td>
</tr>
<tr>
<td>30</td>
<td>9</td>
<td>15.1</td>
<td>12.4</td>
</tr>
<tr>
<td>60</td>
<td>13.9</td>
<td>23.2</td>
<td>19.3</td>
</tr>
<tr>
<td>120</td>
<td>21.1</td>
<td>35.2</td>
<td>31.5</td>
</tr>
</tbody>
</table>

The next experiments were performed with a random job schedule. This schedule was run for a 24 hour period, instantiating a random number of virtual machines during each time period. The experiments each used a maximum number of virtual machines similar to the experiments using the typical job schedule. Figure 5.6 compares the power consumed by PASC to an always on storage node using a random virtual machine instantiation schedule. As the chart shows, the amount of power consumed by the cluster virtual machine storage nodes, and all other virtual machine types, is constant until 60 virtual machines are requested. When 60 cluster virtual machines are requested, the storage nodes consume 2.33 kWh. At 120 requested virtual machines, the storage nodes consume 3.98 kWh.

The web server virtual machine storage nodes consume 1.39 kWh when 60 virtual ma-
chines are requested. The maximum power consumed is 2.39 kWh when 120 web server virtual machines are requested.

The database virtual machine storage nodes consume 1.26 kWh when 60 virtual machines are requested. The maximum power consumed is 3.38 kWh when 120 database virtual machines are requested.

![Figure 5.6: Comparing Power Consumed PASC vs Always On (Random Schedule).](image)

Figure 5.6: Comparing Power Consumed PASC vs Always On (Random Schedule).

Figure 5.7 shows the percentage of power consumed by the PASC approach compared to an always on approach to persistent storage. These results are given using the random job schedule while requesting an increasing number of virtual machines of a given type. Comparing the web servers, the persistent storage nodes saved 86.8% to 78.7% of the power with the PASC approach compared to the always on storage node approach. When comparing the cluster instances, the power savings was between 78% and 39.1%. The database virtual machine saves between 80.7% and 48.3%.
Table 5.3 gives a summary comparison of the amount of power the cloud architecture uses comparing PASC to always on storage nodes. In the best case, the web server virtual machine type consumes only 13.2% of the power for storing persistent data. In the worst case, the cluster virtual machine consumes 60.9% of the power for persistent storage compared to leaving the storage nodes in an always on state.

Table 5.3: PASC Power Consumption vs Always on Storage Controllers (Random Schedule).

<table>
<thead>
<tr>
<th>Number of VMs</th>
<th>Web</th>
<th>Cluster</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13.2</td>
<td>22</td>
<td>18.4</td>
</tr>
<tr>
<td>20</td>
<td>14.8</td>
<td>24.8</td>
<td>21.4</td>
</tr>
<tr>
<td>30</td>
<td>14.3</td>
<td>23.9</td>
<td>19.3</td>
</tr>
<tr>
<td>60</td>
<td>21.3</td>
<td>35.5</td>
<td>29.8</td>
</tr>
<tr>
<td>120</td>
<td>36.5</td>
<td>60.9</td>
<td>51.7</td>
</tr>
</tbody>
</table>
5.8 Availability of Resources

One of the major potential trade-offs of developing a power conservative solution is the availability of resources. This can be understood when a cloud user requests computational resources but has to wait for the persistent storage devices to become ready before accessing their data. In our design, resource latency is consolidated to as low of a time as possible.

When a user requests computational resources in the local cloud, those resources have a time span of which they need before the user can access them. This time span is generally dominated by the operating system boot time. Other services are also configured such as placement of the virtual machine, networking interfaces, and mounting of the persistent storage. Since the user is waiting for these computational resources to become available, it is possible to move most of the latency of configuring the persistent storage resources to this same time span.

When a file is created or manipulated, a time stamp is created to notify the user or operating system of the last time the file was accessed. This time stamp is used to determine how frequently a file is accessed. Over time, the user will acquire many files in the persistent storage cloud servers. While the user is not actively using cloud resources, all of these persistent files are stored on cold storage nodes. When the user becomes active by requesting computational resources, the persistent data files are transferred from the cold storage node to a hot storage node. Using each file’s time stamp as a guide, the data is transferred based on the last time it was accessed or modified. The files with newer time stamps are transferred first until all of the data is stored on the hot storage node. This approach alleviates some of the latency while accessing persistent data by assuming “hot” data will be more likely to be accessed than “cold” data.
Chapter 6

DEPLOYMENT OF LOCAL CLOUD RESOURCES IN A LIMITED NETWORKING ENVIRONMENT

6.1 Introduction

Customers may be reluctant to move their data to off-site cloud vendors due to access regulations, cost, and trust issues [Eucalyptus, 2011c]. Instead, new software has become available for them to build their own personal cloud computing system. This gives the customer the ability to understand the issues and benefits of using cloud technology for distribution of resources before making the move to an enterprise vendor.

There is a major problem with deploying a cloud locally; network resources in terms of IP addresses are usually expensive, and therefore, limited for smaller organizations. In this chapter, a solution is proposed that needs only one public IP address for resource distribution to users of the local cloud. This approach will provide benefits of more than just efficient use of network resources. By having only one public address associated with the cloud system, security vulnerabilities are decreased. This is due to the fact that all incoming and outgoing traffic to the cloud will be associated with one IP address. This address can be monitored more efficiently than if the entire cloud system were implemented on multiple public IP addresses. In the case of our cloud construct proposition, no new network infrastructure will be needed in the organization.

In some cases, organizations may have resources geographically distributed. The idea of implementing a cloud architecture across these resources may be a possibility if a minimum of
networking resources are used to connect them. It is possible to scale an organization’s local cloud to other sites using VPN and a single IP address per distributed cluster. However, we emphasize that as demands rise to access resources not located in the local cloud, a bandwidth bottleneck may arise. Implementation of this architecture should give careful attention to the bandwidth requirements of the local cloud WAN IP address, and the bandwidth needed to connect each distributed resource to the local cloud.

In recent years, there has been an increased need for scalable computational resources in both public and private organizations. The use of cloud architectures is a great way for provisioning these resources to both the individual consumer as well as large organizations. In the past, it has been common practice for an organization to purchase dedicated hardware and software solutions to provide the infrastructure for their individual computational needs. However, with the introduction of the cloud, organizations may no longer be dependent on dedicated systems. These organizations can increase utilization on current hardware using cloud architectures. Even though most of the technologies used in creating cloud architectures are not new, the cloud gives new meaning to how users interface and consume computational resources.

Following in the footsteps of cloud leaders such as Amazon and Google, open source communities have provided software packages that allow individuals to deploy their own local cloud. There could be many reasons for a user or organization to deploy a local cloud. One example would be the user wanting a higher utilization of their physical resources. Another example could be that a public cloud vendor cannot store some data created by local organizations. Whatever the reason might be for an organization to deploy a local cloud, users still need access to their data in some way, which typically would require expensive networking resources. For example, when a
user needs to connect to a local cloud resource, each instance of that resource has to have a unique private IP address.

