THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# INVESTIGATING APPLICATION LATENCY IN DISTRIBUTED LOW POWER LOW RELIABILITY DATA CENTERS

SAUL WECHT
SPRING 2013

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate
in Computer Engineering
with honors in Computer Engineering

Reviewed and approved* by the following:

Bhuvan Urgaonkar
Associate Professor
Thesis Supervisor

Lee Coraor
Associate Professor, Director of Academic Affairs
Honors Adviser

* Signatures are on file in the Schreyer Honors College.

# ABSTRACT

In this paper, we examine the effect of a geo-distributed client-server relationship on application performance. Today, data centers are built to be very high reliability and as a result they have a very high level of power consumption. This paper is the beginning of a study to determine the feasibility of moving to a distributed system in which redundant lower reliability and therefore lower power data centers are used instead.

This paper investigates the latency (and therefore performance) impact that geo-distribution may have on an application which utilizes a data center. Specifically, we investigated the various latencies associated with applications representative of common usage scenarios when subjected to varying distances between clients and servers. This paper will focus on the MediaWiki application and a simple web service. These applications were chosen because of their typically centralized and high-demand usage scenario.

# TABLE OF CONTENTS

# LIST OF TABLES

# ACKNOWLEDGEMENTS

The writer would like to acknowledge Bhuvan Urgaonkar, Sriram Govindan, and Aman Kansal for their patience and support throughout the experimentation and thesis-writing process.

# Chapter 1

## Introduction

With the increasing focus on cloud computing in today's technology marketplace and the rise of online data storage and manipulation services, data centers have become essential tools. However, due to the always-on nature of the "cloud", data centers must be built to be highly reliable. As a result, many redundancy and failsafe mechanisms are included in their construction. Redundant data copies, backup power sources, and similar mechanisms are all incorporated into modern data centers in order to maximize up-time. These mechanisms, while successful in fulfilling their purpose, consume often unnecessarily high levels of power.

To argue that these mechanisms are unnecessary would be misguided, however. Data is stored on hard disk drives, and hard disk drives are mechanical devices that are prone to failure. Imagine then, if there were no redundant or error recovery systems and a single hard disk failure in a data center meant permanently lost data for paying costumers. That data could be banking information, email, documents, or any number of other different types of digital information. In today's technology landscape, failure is not an option.

In addition to this, data centers are built to withstand higher levels of traffic than they ever expect to see. Servers are built, programmed, and plugged in as an assurance that should there someday be a spike in activity, the data center will be able to handle it. As a result, many servers remain idle for the majority of their lives, waiting for that unexpected spike in activity. This translates directly to wasted power.

So, the question becomes this: How do we maintain a reliable high-performance data center while reducing its exorbitant energy demands? The solution that is explored in this paper

suggests a simple approach. Instead of building one very high reliability data center, build two independent lower reliability data centers. The probability that both will fail simultaneously can be made similar to, if not less likely than, the original failure probability (since two independent unlikely events are very unlikely to occur simultaneously). In this scenario, we imagine two redundant data centers with considerably less redundancy and error prevention/recovery features located in significantly different places (East and West coasts of the United States, for instance). The geo-location difference insures that the two data centers will not be disabled by the same regional event and gives the data centers independence from each other. The lack of redundancy and error prevention/recovery features will significantly reduce the power demand of each data center to the point where even having two of them is still less demanding than the original one.

The above claim of power usage in two smaller data centers may seem counter-intuitive at first, and is worth explaining. For extreme examples, consider two data centers with up-time requirements of 2% and 100%. If a data center is already able to be functional 1% of the time, how much effort will it take to be available 2% of the time? In a realistic scenario, both of these numbers are incredibly low. Moving from 1% to 2% will require very little (if any) extra investment. In fact, the most simple web-server with absolutely no redundancy or error prevention/recovery mechanisms will have enough reliability to be functional far more than 2% of the time. Now, consider the data center with the 100% reliability requirement. If a data center is already able to be functional 99% of the time, how much more will it take to push it to 100%? In reality, 100% reliability is impossible to achieve. One can never guarantee that a data center will not succumb to a natural disaster, for example. So, to reach 100% reliability, we would have to invest infinite resources in a data center, even if that data center already had 99% reliability. So we see that the amount of extra resources needed to increase reliability increases as we have higher expectations. For this reason, it is possible to build two lower reliability data centers that will use fewer resources combined than a single very high-reliability data center.

