Performance Measurement and Analysis of Transactional Web Archiving

Shivam Maharshi

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Edward A. Fox, Chair
Zhiwu Xie
Dongyoon Lee

2 May 2017
Blacksburg, Virginia 24061

Keywords: Web Archiving, Digital Preservation, Performance Benchmark
Copyright 2017 Shivam Maharshi

Performance Measurement and Analysis for Transactional Web Archiving

Shivam Maharshi

(ABSTRACT)

Web archiving is necessary to retain the history of the World Wide Web and to study its evolution. It is important for the cultural heritage community. Some organizations are legally obligated to capture and archive Web content. The advent of transactional Web archiving makes the archiving process more efficient, thereby aiding organizations to archive their Web content.

This study measures and analyzes the performance of transactional Web archiving systems. To conduct a detailed analysis, we construct a meaningful design space defined by the system specifications that determine the performance of these systems. SiteStory, a state-of-the-art transactional Web archiving system, and local archiving, an alternative archiving technique, are used in this research. We experimentally evaluate the performance of these systems using the Greek version of Wikipedia deployed on dedicated hardware on a private network. Our benchmarking results show that the local archiving technique uses a Web server's resources more efficiently than SiteStory for one data point in our design space. Better performance than SiteStory in such scenarios makes our archiving solution favorable to use for transactional archiving. We also show that SiteStory does not impose any significant performance overhead on the Web server for the rest of the data points in our design space.

This work was supported in part by the Mellon Foundation through the Columbia University Web Archiving Incentive Awards, and Institute of Museum and Library Services grant LG-71-16-0037-16.

Performance Measurement and Analysis for Transactional Web Archiving

Shivam Maharshi

(GENERAL AUDIENCE ABSTRACT)

Web archiving is the process of preserving the information available on the World Wide Web into archives. This process provides historians and cultural heritage scholars access to the data that allows them to understand the evolution of the Internet and its usage. Additionally, Web archiving is also essential for some organizations that are obligated to keep the records of online resource access for their customers. Transactional Web archiving is an archiving technique where the information available on the Web is archived by capturing a transaction between a user and the Web server processing the user's request. Transactional Web archiving provides a more complete and accurate history of a Web server than the traditional Web archiving models. However, in some scenarios the transactional Web archiving solutions may impose performance issues for the Web server being archived.

In this thesis, we conduct a detailed performance analysis of SiteStory, a state-of-the-art transactional Web archiving solution, in various experimental settings. Furthermore, we propose a novel transactional Web archiving approach and compare its performance with SiteStory. To conduct a realistic study, we analyze real-life traffic on Greek Wikipedia website and generate similar traffic to perform our benchmarking experiments. Our benchmarking results show that our archiving technique uses a Web server's resources more efficiently than SiteStory in some scenarios. Better performance than SiteStory in such scenarios makes our archiving solution favorable to use for transactional archiving. We also show that SiteStory does not impose any significant performance overhead on the Web server in other scenarios.

This work was supported in part by the Mellon Foundation through the Columbia University Web Archiving Incentive Awards, and Institute of Museum and Library Services grant LG-71-16-0037-16.

Dedication

I dedicate this thesis to my parents, Gopesh and Saroj, my sister Shivangi, and my brother Himanshu for their continuous support, inspiration, and sacrifices. They have always motivated me to be the best version of myself. This journey would not have been possible without them.

Acknowledgments

I acknowledge and thank some of the people who played a very important role in my Masters thesis. Without these people, my journey through graduate school would not have been so full of learning and excitement. I am deeply grateful to all these people and owe them a debt of gratitude.

First and foremost, I would like to acknowledge my advisors: Dr. Edward Fox, Dr. Zhiwu Xie, and Dr. Dongyoon Lee, for their continuous help and tremendous efforts throughout my graduate education. I cannot even conceive the notion of successfully conducting this research without their guidance. Dr. Fox has always been a conscientious and a very supportive mentor. His great expertise in research helped me to formulate a strategy and organize my research work very efficiently. His calm disposition and the ability to comprehend problems quickly made it easy for me to approach him with any questions and problems, which at times seemed too naive. His punctuality and well organized schedule motivated me to push myself harder to finish work in a timely fashion. Finally, his excellent command over English and writing skills helped me to improve the quality of this thesis immensely.