This chapter presents an architecture that takes the idea of deploying a local private cloud and extending it to multiple geographically distributed clusters. Each cluster, except the main cloud cluster, is composed of compute and persistent storage resources, each having only one public IP address. The main cloud cluster contains administrative components that maintain all of the cloud resources, and a graphical web interface that provides connection information to the users. Each of the auxiliary clusters maintains a connection to the main cloud cluster for commands to issue user requests for resources. Once a user is connected to a cloud cluster, all generated data, and requests for resources are confined to that cluster. This decreases the need for large data transmissions between cloud clusters.

Even though many organizations are moving to this new architecture, several issues have yet to be resolved. One of the issues is the deployment of cloud architectures into small and medium-sized businesses. Deploying such an architecture in this field requires an interface that is intuitive and not heavily demanding on the user’s technical skills. Accessing cloud resources, especially IaaS resources, is very complex without the help of a graphical interface. Users must be proficient using terminal commands for creating, accessing, and controlling cloud resources. Many users in small organizations who wish to use cloud resources may not have the knowledge or training to access these resources without some help. This may be a problem, especially in small organizations, which may have limited resources to train employees to efficiently use cloud computing resources.

This chapter also describes a simple graphical interface for Infrastructure-as-a-Service (IaaS) private clouds. IaaS clouds provide compute, storage, and network resources to users in
the form of virtual machines and network storage. In the case of our proposed architecture, the computational resources are stateless virtual machines. When a user requests resources, virtual machines are created based on CPU, RAM, and disk storage needed. The user is given a list of operating systems to choose from, and if persistent storage is needed, they have the ability to mount space located on the storage controller. Once the virtual machine is deployed on a physical compute node, the user has access to the provisioned resources. Users have the ability to request as many or as few resources they need to complete their tasks. IaaS architectures provide resources in their most unaltered form (e.g. 4 CPU cores, 4 GB RAM, 30 GB disk space, Ubuntu 10.04 server operation system). Designers of an IaaS cloud cannot make assumptions on how these resources are used as they can in the case of Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) cloud architectures. In an IaaS cloud, some set of users may be using resources for a massively parallel Hadoop (Apache Hadoop) job, while others may be hosting websites. It can be a problem to access cloud resources by some users without the help of a graphical user interface.

The platform assumed in our research and discussed in chapters three and four, adds to the problem with another layer of difficulty. While this platform reduces the cost of the private cloud operation by having only one public IP address for all cloud resources, it complicates the networking characteristics by heavy use of dynamic port forwarding. Users of this architecture, who utilize our applied graphical user interface, will only need to know the size and type of virtual machine they want in the cloud, and the outbound ports they want to use. They will be presented with a web interface, using drop down and check boxes for initial setup. The interface will return to the user how to connect to their resources through forwarded ports on the single WAN IP address of the cloud once they are available.

In summary, this chapter provides a solution for deploying a cost efficient geographically
distributed cloud architecture. This is accomplished by limiting the amount of public IP addresses and bandwidth needed, decreasing power consumed by computational and storage nodes, and designing a graphical user interface that is easy to use while hiding most of the networking complexities from the user. Section 6.2 describes how users interface with the cloud resources with limited networking resources. In section 6.3, configuration settings for the router and logging information is given. Section 6.4 gives the implementation of the user interface. Section 6.5 gives networking considerations when deploying a local cloud architecture. In section 6.6, the design of the previous sections are extended to many distributed auxiliary cloud clusters.

6.2 Cloud Interface With Network Limitations

In this section, a cloud deployment scheme is presented which allows an organization to relatively simply introduce a cloud infrastructure into their environment using a limited amount of networking resources. This deployment technique will take some initiative on the IT staff members depending on the amount of availability and scalability needed. Some things to take into consideration before deploying this type of cloud architecture are WAN port bandwidth for single cluster setups and bandwidth between the NAT routers while running the VPN protocol when deploying a distributed cluster setup.

As previously stated, the resources given to users in our setup are virtual machines hosted on the node controllers. Users do not have physical access to the cloud’s hardware resources, but can spawn virtual machines of different configurations to meet their demands. Users also have full root access to their instantiated virtual machines, and are responsible for any security precautions presented by having resources available from potentially unsecured networks (WAN).

The cloud architecture will be hosted completely within its own private network. A NAT router using DD-WRT firmware will be used to manage ports opened to the virtual machines
created by users outside of the cloud’s private network. The physical resources for the cloud would only demand the multiple servers needed, switches, and a single router. Users will access the cloud resources using commands to launch the virtual machines, and opened forwarded ports handled by the router.

6.3 Router Setup and Log Information

The NAT router is loaded with DD-WRT firmware, giving a great deal of control over its packet forwarding features. When connecting to these cloud resources, users should not worry about the networking responsibilities of providing these resources. We have the ability to dynamically open and close ports and manipulate the iptables of our private network. This depends on the current number of virtual machines in execution. Port forwarding is necessary simply because we are depending on one WAN address to connect many users to many virtual machines that could be running similar protocols.

The cloud controller keeps a list of the currently running virtual machines, as shown in Figure 6.1. This information will be used to maintain a log.txt file also located on the cloud controller. Each row in the log file is associated with an individual virtual machine. The only information required is the private IP address assigned to the virtual machine by the cluster controller, and the port forwarding information decided by the specific protocols the user chooses during launch.

Figure 6.1: Log.txt File Located on Cloud Controller.

When a virtual machine is launched, the cluster controller informs the cloud controller
of the pending network details. During the instantiation of the virtual machine(s) the graphical interface allows the user to select their preferred protocol. The default protocol used is SSH, which will initially only open port 22 on the virtual machine. All other ports will be closed, and therefore unreachable. Once the command has been sent to the cloud controller to boot a virtual machine, a process writes the private IP address of the new virtual machine to the log file. It then reads the log for protocols that are already used on other virtual machines, updates the log with the forwarded port for the user to utilize for each protocol, and notifies the user of these forwarded ports. Figure 6.2 shows the current status of the log file after a user instantiates a new virtual machine (151) with other virtual machines already running in the cloud.

![Log.txt file](image)

Figure 6.2: Example of Log.txt file With Instantiated VM’s.

Once the log file has been updated for a specific virtual machine, the process communicates with the NAT router via SSH and updates the iptables. The following code shows how the cloud controller can update the iptables:

```bash
1  iptables -t nat -I PREROUTING -p tcp --d$(nvram get wan_ipaddr) --dport 1200 -j DNAT --to 192.168.1.151:22
2  iptables -I FORWARD --p tcp --d 192.168.1.151 --dport 22 -j ACCEPT
```

After the execution of these two commands, the router will forward incoming WAN packets from port 1200 to port 22 of the virtual machine with the IP address of 192.168.1.151. All protocols will be forwarded in this manner by the NAT router.