We cannot, however, simply replace one data center with two. Independent data centers that are meant to serve as backups for each other present problems of their own. While we may have reduced the power consumption, we have introduced the possibility for a host of other problems. Chief among those problems are continuity and latency.

First, we look at continuity. In a data storage system where two separate locations could be called upon to store or retrieve the same data, we must make accommodations to maintain data integrity. If, for example, a piece of data is updated at one data center and then accessed from another, we must be sure that the data being accessed has been copied over from the data center to which it was written previously. Otherwise, we are producing outdated and therefore incorrect data.

Next, we consider latency. When working with any networking applications, we must consider how long it takes to communicate between point A and point B. In this case, we must consider communication between the two data centers as well as communication between the client application and one or both data centers. This element of the setup is explored in depth in this paper.

# Chapter 2

# Experimentation

In order to explore latency as may be experienced in a distributed data center system, we first must emulate such a system as realistically as possible. In order to do this, we identified Microsoft's Azure platform (http://www.windowsazure.com) as the ideal candidate. The Azure platform allowed us to set up and run a virtual server within a Microsoft data center. We were able to customize our virtual machine (VM) as if we were accessing it locally, and we were able to specify a geographical region in which our VM would be hosted. By creating servers in selected locations, then, we were able to emulate a simple distributed data center setup.

For these experiments, we set up individual VMs to act as servers and clients. All VMs used the Windows Server 2008 R2 operating system. Each server was configured using the XAMPP server control utility, Apache webserver, and MySQL database. Virtual endpoints were then configured within the Azure environment for each server to allow external web access to the server.

Next, we had to decide on an application that could accurately represent a common usage model for a data center. For this particular experiment, we decided on MediaWiki. MediaWiki is an open source application which runs on a web-server and creates and maintains a wiki-style website. MediaWiki was chosen because of the centralized nature of its backend database system as well as its standard usage model and open source nature. Because the application is meant to run and store its data on a single server, MediaWiki is a prime candidate for exploring future

distributed layout effects. And, since a wiki site is read more often than written but still written more often than a standard website, MediaWiki provides an interesting usage scenario. Lastly, the open source nature of MediaWiki means that any customizations necessary in future experimentation in order to move to a distributed layout will be significantly easier.

Finally, we had to decide on an experimentation method. An automated approach was decided upon, as it offered several large advantages over manual testing. A custom program was written using the DotNetWikiBot open source framework (http://dotnetwikibot.sourceforge.net/) to read and write to the MediaWiki server. The program was written in C# and performed a user-defined percentage of reads and writes per execution. Each read and write used the same data (a copy of a standard Wikipedia page). Each operation was also timed by the C# system timer, and reported latency to a log file in milliseconds.

Unfortunately, initial tests with this setup exhibited extremely high latencies. In the first test, we ran both the server and the client in the East US region. Averages of 2.3 seconds per read and 8.6 seconds per write were observed on a set of 5000 operations of roughly 10% write and 90% read between the client and server. These latencies were deemed too high to move forward with, and so we set out to investigate the cause. After examining Apache, MySQL, and CPU/Network utilization logs, we were still unable to find the root cause of this issue. Presently, it is our belief that PHP page generation is taking longer than it should on our server, but we have not been able to pinpoint the exact cause.