Dr. Xie has always been a very honest and frank guide to me. His critical thinking pushed me to dive into great technical depth in the search for an answer to any inconsistencies present in the results. Learning this critical reasoning made me a better researcher and is the best learning experience of my graduate school. His excellent technical knowledge and constant guidance helped me to debug and solve technical problems at a faster pace. His attention to details and precision helped me to eliminate all the causes for inconsistencies, and to generate reproducible results.

Dr. Lee has been a very kind and supportive advisor throughout my graduate studies. Very easy to approach, Dr. Lee always made himself available for quick meetings and discussions. His valuable feedback on the experiments, presentation, and the thesis, helped me to improve the quality of this research.

Apart from my advisors, I am also thankful to all my colleagues from the Digital Library Research Laboratory and the technical staff of University Libraries at Virginia Tech. Sunshin Lee, Prashant Chandrasekar, and Paul Mather helped me to debug technical issues that occurred during the research environment setup as well as provided constructive feedback on my presentations at DLRL. Ronald Mecham, Tingting Jiang, and Liuqing Li coordinated

and helped me to acquire the resources that were necessary to conduct this research. Rob Hunter from the technical support staff of the Department of Computer Science at Virginia Tech helped me with the CentOS installation issues on old proprietary Apple hardware.

In addition to people from Virginia Tech, I am also thankful to Kevin Risden, Andy Kruth, and Sean Busbey from the Yahoo Cloud Serving Benchmark development community. They reviewed my code multiple times and provided valuable feedback. Kevin Risden showed extra diligence and reviewed my work repeatedly over a period of two months. This finally lead to the acceptance of my first open source contribution in the official YCSB repository on GitHub. This accomplishment would not have been possible without these helping hands.

Finally, I am very grateful for the continuous support and motivation that I received from my family and friends.

This work was supported in part by the Mellon Foundation through the Columbia University Web Archiving Incentive Awards, and Institute of Museum and Library Services grant LG-71-16-0037-16.

Contents

1 Introduction **xviii**

1.1 Objective and Overview xviii
1.2 Web Archiving xviii
1.2.1 Traditional Web Archiving xix
1.2.2 Transactional Web Archiving xix
1.3 SiteStory xx
1.4 Yahoo Cloud Serving Benchmark xx
1.5 Modes xx
1.6 Design Space xx
1.6.1 Fast Computation Fast Network xxii
1.6.2 Fast Computation Slow Network xxii
1.6.3 Slow Computation Fast Network xxii
1.6.4 Slow Computation Slow Network xxii
1.7 Hypothesis xxiii
1.8 Research Questions xxiii
1.9 Contributions xxiii
1.10 Thesis Roadmap xxiv

2 Related Work **xxv**

2.1 Related Work xxv
2.2 SiteStory xxvi

2.2.1 Architecture Overview xxvii
2.2.2 SiteStory Module xxviii
2.2.3 SiteStory Web Archive xxviii
2.3 Yahoo Cloud Serving Benchmark xxviii
2.3.1 Benefits xxix
2.3.2 Design & Architecture xxix

3 Local Transactional Archiving **xxxii**
3.1 SiteStory Performance Concerns xxxii
3.2 Local Transactional Archiving xxxiii
3.2.1 Local Archiving with Persistent Files xxxv
3.2.2 Local Archiving using RAM as Store xxxvi

4 Data **xxxviii**
4.1 Introduction xxxviii
4.2 Greek Wikipedia xxxviii
4.2.1 Why Use Greek Wikipedia xxxviii
4.2.2 Setup xxxix
4.3 Benchmarking Trace xxxix
4.3.1 Why Use Benchmarking Trace xxxix
4.3.2 Preparing Greek Wikipedia Benchmarking Trace xl
4.3.3 Zipfian Distribution xli
4.3.4 Read Trace xliii
4.3.5 Write Trace xlv
4.3.6 Write Size Trace xlvi
4.3.7 Trace Cleaning xlvi