When the user decides to terminate their virtual machine, a process on the cloud controller
updates the log file by removing the record with the terminating virtual machine’s IP address. Once this record is removed from the log, the cloud controller updates the iptables of the router by executing the following commands. The variable 1 is the record in the iptable specific to the terminating virtual machine’s IP address:

```
1  iptables -D FORWARD 1
2  iptables -D PREROUTING 1 -t nat
```

6.4 User Interface

The user interface for connection to our cloud resources is relatively simple. The user should first create an account to use on the cloud. This is accomplished by using the default Eucalyptus web interface. Once the user has an account, they must login and download the Credentials.zip file to use when connecting to virtual machines they instantiate. This file contains an RSA key, used to authenticate the user whenever they attempt to use resources on the cloud.

As shown in Figure 6.3, the user navigates to the custom web interface that gives them the choice of operating systems, size of virtual machine (CPU, RAM, disk space), and protocols needed. Once their choices are made, they launch the virtual machine. The interface returns the IP address of that virtual machine and port forwarding values for the protocols they chose. The user then has the ability to connect to the virtual machine (typically by SSH) using their RSA key. From this point, the user has root access to perform any tasks they desire.

This interface also gives the user the ability to terminate their running virtual machines. This will send the shutdown command to the VM and free any resources being used on the compute node. Once the user has released their resources, the IP address used by that user is also free to be used by other virtual machines. The requirements set for this local cloud application are that it should be web based (Cross platform, central location) and have decreased complexity for users.
This web interface was created using PHP-CLI. Once the users have selected the number and types of virtual machines they need, the interface returns enough information for them to reach these and persistent storage resources. The interface also allows for administration of cloud resources by supervisors.

Figure 6.3: User Interface Creation/Deletion of VM’s.

6.4.1 Interfacing the User to Cloud Resources

When the user needs to request cloud resources, they navigate to the login screen. This allows them to login or register for a new account. The registration screen takes the user to the default web interface for Eucalyptus as packaged with the Ubuntu Enterprise Cloud (UEC). Using this website, the user can request an account and return to the login screen once they have been authenticated. Figure 6.4 shows the Login screen.
The interface is written in PHP and communicates with a MySQL database which stores the user’s name and password for their account. The database is hosted on the main cloud cluster’s cloud controller node, along with the interface. An accounts table is all that is needed for authentication from the user.

Figure 6.4: Login Screen.

Once the user authenticates with the Login screen, they are presented with a tabbed interface that allows them to create instances, show running instances, and show persistent storage information. Figure 6.5 shows the interface for creating a virtual machine instance. The create instance tab allows the user to pick the size of the virtual machine (VM) they need. The drop-down box gives the option for five different sizes of virtual machines. These size options can be set by administrators using EUCA commands or with the administrative web interface that comes as a part of UEC. The sizes range from very small to very large. The size of the VM determines the number of virtual CPU cores, amount of RAM, and total physical disk space allocated for each instance. The VM type drop-down box gives the user the ability to select the operating system they wish to load. The last option for the user when setting up the VM is selecting the outbound
protocols. This option is needed since users will connect to cloud resources using a single public IP address. Inherently, most of the network traffic will be contained within the private cloud architecture. This gives the user the ability to connect to the VM’s using protocols, such as RDP and SSH, and gives the VM the ability to connect to outbound web and mail servers, if needed.

Once the user has logged in to the system, the user is presented with three tabs, as shown in Figure 6.5 Create Instance, Running Instances and Storage. The Create Instance tab of the user interface provides users with various options for creating new VM instances. Although not shown in the example in Figure 6.6, the user has the option of choosing desktop or server versions (“D” or “S”) and 32 or 64-bit for the operating system type. The WAN IP address is given to the user as the way to connect to their cloud resources. All of the cloud resource’s compute and storage will use a single WAN IP address. As additional geographically distributed clusters are added to the local cloud, each will need its own public WAN IP address. The LAN IP address is the VM instance’s private local IP address. The user may or may not need this information, depending on their desired job. The outbound protocols are listed based on the ones the user chooses from the initialization screen. All of the dynamic port forwarding is achieved without requiring any user input, decreasing the complexity of connecting to cloud resources. The specifics on how outbound protocols are maintained are explained in section 6.3. The user has the ability to shut down their virtual machine from within the instance, or by right clicking a particular instance in the interface and choosing terminate.

After the user initializes their VM instances, the Running Instances tab shows information on their currently available computational resources (Figure 6.6). In the Running Instance tab of the interface, each running virtual machine is listed down the page. The instance name is a unique identifier among all running virtual machines in the cloud, not just specific users’ instances. The
Figure 6.5: Initializing a New VM Instance.

size shows the option picked during instantiation. The type field shows the operating system loaded on each virtual machine. This data is gathered from the cloud controller before the request is sent to a cluster controller in an auxiliary cluster.

Figure 6.6: Running VM Instances.

The last tab that is available to the user is the Storage tab. This allows the user to locate
their persistent storage. As shown in Figure 6.7, this tab also shows their current storage usage, as well as their maximum storage quota. Generally, this network storage will be mounted in the virtual machine instance after it has been booted. We have proposed a power efficient persistent storage algorithm in which is described in Chapter 5. Persistent cloud storage is only available to users that have active virtual machines. Currently, there is no way to mount these network storage locations to machines outside of the private cloud architecture. The ability to access cloud based storage is described in Chapter 7 as a future work topic.

![Figure 6.7: Persistent Storage Information.](image)

6.5 Network Considerations

Deploying a scalable private cloud in the manner described in this paper requires careful planning to ensure availability to the cloud resources. The WAN connection will be the bottleneck when deploying this architecture. IT administrators should make sure to allocate as much bandwidth to the WAN port as possible. Generally, users will at least have SSH sessions open on each virtual machine. SSH, by protocol design, delivers constant data transmissions between the client
and server processes to minimize attacks on traffic patterns. In our design, we used a 1Gbps copper Ethernet link to supply traffic to the WAN port. This provided sufficient bandwidth to host all of our available resources (32 virtual machines) to users.

Another consideration that should be made is the selection of the NAT router. Our choice of the WRT54GL was one of low cost and ease of manipulating the firmware to allow root access. Even though we had a 1Gbps WAN link, our maximum sustainable throughput from WAN to LAN address was around 50Mbps. These speeds are typical of the 10/100Mbps Ethernet switch used by this router. This is enough to satisfy the current scale of our cloud, but would become a factor if a major increase in cloud resources were needed. Gigabit NAT routers such as the Linksys E4200 or Cisco RV220W provide excellent sustainable WAN to LAN throughput of 686Mbps and 720Mbps respectively. The implementation of the distributed multi-cluster setup described in the next section brings even more demanding requirements from the organization’s available networking resources.

6.6 Geographically Distributed Private Cloud Architecture

Depending on an organization’s geographical distribution, a multiple cluster setup may be desired. Each cluster in the Eucalyptus architecture contains the cluster and storage controller, and multiple node controllers. The DHCP service hosted on the cluster controller can be configured to accommodate unique IP subnets for each cluster. Using Eucalyptus’ default of IPv4, the private distributed cloud could potentially have on the order of 232 virtual machines. IP addressing is clearly not a scalability factor when deploying this type of cloud.