Because of this latency issue, we moved instead to a simple web service. The web service is hosted on a server and simply responds to a user request, basically providing a ping. Using this service and an automated client application that repeatedly sent requests to the server similar to the MediaWiki client, we were able to obtain meaningful data. We set up VMs to act as clients and servers in every geographical region Azure offered: East US, West US, East Asia, Southeast Asia, North Europe, and West Europe. We then ran the client service in each region so that it

communicated with a server in every other region. Via this method, we were able to obtain communication latencies for communication between all possible combinations of the six geographic locations. The results that followed were much more encouraging than our previous experiments.

# Chapter 3

## Results

The following tables summarize our results:

| | | Client | | | | | |
|---|---|---|---|---|---|---|---|
| | | East US | West US | East Asia | Southeast Asia | North Europe | West Europe |
| **Server** | East US | x | 0.064263452 | 0.211912364 | 0.241107239 | 0.086411146 | 0.085848137 |
| | West US | 0.064274343 | x | 0.162654793 | 0.190900122 | 0.15060152 | 0.146855649 |
| | East Asia | 0.212773794 | 0.160656943 | x | 0.036805098 | 0.28807951 | 0.294273283 |
| | Southeast Asia | 0.244917193 | 0.189686568 | 0.037047301 | x | 0.315662569 | 0.322831426 |
| | North Europe | 0.082241605 | 0.157337233 | 0.286787599 | 0.316095289 | x | 0.022737826 |
| | West Europe | 0.084470193 | 0.147265588 | 0.300782125 | 0.322603408 | 0.026580609 | x |
| Average Latency (seconds) | | | | | | | |

**Table 1:** Average web service latencies of geo-distributed clients in tests of 200 operations

| | | Client | | | | | |
|---|---|---|---|---|---|---|---|
| | | East US | West US | East Asia | Southeast Asia | North Europe | West Europe |
| **Server** | East US | x | 0.000737767 | 0.0069554 | 0.006168798 | 0.0073327 | 0.013208593 |
| | West US | 0.000983533 | x | 0.008657917 | 0.006995303 | 0.007201098 | 0.001249224 |
| | East Asia | 0.006659219 | 0.000932438 | x | 0.001562885 | 0.019980555 | 0.010340934 |
| | Southeast Asia | 0.015685222 | 0.002945839 | 0.010011707 | x | 0.005616916 | 0.002398691 |
| | North Europe | 0.001044801 | 0.036559848 | 0.010693646 | 0.038176722 | x | 0.006620632 |
| | West Europe | 0.001039083 | 0.001617726 | 0.012272679 | 0.001105916 | 0.031511264 | x |
| Standard Deviation (seconds) | | | | | | | |

**Table 2:** Standard Deviation of web service latencies

As can be seen in Tables 1 and 2, the resulting data of the web service experiment was telling. A client and server in relatively close geographical proximity had significantly lower latencies than the same client when communicating with a server farther away. For example, an East US client exhibited an average latency of 64ms when communicating with a West US server and a latency of 212ms when communicating with an East Asia server. All other factors are held constant, and so we can safely attribute the doubling in latency to physical distance. Table 2 shows that the standard deviation of each trial set is relatively low (about 1ms and 7ms

respectively), and so we can conclude our data is valid. Similar trends are seen throughout the tables.

In summary, our data shows us conclusively that geo-distribution of datacenters will adversely affect application latency. The low standard deviation shows that the latencies remain relatively constant over repeated trials, which signifies that applications may have very limited tolerance for latency variation. So, modifying an application to allow for more frequent data center failures and therefore latency variations may prove to be non-trivial.

# Chapter 4

## Conclusions and Further Research

Our experimentation in this paper proves that geo-location differences will affect application latency. Regardless of the usage model, if an application has to talk to a data center then it will incur a latency that is directly related to geographical distance between that application and its data center. Applying this knowledge to the proposed distributed data center model, we can begin to understand what may happen in the case of a failure.

If one data center fails, all of its traffic will be redirected to the second data center. In that case, applications must be able to first detect and implement the change in communication and then adjust for significant latency differences. Some applications may be able to do this more easily than others. If, for example, an application relies on constant data center communication then a sudden change in that communication may prove fatal. Another application, however, may only need the data center sporadically and may be able to tolerate less than ideal communication experiences.