5 Benchmarking **xlviii**
5.1 Introduction xlviii
5.2 Benchmarking Practices xlviii

5.2.1 Incremental Workload xlviii
5.2.2 Sufficient Duration xlix
5.2.3 Consistent System Configuration xlix
5.2.4 Bottleneck Identification l
5.2.5 Multiple Iterations l
5.2.6 Resource Monitoring li
5.3 YCSB REST Workload and REST Module li
5.3.1 Architecture li
5.3.2 REST Workload lii
5.3.3 REST Module liv
5.3.4 REST Configurations liv

6 Experimental Design **lviii**
6.1 Introduction lviii
6.2 Experimental Setup lviii
6.2.1 YCSB Client lviii
6.2.2 Apache Web Server lix
6.2.3 SiteStory Server lx
6.2.4 Ethernet Switch lxi
6.3 Modes lxi
6.3.1 Mode A lxii
6.3.2 Mode B lxiii
6.3.3 Mode C lxiv

7 Results **lxv**
7.1 Introduction lxv
7.2 Fast Computation Fast Network lxv
7.2.1 Benchmarking Results lxvi
7.2.2 Performance Evaluation lxx

7.3 Fast Computation Slow Network lxx
7.3.1 Benchmarking Results lxxi
7.3.2 Performance Evaluation lxxv
7.4 Slow Computation Fast Network lxxv
7.4.1 Benchmarking Results lxxv
7.4.2 Performance Evaluation lxxix
7.5 Slow Computation Slow Network lxxix
7.5.1 Benchmarking Results lxxix
7.5.2 Performance Evaluation lxxxii

8 Conclusions **lxxxiii**
8.1 Conclusion lxxxiii
8.2 Future Work lxxxiv

Bibliography **lxxxiv**

A Results for Additional Modes **lxxxix**
A.1 Running SiteStory Locally - Mode D lxxxix
A.1.1 Fast Computation Fast Network xc
A.1.2 Fast Computation Slow Network xciii
A.1.3 Slow Computation Fast Network xcvii
A.1.4 Slow Computation Slow Network c
A.2 Using RAM as a Store Local Archiving Technique - Mode E ciii
A.2.1 Fast Computation Fast Network civ
A.2.2 Fast Computation Slow Network cvii
A.2.3 Slow Computation Fast Network cx
A.2.4 Slow Computation Slow Network cxiii

B Yahoo Cloud Service Benchmark **cxvii**
B.1 REST Workload cxvii

B.2 REST Client cxxiv
B.3 Integration Test cxxxii
B.4 REST Client Test cxxxvii
B.5 REST Test Resource cxli
B.6 Resource Loader cxlii
B.7 Utils cxliii

C Local Archiving and Monitoring **cxlv**

C.1 Local Archiving Service Using Persistent Files cxlv
C.2 Local Archiving Service Using RAM as Store cxlvi
C.3 Monitor Resources Script cxlvii
C.4 Logstash Startup Script cxlvii
C.5 Logstash Visualization Configuration cxlviii
C.6 Logstash Dashboard Configuration cli