Each of the clusters in the geographically distributed cloud architecture needs a way of communicating with the main cloud cluster. This can be accomplished by using the EoiP protocol. EoiP is a service that connects remote private networks by creating a “tunnel” between the NAT routers using a public IP connection. This interface appears as an Ethernet interface and creates a
transparent LAN service connecting the remote networks as if they were on the same LAN. This is considered a “transparent” protocol because users see the network as a single connected LAN regardless of how the private networks are connected, e.g. a fiber optic network or a wireless network.

Figure 6.8: Distributed Cloud Architecture.

As shown in Figure 6.8, all auxiliary cloud clusters are connected with the main cloud cluster using a single public IP address. Auxiliary cloud clusters have the same architecture as the main cloud cluster without hosting their own cloud controller. Resource requests on auxiliary cloud clusters come from the main cloud cluster’s cloud controller node. The EoIP protocol is
only used for administrative communication between auxiliary cloud clusters and the main cloud cluster. No user-generated data is transferred between cloud clusters. Instead, it is expected that users login to the cluster where their data is stored. This keeps the communication between cloud clusters to a minimum and reducing the need for higher bandwidth. Figure 6.9 show the number of virtual machines that can be requested from different cloud clusters. The user or administrator can query the cloud controller for this information. There are, by default, five different sizes of virtual machines that can be requested. Each virtual machine differs in the number of CPU cores, amount of RAM needed in megabytes, and local hard disk space needed in gigabytes. The number of available virtual machines is given, determined by the number (and physical characteristics) of compute nodes in each of the cloud clusters.

![Figure 6.9: Example of Available VM’s in Different Cloud Clusters.](image)

### 6.6.1 Configuration of EoiP

The EoiP configurations are set in the NAT routers for each of the cloud clusters. Each router must have a static public IP address and a firmware that allows direct manipulation of the
Table 6.1: EoiP Setup.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Private Subnet</th>
<th>EoiP Interface</th>
<th>EoiP Gateway</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Cloud Cluster</td>
<td>192.168.1.x</td>
<td>192.168.100.1</td>
<td>192.168.100.2-n</td>
</tr>
<tr>
<td>Auxiliary Cluster 1</td>
<td>192.168.2.x</td>
<td>192.168.100.2</td>
<td>192.168.100.1</td>
</tr>
<tr>
<td>Auxiliary Cluster 2</td>
<td>192.168.3.x</td>
<td>192.168.100.3</td>
<td>192.168.100.1</td>
</tr>
<tr>
<td>Auxiliary Cluster n</td>
<td>192.168.n.x</td>
<td>192.168.100.n</td>
<td>192.168.100.1</td>
</tr>
</tbody>
</table>

IPTables. In our design, each cloud cluster has a unique private IP subnet. The main cloud cluster uses IP addresses 192.168.1.1, 192.168.1.2, etc. and auxiliary clusters each have unique subnets: 192.168.2.x, 192.168.3.x, etc.

To connect the main cloud cluster to the first auxiliary cluster, we make sure to set the subnet of the main cloud cluster’s router to 1.x and the subnet of the auxiliary cluster’s router to 2.x. The EoiP subnet is then set to a private IP address range that is not used for identifying physical devices or virtual machines. In this example, we chose 192.168.100.0. All EoiP traffic from any cluster’s NAT router will be sent to this address. The main cloud cluster will be designated as EoiP address 192.168.100.1. All other clusters will be assigned 100.2, 100.3, etc. for communication back to the main cloud controller.

As shown in table 6.1, the EoiP gateway for each of the auxiliary clusters connects back to the main cloud cluster. This allows communication between the main cloud cluster and all auxiliary clusters, but no communication between auxiliary clusters. Users communicate with the web interface located in the main cloud cluster when requesting resources and connect to either the main cluster, or one of the auxiliary clusters once the resources are provided. The user remains connected to only one cluster for the time they are active.
6.6.2 Scalability Considerations With a Distributed Architecture

The potential for scaling using a distributed architecture ideally should be unlimited. Careful consideration should again be given to the available bandwidth to the WAN ports of each NAT router. The traffic generated in this setup is substantially greater than in the single cluster architecture. Organizations may consider using protocols such as ATM or others that guarantee a minimum bandwidth between NAT routers. The NAT router selected for this architecture is also more important than the one used in the single cluster architecture due to the increased demands on networking resources.

Users will still use the single point of interface with the main cloud controlled cluster. There is no cloud controller in the auxiliary clusters, and therefore, no interface to use those resources directly.
Chapter 7

FUTURE WORK

This dissertation focused on creating a cost effective geographically distributed private cloud architecture. A local cloud architecture can provide computational, storage, and networking resources to users of an organization in a very efficient way. To support ease of usage, a robust graphical interface is required. In addition to interfacing requirements, the local cloud architecture should be low cost. The area of cloud computing is ever evolving, so this local cloud implementation should do the same.

7.1 Cloud Architecture

The future research direction of this power aware local IaaS cloud architecture is to rebuild the cloud middleware and remove the need for Eucalyptus. Most of the work presented in this dissertation uses the Eucalyptus cloud middleware. Beginning with Eucalyptus 1.5, each iteration of Eucalyptus was used until Eucalyptus version 3.1. A future direction would be to build a more optimized version of the cloud middleware that has lower overhead for administration and tailored for an organization’s specific needs. The architecture does not have to be a vertical cloud architecture, just one that encompasses the organization’s needs with lower overhead than general purpose open source cloud architectures.

7.1.1 Cloud Middleware API

There are multiple approaches to creating a lightweight cloud middleware. The goal for the middleware would ultimately be the same; give users efficient access to cloud resources, handle
faults, and be scalable according to user requests. The cloud middleware would be responsible for internode cloud communication. A cloud controller node would communicate with other cloud resources using the middleware.

The first approach would have cloud resource communication using SSH. In this architecture, the cloud controller would open an SSH connection to other cloud resources to perform administrative commands. A second approach would be to implement the cloud middleware using socket programming. In this case, the cloud controller and all other administrative components in the cloud architecture would execute as a process the cloud middleware. This middleware would be used for communication of administrative tasks.

7.2 Cloud Compute Resources

The compute resources in the local cloud architecture can be optimized as well to accommodate different jobs. Since the local cloud architecture is a heterogeneous design, some machines will respond to certain requests better than others. Compute nodes can also be optimized for different guest operating systems, such as different Linux distributions or Windows.

7.2.1 GPU Related Compute Resources

Compute nodes could house GPU cards for OpenMP or Cuda accelerated jobs. This is a relatively new research field, which allows specific hypervisors to give user’s access to the GPU resources through a virtualized interface. Nvidia has released their K1 and K2 grid GPU cards specifically for this purpose. Currently there are two hypervisors which support the Nvidia grid cards, Citrix’s Xen with VGX software and VMware’s View 5.2 with vSGA.