There is much work still to be done in order to carry this idea forward. Continuing research to that end includes setting up varied applications on distributed servers and experimenting with their performance attributes. Work is currently underway to implement HDFS (Hadoop Distributed File System) across distributed servers and measure its performance. But we must also examine how these applications will respond to failures. If we are purposefully building less reliable data centers, we must anticipate failure scenarios and be prepared to handle them in our applications. Continuing research will include classifying applications by how well they are able to be adapted to a distributed system in which failure is more likely.

If we are able to prove that distributed data centers can perform just as well as a single data center, then we will be able to move the computing industry in a much more sustainable direction. With the growing popularity of mobile computing and cloud functionality, data centers are only going to become more important in the future. If we are able to make them more power efficient and sustainable now, then we will save enormous amounts of resources in the future.

**REFERENCES**

Glanz, James. "THE CLOUD FACTORIES; Power, Pollution and the Internet." *The New York Times*. The New York Times, 23 Sept. 2012. Web. 25 Sept. 2012.

# ACADEMIC VITA
# SAUL WECHT

690 Maple Hill Drive, Blue Bell PA 19422
Cell: 215-260-6160 Email: SLW5268@psu.edu

EDUCATION:

**The Pennsylvania State University** Graduation May, 2013

- Bachelor of Science Computer Engineering
- Schreyer Honors College

TECHNICAL EXPERIENCE:

**Validation Intern**
*Intel Corporation* – Hillsboro, OR 05/12 – 08/12

- Work included Xeon Phi cluster deployment, power measurement, package validation, WHQL Certification, and process automation.
- Developed power measurement scripts to measure power on an individual card in real time as well as "wall power" for entire clusters over varying time periods (written in Python).
- Implemented WHQL Certification process in its entirety. Took the process from scratch to full automation using prior internship experience (see below).

**Engineering Intern**
*Intel Corporation* – Folsom, CA 05/11 – 08/11

- Developed network applications and employee facing webpages to automate lab system maintenance
- Began the team transition from WLK 1.6 to WLK 2.0 by investigating Microsoft preview WLK releases and implementing fundamental system automation

**Engineering Intern**
*Freedom Sciences LLC* – Philadelphia, PA 05/10 – 08/10

- Developed software to analyze data generated from hundreds of ATRS wheelchair docking/transport system tests and find system flaws/weaknesses. Also developed a 3D high fidelity system simulator. Software projects completed in Python programming language.
- Also assisted in electrical engineering design/development projects including PCB design and ATRS system wiring and troubleshooting.

**FIRST Robotics** 09/05 – 05/09

- Full robot development (design, prototype, development, wiring, programming, and testing) limited to six weeks
- Programming Captain 09/06 – 05/09
  - Led a team of 10-15 student programmers in programming a 4-foot, 120 pound robot

**Tennis/Robotics Instructor**
*Sesame Rockwood Day Camps* Summers, 06-09

- Developed and ran four two week robotics programs for children ages 8-12

WORK EXPERIENCE:

**Resident Assistant**
*Penn State Residence Life* – University Park, PA 01/11 – 05/13

- Responsible for building community and enforcing Residence Life policies within the residence halls on Penn State's campus. Act as a resident's first point of contact for the University and office of Residence Life.

**5th/6th Grade Jewish Studies Teacher**
*Congregation Brit Shalom* – State College, PA 01/10 – 05/11

ACTIVITIES/ ACHIEVEMENTS:

**Penn State Aikido Club President** 01/11 – 01/13
**Penn State THON OPP Captain** 09/11 – 03/13

- Lead a team of 37 students in assisting with the planning, setup, operation, and teardown of the world's largest student-run philanthropy.

**Engineering Ambassador** 08/11 – 05/13

- Act as the student representative of Penn State's College of Engineering by giving campus tours, various presentations, and traveling to high school classrooms.