List of Figures

1.1 Design Space xxi

2.1 Architecture of SiteStory [1] xxvii

3.1 Architecture of Local Archiving with Persistent Files xxxv

3.2 Architecture of Local Archiving using RAM as Store xxxvi

4.1 Log(Read Counts) versus Log(Page Rank) for all endpoints accessed in November 2015 xlii

4.2 Log(Write Counts) versus Log(Page Rank) for all endpoints accessed in November 2015 xliii

4.3 Log(Read Counts) versus Log(Page Rank) for top 10K URLs xliv

4.4 Log(Read Counts) versus Log(Page Rank) for top 20K URLs xliv

4.5 Log(Write Counts) versus Log(Page Rank) for top 5K URLs xlv

4.6 Log(Write Counts) versus Log(Page Rank) for top 10K URLs xlvi

5.1 YCSB REST Module Architecture lii

5.2 YCSB REST Workload liii

6.1 Architecture of Mode A lxii

6.2 Architecture of Mode B lxiii

6.3 Architecture of Mode C lxiv

7.1 Average Latency and Errors versus RPS for Fast Computation Fast Network data point lxvi

7.2 Average Latency versus RPS for Fast Computation Fast Network data point lxvi

7.3 CPU Usage for Fast Computation Fast Network data point lxvii
7.4 Network Write Bandwidth Usage for Fast Computation Fast Network data point lxvii
7.5 Network Read Bandwidth Usage for Fast Computation Fast Network data point lxviii
7.6 Disk Read Bandwidth Usage for Fast Computation Fast Network data point lxix
7.7 Disk Write Bandwidth Usage for Fast Computation Fast Network data point lxix
7.8 Average Latency versus Request Per Second for Fast Computation Slow Network data point lxxi
7.9 Average Latency versus RPS for Fast Computation Slow Network data point lxxi
7.10 CPU Usage for Fast Computation Slow Network data point lxxii
7.11 Network Write Bandwidth Usage for Fast Computation Slow Network data point lxxiii
7.12 Network Read Bandwidth Usage for Fast Computation Slow Network data point lxxiii
7.13 Disk Read Bandwidth Usage for Fast Computation Slow Network data point lxxiv
7.14 Disk Write Bandwidth Usage for Fast Computation Slow Network data point lxxiv
7.15 Average Latency versus Request Per Second for Slow Computation Fast Network data point lxxv
7.16 CPU Usage for Slow Computation Fast Network data point lxxvi
7.17 Network Write Bandwidth Usage for Slow Computation Fast Network data point lxxvi
7.18 Network Read Bandwidth Usage for Slow Computation Fast Network data point lxxvii
7.19 Disk Read Bandwidth Usage for Slow Computation Fast Network data point lxxviii
7.20 Disk Write Bandwidth Usage for Slow Computation Fast Network data point lxxviii
7.21 Average Latency versus Request Per Second for Slow Computation Slow Network data point lxxix
7.22 CPU Usage for Slow Computation Slow Network data point lxxx
7.23 Network Write Bandwidth Usage for Slow Computation Slow Network data point lxxx
7.24 Network Read Bandwidth Usage for Slow Computation Slow Network data point lxxxi

7.25 Disk Read Bandwidth Usage for Slow Computation Slow Network data point lxxxi
7.26 Disk Write Bandwidth Usage for Slow Computation Slow Network data point lxxxii

A.1 Average Latency and Errors versus RPS for Fast Computation Fast Network data point xc
A.2 CPU Usage for Fast Computation Fast Network data point xc
A.3 RAM Usage for Fast Computation Fast Network data point xci
A.4 Network Write Bandwidth Usage for Fast Computation Fast Network data point xci
A.5 Network Read Bandwidth Usage for Fast Computation Fast Network data point xcii
A.6 Disk Read Bandwidth Usage for Fast Computation Fast Network data point xcii
A.7 Disk Write Bandwidth Usage for Fast Computation Fast Network data point xciii
A.8 Average Latency and Errors versus RPS for Fast Computation Slow Network data point xciii
A.9 CPU Usage for Fast Computation Slow Network data point xciv
A.10 RAM Usage for Fast Computation Slow Network data point xciv
A.11 Network Write Bandwidth Usage for Fast Computation Slow Network data point xcv
A.12 Network Read Bandwidth Usage for Fast Computation Slow Network data point xcv
A.13 Disk Read Bandwidth Usage for Fast Computation Slow Network data point xcvi
A.14 Disk Write Bandwidth Usage for Fast Computation Slow Network data point xcvi
A.15 Average Latency and Errors versus RPS for Slow Computation Fast Network data point xcvii
A.16 CPU Usage for Slow Computation Fast Network data point xcvii
A.17 RAM Usage for Slow Computation Fast Network data point xcviii
A.18 Network Write Bandwidth Usage for Slow Computation Fast Network data point xcviii
A.19 Network Read Bandwidth Usage for Slow Computation Fast Network data point xcix
A.20 Disk Read Bandwidth Usage for Slow Computation Fast Network data point xcix