7.3 Virtual Machine Live Migration

Virtual machine live migration is a researched topic in the area of cloud computing. In this dissertation, the approach of virtual machine live migration gives the advantage to lowering
the power consumption while handing job requests. An approach of changing the virtual machine placement algorithm for load balancing will give the advantage to virtual machine performance. Load balancing achieves this at the cost of consuming a higher amount of power. The future work direction would involve generating a dynamic approach for switching between a load consolidation and load balancing approach. This new approach would change the dynamics of the cloud architecture to give a higher performance model during times of heavy usage, and switch to a power conservative state when usage is lower.

Other areas of future work dealing with virtual machine live migration are evaluating the performance of multiple virtual machines while in migration, fault tolerance of the cloud architecture during a live migration, and migrating guest operating systems between different hypervisors. Virtual machine caching will also be examined. The idea of a virtual machine cache has many commonly requested virtual machines booted and running idle on a specialized compute node called the virtual machine cache node. Once a user requests one of these virtual machines, they will be migrated from the virtual machine cache node to a free compute node. This approach will decrease the latency of the user having to wait for the virtual machine to boot before they can use cloud resources.

7.4 Cloud Storage

Persistent cloud storage is an important design feature with future work possibilities. One enhancement to persistent storage is to give users access to their data without consuming compute node resources. Another future work direction is the micromanagement of hard drives and SSD devices in the storage nodes to give a power savings or performance strategy similar to the future work direction of the computational resources.

Persistent data storage is a heavily researched topic in the area of cloud computing. To
extend the proposed cloud architecture described in this dissertation, many future directions can be applied. Replication of data in the cloud architecture is one direction. Most public data centers tend to have a replication count of at least three when it comes to users data. In our cost efficient local cloud architecture, a replication count of N can be achieved by implementing a network RAID. In the network RAID, there is one primary storage node, and N backup nodes. The user always connects to the primary node for data access. Any manipulation of the data on this primary storage node is immediately replicated to the backup nodes. In the case of a node failure, specifically the primary storage node, a backup storage node becomes the primary storage node.

Furthering our research on persistent cloud storage we plan on involving solid state disk drives in the future. The advancement of this technology brings the lifespan of the disk close to that of a traditional magnetic disk. The SSD also allows for much higher data throughput from the disk, seven to eight times higher on average than HDDs. The SSD also consumes less power in both active and idle states. Applying SSDs and RAID configurations at the hardware level of the storage node should increase the efficiency of the IaaS cloud.

Another improvement would be implementing multiple overlay networks connecting storage and compute resources to the cloud network backbone. As of our current experiment with storage nodes using HDDs, a single gigabit Ethernet interface was sufficient in accommodating users of the cloud resources since the throughput of the disk was lower than the network interface of the storage node. Increasing the disk throughput with SSDs will saturate the IaaS cloud network and be limited on performance. We will be exploring cost efficient options for network interfaces that could be considered by small to medium sized organizations.
7.5 Interfacing with Cloud Resources

A last future work effort is to increase the overall performance and usefulness of the cloud interface. Giving cloud users and easy way to administer their resources is important for the long term support of the local cloud architecture. The proposed cloud middleware API earlier in this chapter could be invoked from a web interface, giving the user access to their cloud resources.
Chapter 8

CONCLUSION

The objective of this research is to achieve a local cloud architecture that can be implemented in an environment with cost constraints.

Cloud computing is the dynamic allocations of platforms, software, and hardware. There are many vendors that are emerging to provide these services to the customer. These services are in the public cloud, accessible to anyone depending on the vendor’s subscription. The customer who wishes not to move their sensitive data to the Internet can implement private clouds. Once the cloud, public or private, is instantiated, the customer can access the services through a client device. In terms of reliability and availability, vendors make hardware and software disruptions transparent by replicating resources throughout many data centers. Security will always be a factor when moving data to the cloud. Most cloud storage vendors have encryption measures in place to ensure the protection of their customer’s data, but additional security measures are needed. Another major aspect of cloud based computing is load balancing. Load balancing allows the cloud vendors to expand their services by simply adding more physical resources. The load balancing algorithms also distribute the workload among the physical devices, potentially giving the customer the most efficient access to the cloud vendor’s resources.

With the increased use of local cloud computing architectures, organizations are becoming aware of wasted power consumed by underutilized resources. We introduced a load balancing algorithm that balances resources across available compute nodes in a cloud with power savings in
mind. Since the cloud architecture implemented by local organizations tends to be heterogeneous, we take this into account in our design. The Power Aware Load Balancing algorithm, PALC, maintains the state of all compute nodes, and based on utilization percentages, decides the number of compute nodes that should be operating.

8.1 PALC

Using PALC, organizations wanting to build local clouds using Eucalyptus would be able to save on energy costs. This is due to the fact that Eucalyptus does not account for power consumption when applying its default load balancing technique. Depending on the job schedule distribution and virtual machine request size, organizations can save 70% to 97% of the energy consumed compared to using load balancing techniques that are not power aware.

8.2 Virtual Machine Live Migration

Virtual machine live migration allows for cloud administrators to change the approach of how resources are accommodated. Chapter 4 introduced the idea of migrating a virtual machine, and presented three performance metrics on the topic. The evaluated metrics were how the virtual machine performed during a live migration, the cloud architecture performance of performing a live migration, and the amount of power consumed while the virtual machine was being migrated. For each experiment, three different types of virtual machines were used. A Web server was used for its high network saturation, Cluster virtual machine for its high CPU saturation, and a database virtual machine for its high disk I/O saturation.

8.2.1 Virtual Machine Performance During Live Migration

Migrating the virtual machine from one physical server to another has an impact on the performance of that virtual machine. In the experiments the web server virtual machines had the largest decrease in performance. This was due to saturation of the network from HTTP requests
and performing a live migration. The cluster based virtual machine had the least amount of decrease in performance. This is due to the cluster based virtual machine being CPU intensive. The computational process was copied to the destination server with little impact on disk I/O or networking constraints.

8.2.2 Cloud Performance During Live Migration

While performing the live migrations, as expected, the web servers take the longest time to migrate. This is due to the network saturation of using the web server while performing the live migration. It is useful to note here that the smallest instance of the web server was unable to migrate due to resource constraints. The small instance continued to execute on the original compute node. The cluster virtual machine had a maximum migration time of 13.5 seconds. Also, the smallest instance of the cluster virtual machine was unable to migrate due to resource constraints. This instance became unstable after initiating a live migration, and is unreachable once the migration is finished. The database virtual machine was the only type to migrate every virtual machine size. The small and medium sized database virtual machines have similar migration performances while the large instance takes longer due to number of allocated CPU cores, RAM, and Disk space.

8.3 PASC

Persistent storage is necessary in a local cloud architecture. A power aware storage balancing algorithm was introduced that decreases the power consumption of storage nodes. This algorithm takes into account the number of active users in the local cloud and makes only their persistent storage available on hot storage nodes. Other cloud users have their data located on cold storage nodes that are normally in a powered off state. This scheme of storing persistent data will reduce the power requirement of hosting persistent storage compared to leaving all storage nodes powered on.
As organizations move towards consolidation of resources by using technologies such as IaaS cloud architectures, power consumption of unused components is becoming a major concern. This paper introduces a power saving consolidation approach to persistent cloud storage that uses as few storage node resources as possible to accommodate users of the private cloud. Organizations designing a local IaaS cloud architecture can benefit in terms of reducing costs by implementing power aware persistent storage algorithms. The default action for persistent network data storage is to always have the storage resources powered on and available to users. This will potentially waste power when the number of users is low. Based on a typical job schedule, organizations can save between 65% and 92% on wasted power consumption of persistent storage devices.

8.4 Interfacing with Geographically Distributed Cloud Resources

This dissertation presents the idea of deploying a private cloud using a limited amount of networking resources. Organizations have the ability to create a private cloud and introduce it into their network without greatly affecting their current infrastructure. With the introduction of the single cluster cloud, users are able to request resources through a simple interface. Once the user has access to these resources, they can use it to perform any task they desire. When the user no longer needs the resources, they are released and can be used by others. The concept of a distributed cluster architecture was also introduced. In this architecture, EoiP is used to connect the auxiliary cloud clusters back to the main cloud cluster. Each cluster is hosted on its own subnet of IP address, giving each virtual machine the ability to have a unique address.

Scalability needs careful consideration while implementing the single or distributed cloud architecture. Administrators should be aware that the single WAN ports could be heavily utilized and should be allocated as much bandwidth as possible. This is especially true in the case of the distributed cloud topology. Administrators should also carefully consider the NAT router used
when deploying this cloud. A few NAT router suggestions were given, and if more performance is needed, a dedicated NAT routing server could be used.

The combined solutions illustrated in this dissertation provide insights on developing a local cloud architecture well suited for small to medium-sized organizations. The goal is reducing power consumption while maintaining high availability to cloud resources. The architecture presented in this dissertation provides a comprehensive solution for deploying a local cloud architecture that is cost efficient to maintain.
REFERENCES


Corradi, A., M. Fanelli, and L. Foschini (2011). Increasing cloud power efficiency through con-


Eucalyptus (2011b). "eucalyptus cloud software".


Kandaswamy, G., L. Fang, Y. Huang, S. Shirasuna, S. Marru, and D. Gannon (2006). Build-
ing web services for scientific grid applications. *IBM Journal of Research and Development* 50(2.3), 249–260.


Mike Bradley, D. L. (November 16, 2009). Oak ridge ’jaguar’ supercomputer is world’s fastest. Oak Ridge National Laboratory.


SunMicrosystems (2010). Introduction to cloud computing architecture by sun microsystems, inc.


University, S. (2011). "folding@home distributed computing".


Appendices
Appendix A
BUILDING AND DEPLOYING VIRTUAL MACHINES USING UBUNTU 12.04.1 LTS

This guide explains how you can install and use KVM for creating and running virtual machines on an Ubuntu 12.04 LTS server. It will show how to create image-based virtual machines. KVM is short for Kernel-based Virtual Machine and makes use of hardware virtualization, i.e., you need a CPU that supports hardware virtualization, e.g., Intel VT or AMD-V.

A.A Preliminary Note
We need to execute the steps in the tutorial with root privileges. We can either execute all commands using “sudo”, or become root by typing:

```
sudo su
```

A.B Installing KVM and vmbuilder

Check if your CPU supports hardware virtualization by executing the command:

```
egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

should display something like this:

```
root@node1:~ # egrep '(vmx|svm)' --color=always /proc/cpuinfo
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
        → clflush mmx fxsr sse sse2 ht syscall nx mmxt x fxsr_opt rdtscp lm 3dnowext 3dnow rep_good
        → nopl extd_apicid pni cx16 lahf_lm cmp_legacy svm extapic cr8_legacy 3dnowprefetch lbrv
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
        → clflush mmx fxsr sse sse2 ht syscall nx mmxt x fxsr_opt rdtscp lm 3dnowext 3dnow rep_good
        → nopl extd_apicid pni cx16 lahf_lm cmp_legacy svm extapic cr8_legacy 3dnowprefetch lbrv
root@node1:~ #
```

If nothing is displayed, then your processor doesn’t support hardware virtualization, and you must stop here.

To install KVM and vmbuilder (a script to create Ubuntu-based virtual machines), execute:

```
apt-get install ubuntu-virt-server python-vm-builder kvm-pxe
```

Afterwards we must add the user as which we’re currently logged in (root) to the group “libvirtd”:

```
adduser `username` libvirtd
adduser `username` kvm
```

*You need to log out and back in again for the new group memberships to take effect.

To check if KVM has successfully been installed, run:

```
virsh --c qemu:///system list
```

It should display something like this:

```
root@node1:~ # virsh --c qemu:///system list
Id Name State
________________________________________
root@node1:~ #
```

If it displays an error instead, then something went wrong.

Next we need to set up a network bridge on our server so that our virtual machines can be accessed from other hosts as if they were physical systems in the network.

To do this, we install the package bridge-utils...
apt-get install bridge-utils

... and configure a bridge. Open `/etc/network/interfaces`:

`vim /etc/network/interfaces`

Before the modification, the interfaces file looks like:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
    address 192.168.1.245
    netmask 255.255.255.0
    network 192.168.1.0
    broadcast 192.168.1.255
    gateway 192.168.1.1
    dns-nameservers 8.8.8.8 8.8.4.4

# The secondary network interface
auto br0
iface br0 inet static
    address 192.168.1.245
    network 192.168.1.0
    netmask 255.255.255.0
    broadcast 192.168.1.255
    gateway 192.168.1.1
    dns-nameservers 8.8.8.8 8.8.4.4
    bridge_ports eth0
    bridge_fd 9
    bridge_hello 2
    bridge_maxage 12
    bridge_stp off
```

(Make sure you use the correct settings for your network!)

Then restart the network:

```
/etc/init.d/networking restart
```

*NOTE: If you are remotely connected using SSH, your connection may drop once you restart networking! This may happen if you update eth0 to a static IP address rather than using a DHCP IP address. Just SSH back into the machine using the new IP address if this happens.

Then execute:

```
ifconfig
```

It should now show the network bridge (“br0”):
Before creating the virtual machine, reboot the system:

```
reboot
```

If you don’t do this, you might get an error like open”/dev/kvm: Permission denied” in the virtual machine logs in the “/var/log/libvirt/qemu/” directory.

A.C Creating an Imaged-Based Virtual Machine

We can now create our first virtual machine - an image-based VM.

I want to create my virtual machines in the directory “/var/lib/libvirt/images/” (they cannot be created in the “/root” directory because the “libvirt-qemu” user doesn’t have read permissions in that directory).