A.21 Disk Write Bandwidth Usage for Slow Computation Fast Network data point c
A.22 Average Latency and Errors versus RPS for Slow Computation Slow Network data point c
A.23 CPU Usage for Slow Computation Slow Network data point ci
A.24 RAM Usage for Slow Computation Slow Network data point ci
A.25 Network Write Bandwidth Usage for Slow Computation Slow Network data point cii
A.26 Network Read Bandwidth Usage for Slow Computation Slow Network data point cii
A.27 Disk Read Bandwidth Usage for Slow Computation Slow Network data point ciii
A.28 Disk Write Bandwidth Usage for Slow Computation Slow Network data point ciii
A.29 Average Latency and Errors versus RPS for Fast Computation Fast Network data point civ
A.30 CPU Usage for Fast Computation Fast Network data point cv
A.31 Network Write Bandwidth Usage for Fast Computation Fast Network data point cv
A.32 Network Read Bandwidth Usage for Fast Computation Fast Network data point cvi
A.33 Disk Read Bandwidth Usage for Fast Computation Fast Network data point cvi
A.34 Disk Write Bandwidth Usage for Fast Computation Fast Network data point cvii
A.35 Average Latency and Errors versus RPS for Fast Computation Slow Network data point cvii
A.36 CPU Usage for Fast Computation Slow Network data point cviii
A.37 Network Write Bandwidth Usage for Fast Computation Slow Network data point cviii
A.38 Network Read Bandwidth Usage for Fast Computation Slow Network data point cix
A.39 Disk Read Bandwidth Usage for Fast Computation Slow Network data point cix
A.40 Disk Write Bandwidth Usage for Fast Computation Slow Network data point cx
A.41 Average Latency and Errors versus RPS for Slow Computation Fast Network data point cx
A.42 CPU Usage for Slow Computation Fast Network data point cxi

A.43 Network Write Bandwidth Usage for Slow Computation Fast Network data point cxi

A.44 Network Read Bandwidth Usage for Slow Computation Fast Network data point cxii

A.45 Disk Read Bandwidth Usage for Slow Computation Fast Network data point cxii

A.46 Disk Write Bandwidth Usage for Slow Computation Fast Network data point cxiii

A.47 Average Latency and Errors versus RPS for Slow Computation Slow Network data point cxiii

A.48 CPU Usage for Slow Computation Slow Network data point cxiv

A.49 Network Write Bandwidth Usage for Slow Computation Slow Network data point cxiv

A.50 Network Read Bandwidth Usage for Slow Computation Slow Network data point cxv

A.51 Disk Read Bandwidth Usage for Slow Computation Slow Network data point cxv

A.52 Disk Write Bandwidth Usage for Slow Computation Slow Network data point cxvi

List of Tables

6.1 System specification of YCSB Client lix

6.2 System specification of the low computational capacity Web server lx

6.3 System specification of the high computational capacity Web server lx

6.4 System specification of the SiteStory server lx

6.5 Specification of the low network bandwidth switch lxi

6.6 Specification of the high network bandwidth switch lxi

Chapter 1

Introduction

1.1 Objective and Overview

The objective of this thesis is to analyze the performance and identify the performance concerns of SiteStory, on four data points in a design space represented by the computation power and network bandwidth of a Web server. We further propose an alternative transactional Web archiving solution to address these performance concerns.

1.2 Web Archiving

Web archiving is the process of preserving in an archive the information contained on websites in the World Wide Web [2]. One of the key issues for Internet scholars of the future is to access the Web of the past [3], because even though the Web is ubiquitous, websites are transitory. Studies found that the average lifespan of Web pages ranges from 44 days to 75 or 100 [4]. Even though the number of initiatives in Web archiving grew significantly after 2003, they are concentrated in developed countries [5]. *"Web archives, all together, must manage more data than any Web search engine"* [5]. Considering the complexity, the fast rate of content change, and the large amounts of data involved in Web archiving, the available resources are scarce [5].