We will create a new directory for each VM that we want to create, e.g. “/var/lib/libvirt/images/vm1”, “/var/lib/libvirt/images/vm2”, “/var/lib/libvirt/images/vm3”, and so on, because each VM will have a subdirectory called “ubuntu-kvm”, and obviously there can be just one such directory in “/var/lib/libvirt/images/vm1”, for example. If you try to create a second VM in “/var/lib/libvirt/images/vm1”, for example, you will get an error message saying “ubuntu-kvm already exists” (unless you run vmbuilder with the –dest=DESTDIR argument):

```
Error Message:
```

```
root@node1:/var/lib/libvirt/images/vm1# vmbuilder kvm ubuntu -o vm2.cfg
2009–05–07 16:32:44.185 INFO Cleaning up
ubuntu–kvm already exists
root@node1:/var/lib/libvirt/images/vm1#
```
We will use the “vmbuilder” tool to create VMs. (You can learn more about vmbuilder [http://help.ubuntu.com/community/JeOSVMBuilder vmbuilder].) “vmbuilder” uses a template to create virtual machines - this template is located in the “/etc/vmbuilder/libvirt/” directory. First we create a copy:

1. `mkdir -p /var/lib/libvirt/images/vml/mytemplates/libvirt`
2. `cp /etc/vmbuilder/libvirt/* /var/lib/libvirt/images/vml/mytemplates/libvirt/`

Now we come to the partitioning of our VM. We create a file called “vmbuilder.partition”:

1. `vim /var/lib/libvirt/images/vml/vmbuilder.partition`

and define the desired partitions as follows:

```
root 8000
swap 4000
--
/var 20000
```

This defines a root partition (“/”) with a size of 8000MB, a swap partition of 4000MB, and a “/var” partition of 20000MB. The — line makes that the following partition (“/var” in this example) is on a separate disk image (i.e., this would create two disk images, one for root and swap and one for “/var”). Of course, you are free to define whatever partitions you like (as long as you also define root and swap), and of course, they can be in just one disk image - this is just an example.

I want to install “openssh-server” in the VM. To make sure that each VM gets a unique OpenSSH key, we cannot install “openssh-server” when we create the VM. Therefore we create a script called “boot.sh” that will be executed when the VM is booted for the first time. It will install “openssh-server” (with a unique key) and also force the user (I will use the default username “kvmuser” for my VMs together with the default password “ubuntu”) to change the password when he logs in for the first time:

1. `vim /var/lib/libvirt/images/vml/boot.sh`

Edit the boot.sh script to contain:

```
# This script will run the first time the virtual machine boots
# It is run as root.

# Expire the user account
passwd --e kvmuser

# Install openssh-server
apt-get update
apt-get install --force-yes openssh-server

*Notice the user account is set to kvmuser.

Now take a look at:
1. `vmbuilder kvm ubuntu --help`

To learn about the available options.

To create our first VM, vm1, we go to the VM directory:

```
echo /var/lib/libvirt/images/vml/|
```

and run “vmbuilder”:

```
vmbuilder kvm ubuntu --suite=precise --flavour=virtual --arch=amd64
--mirror=http://de.archive.ubuntu.com/ubuntu --o --libvirt=qemu:///system --ip=192.168.1.246
--gw=192.168.1.1 --part=vmbuilder.partition --templates=mytemplates --user=kvmuser --name=kvmuser
--pass=ubuntu --addpkg=vim-nox --addpkg=unattended-upgrades --addpkg=acpid
--firstboot=./var/lib/libvirt/images/vml/boot.sh --mem=256 --hostname=vm1 --bridge=br0
```
Most of the options are self-explanatory. “–part” specifies the file with the partitioning
details, relative to our working directory (that’s why we had to go to our VM directory before
running vmbuilder), “–templates” specifies the directory that holds the template file (again relative
to our working directory), and “–firstboot” specifies the firstboot script. “–libvirt=qemu:///system”
tells KVM to add this VM to the list of available virtual machines. “–addpkg” allows you to
specify Ubuntu packages that you want to have installed during the VM creation (see above why
you shouldn’t add “openssh-server” to that list and use the firstboot script instead). “–bridge” sets
up a bridged network; as we have created the bridge “br0” above, we specify that bridge here.

In the “–mirror” line, you can specify an official Ubuntu repository in “–mirror”, e.g.
“http://de.archive.ubuntu.com/ubuntu”. If you leave out “–mirror”, then the default Ubuntu repos-
itory (“http://archive.ubuntu.com/ubuntu”) will be used.

If you specify an IP address in the “–ip” switch, make sure that you also specify the correct
gateway IP using the “–gw” switch (otherwise vmbuilder will assume that it is the first valid address
in the network which might not be correct). Usually the gateway IP is the same that you use in
“/etc/network/interfaces” (see above).

The build process can take a few minutes.

Afterwards, you can find an XML configuration file for the VM in “/etc/libvirt/qemu/” (=>
“/etc/libvirt/qemu/vm1.xml”):

```
ls -l /etc/libvirt/qemu/
root@node1:/var/lib/libvirt/images/vml# ls -l /etc/libvirt/qemu/
total 8
-rw------- 1 root root 4096 May 21 13:00 networks
-rw------- 1 root root 2082 May 21 13:15 vm1.xml
root@node1:/var/lib/libvirt/images/vml#
```

The disk images are located in the ubuntu-kvm/ subdirectory of our VM directory:

```
ls -l /var/lib/libvirt/images/vml/ubuntu-kvm/
root@node1:/var/lib/libvirt/images/vml# ls -l /var/lib/libvirt/images/vml/ubuntu-kvm/
total 604312
-rw------- 1 root root 324337664 May 21 13:14 tmpE4iRv.qcow2
-rw------- 1 root root 294715392 May 21 13:15 tmpxvSVOT.qcow2
root@node1:/var/lib/libvirt/images/vml#
```

A.D Managing and Connecting to Your Virtual Machine

VMs can be managed through “virsh”, the “virtual shell”. To connect to the virtual shell, run:

```
virsh --connect qemu:///system
```

This is how the virtual shell looks:

```
root@node1:~# virsh --connect qemu:///system
Welcome to virsh, the virtualization interactive terminal.
Type: 'help' for help with commands
'quit' to quit

virsh 
```

To show all virtual machines, running and inactive:

```
virsh list --all
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>vml1</td>
<td>shut off</td>
</tr>
<tr>
<td>-</td>
<td>vm2</td>
<td>shut off</td>
</tr>
</tbody>
</table>
Before you start a new VM for the first time, you must define it from its xml file (located in the “/etc/libvirt/qemu/” directory):

```bash
virsh define /etc/libvirt/qemu/vm1.xml
```

*Please note that whenever you modify the VM’s xml file in /etc/libvirt/qemu/, you must run the define command again!

Now you can start the VM:

```bash
virsh start vm1
```

After a few moments, you should be able to connect to the VM with an SSH client such as PuTTY; log in with the default username and password. After the first login you will be prompted to change the password.

```bash
virsh list
```

should now show the VM as running:

```
ID   Name   State
---  ------  ----
1    vm1    running
```

“To connect to the vm:”

```bash
ssh kvmuser@192.168.1.246
```

You will be asked to add the RSA keypair to the known hosts file, and change your password for the “kvmuser” account on vm1.

“Congratulations!! You are now connected to your virtual machine!”
Appendix B
BASH SCRIPT FOR PERFORMING A LIVE MIGRATION OF A KVM VIRTUAL MACHINE USING VIRSH

```bash
#!/bin/bash

string=$(cat /proc/cpuinfo | grep -i "vmx|svm")

function printImages {
  echo $1
}

if [[ $string == *vmx* || $string == *svm* ]]
then
  D_ERROR=$(/(dpkg --list ubuntu-virt-server) 2>&1)
  if [[ $D_ERROR == *not installed* ]]
  then
    # ask for a username
    read -p "Enter a username to run kvm : " username;
    if [[ username == "" ]]
    then
      echo "No username given.";
      exit
    fi
    # Install the crap
    apt-get -y install ubuntu-virt-server python-vm-builder kvm-ipxe
    adduser root libvirtd
    adduser root kvm
    adduser $username libvirtd
    adduser $username kvm
    if [[ $(dpkg --list bridge-utils) 2>&1) == *not installed* ]]
    then
      apt-get -y install bridge-utils
      echo *
      # This file describes the network interfaces available on your system
      and how to activate them. For more information, see interfaces(5).
      auto lo
      iface lo inet loopback
      auto eth0
      iface eth0 inet manual
      auto br0
      iface br0 inet static
        address 192.168.1.245
        network 192.168.1.0
        netmask 255.255.255.0
        broadcast 192.168.1.255
        gateway 192.168.1.1
        dns-nameservers 8.8.8.8 8.8.4.4
        bridge_ports eth0
```
bridge_fd 9
bridge_hello 2
bridge_maxage 12
bridge_stp off
vim /etc/network/interfaces
service networking restart
if [[ $(ifconfig) != br0* ]]
then
echo "Error with br0-ski"
fi
echo "Rebooting"
reboot
else
  # after boot?
  IMAGES=$(ls /var/lib/libvirt/images/)
  printImages $IMAGES
  read -p "Enter an image name : " image_name
  echo "Creating image $image_name"
  $(mkdir -p /var/lib/libvirt/images/$image_name/mytemplates/libvirt)
  $(cp /etc/vmbuilder/libvirt/* /var/lib/libvirt/images/$image_name/mytemplates/libvirt/)
  echo "  root 4000
  swap 2000
  --- > /var/lib/libvirt/images/$image_name/vmbuilder.partition"
  echo "  # This script will run the first time the virtual machine boot
  # It is ran as root.
  # Expire the user account
  passwd -e kvmuser
  # Install openssh-server
  apt-get update
  apt-get install -qqy --force-yes openssh-server" > /var/lib/libvirt/images/$image_name/boot.sh
  cd /var/lib/libvirt/images/$image_name/
  read -p "Enter VM IP Address : " IP
  read -p "Enter Gateway IP Address : " GW
  read -p "Enter Bridge name " BR
  read -p "Is this correct? $IP / $GW / $BR" YN
  while [[ $YN != [Yy] ]]; do
    read -p "Enter VM IP Address : " IP
    read -p "Enter Gateway IP Address : " GW
    read -p "Enter Bridge name " BR
    read -p "Is this correct? $IP / $GW / $BR" YN
  done
  vmbuilder kvm ubuntu --suite=precise --flavour=virtual --arch=amd64 \
  --mirror=http://de.archive.ubuntu.com/ubuntu --libvirt=qemu://system --ip=$IP \
  --gw=$GW --part=vmbuilder.partition --templates=mytemplates --user=kvmuser --name=kvmuser \
  --pass=ubuntu --addpkg=vim-nox --addpkg=unattended-upgrades --addpkg=acpid \
  --firstboot=/var/lib/libvirt/images/$image_name/boot.sh --mem=256 --hostname=$image_name \
  --bridge=$BR
  virsh define /etc/libvirt/qemu/$image_name.xml
clear
read -p "Start VM? " yn
if [[ $yn == [Yy] ]]
then
   virsh start $image_name
fi
fi
Appendix C
PERFORMING A LIVE MIGRATION OF A KVM VIRTUAL MACHINE USING VIRSH AND SSH WITHOUT SHARED STORAGE

This guide explains how you can perform a virtual machine live migration from physical server A to physical server B with no shared persistent storage between the two servers. This assumes both servers have Virsh and SSH installed and both servers can communicate with each other.

C.A Preliminary Note

We need to execute the steps in the tutorial with root privileges. We can either execute all commands using ”sudo”, or become root by typing:

```
sudo su
```

First check to see if KVM is installed:

```
root@node1: ~ $ kvm --version
QEMU emulator version 1.0 (qemu-kvm-1.0), Copyright © 2003–2008 Fabrice Bellard
```

If you receive this message:

```
The program kvm is currently not installed. You can install it by typing:
sudo apt-get install qemu-kvm
```

Then, KVM is not installed. You must install KVM on both physical servers before performing a live migration.

C.B Procedure

Performing a live migration without shared storage requires the Virsh migrate flags –copy-storage-all or –copy-storage-inc. Comments on these Virsh flags:

```
--copy-storage-all
Indicates migration with non-shared storage with full disk copy

--copy-storage-inc
Indicates migration with non-shared storage with incremental copy (same base image shared between source and destination).

--copy-storage-
Options only tell libvirt to transfer data from the images on source host to the images found at the same place on the destination host.
```

The server node1 has a virtual machine named vm1. This virtual machine is to be migrated to the server node2.

To list the virtual machines currently running on node1:

```
root@node1: ~ $ virsh list --all
```

should now show the VM as running:

```
Id Name     State
------- ------
  1 vm1     running
```

Next, list the virtual machines currently running on node2:

```
root@node2: ~ $ virsh list --all
```

should now show the VM as running:
The live migration of the virtual machine uses TCP port 49125 and 49126. To ensure communication between the two physical servers, we must update their firewall. Since we are using Ubuntu 12.04.1 LTS, we can modify the iptables using the Uncomplicated Firewall (ufw). The ufw tool by default is initially disabled.

To enable ufw:

```
root@node1: ufw enable
```

Now we open the ports for live virtual machine migration:

```
root@node1: ufw allow 49125
Rule added
Rule added (v6)
root@node1: ufw allow 49126
Rule added
Rule added (v6)
```

This should be performed on both node1 and node2 physical servers.

Next, on node1, create a virtual disk using qemu-img:

```
root@ubuntu:-2:$ qemu-img create -f qcow2 /var/disk1/images/ubuntu-vm1.img 10G
Formatting '/var/disk1/images/ubuntu-vm1.img', fmt=qcow2 size=10737418240 encryption=off
```

Once the virtual disk is created on node1, node2 can begin the live migration of the virtual machine:

```
root@node1:-# virsh migrate --live --copy-storage-all vm1 qemu+ssh://ubuntu-2/system  --verbose
root@node2's password:
Migration: [ 0 %]
```

The live migration is performed, giving the user a percentage until it completes.

```
root@node1:-# virsh migrate --live --copy-storage-all vm1 qemu+ssh://ubuntu-2/system  --verbose
root@node2's password:
Migration: [ 100 %]
```

Once the virtual machine completes the migration, it will be running on node2:

node1:

```
root@node1:-# virsh list --all
Id Name State
--- ------- ----- 
- vml shut off
```

node2:

```
root@node2:-# virsh list --all
Id Name State
--- ------- ----- 
- 1 vml running
```