More and more heritage institutions are engaging in Web archiving. A survey by the Research Library Group (RLG) in 2006 showed that 60 percent of their members considered that Web archiving was part of their mission [2]. Given the rising importance of Web archiving coupled with the scarcity of available resources, there is a need for more research into novel Web archiving techniques that efficiently solve these problems in handling large datasets.

1.2.1 Traditional Web Archiving

The most widely used technique for Web preservation is Web harvesting, which means that a Web crawler downloads a file from a Web server by following links from a given starting URL [3]. Traditional archiving has a few drawbacks. To begin with, it is a time intensive process that requires Web crawlers to visit the Web graph and archive information one page at a time. This time is limited by crawling restrictions and the hit rate set by websites on the maximum pages that can be crawled. Also, it is a resource intensive process that requires dedicated computing facilities just for crawling Web pages. In addition, the crawling frequency of Web harvesting is not aligned with the rate of change of a server's resources. Hence all versions of a server's resources are not captured by Web harvesting. This results in an archive that may have an acceptable overview of a server's resources history, but does not provide a complete representation of changes. Hence, in the modern world where much data is generated on a daily basis and updated frequently, Web harvesting does not result in the comprehensive Web archives that researchers and historians desire.

1.2.2 Transactional Web Archiving

Transactional archiving, as the name suggests, is the process of capturing the transactions that take place between a Web server and client. It is an event-driven approach, since the event of fetching a Web server's resource initiates its archiving process. Transactional archiving provides a more comprehensive history of a server's resources than does traditional Web harvesting because every actual client-server transaction is recorded [4]. Also, the transactional archiving model emphasizes more the important data, because it archives the frequently requested data more often. This is because every time data is requested from the Web server, an archiving process is initiated to archive the requested data. Hence, there is an increase in the archiving frequency for resources that are accessed more often. However, a limitation of transactional Web archiving models is that they require coordination between the archiving scholars and the data owners, which are mainly organizations. Consequently, it is generally implemented by content owners or hosts rather than external collecting organizations [4].

In this thesis, we discuss in detail about SiteStory, a state-of-the-art transactional Web archiving tool [1]. We verify the claims made by the SiteStory authors, who state that "SiteStory does not significantly affect content server performance when it is performing transactional archiving" [6].

1.3 SiteStory

SiteStory is a state-of-the-art transactional Web archiving tool that captures Web resources as they are served by a Web server. A typical example of such a scenario is an end user requesting a Web page from a Web browser. SiteStory was developed at the Las Alamos National Laboratory's Research Library. We elaborate more on SiteStory and its architecture in Chapter 2.

1.4 Yahoo Cloud Serving Benchmark

To analyze the performance of SiteStory, we needed a benchmarking tool capable of generating real-life workloads. For this purpose we used Yahoo Cloud Serving Benchmark (YCSB) [7], which is an open source framework for benchmarking the performance of different key-value stores and databases. It was developed in 2010 by Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears at the Yahoo Research Labs. We discuss YCSB, its architecture, and our contribution to it in detail in Chapter 2.

1.5 Modes

To quantify our performance concerns about SiteStory, we conduct our benchmarking experiments on three different experimental setups. We refer to these different experimental setups as benchmarking modes: Mode A, Mode B, and Mode C. Performing our experiments in three benchmarking modes helped us to quantify SiteStory's performance, and to compare it with an alternative transactional Web archiving solution. We provide an in-depth explanation of the benchmarking modes and their architecture in Chapter 5.

1.6 Design Space

A design space for a system is a space that includes all possible designs that exist for the system. A design space can be represented as a hypothetical N-dimensional space. In such a representation, every dimension of the design space represents an independent variable that affects the design of the system. A data point in a design space is any point that can be uniquely identified by a set of values for all of the independent variables of this design space. A design space can have an infinite numbers of data points.

The design space for transactional Web archiving systems can be represented by many independent variables like CPU, RAM, disk bandwidth, network bandwidth, etc. However, due to the limitations of the computing resources available to us, we had control to vary only

the CPU, RAM and network bandwidth. Hence we chose to represent our design space with two variables: computational capacity and network bandwidth. Computational capacity for a system, the first variable in our design space, is determined by its CPU specifications and RAM size. Network bandwidth for a system, the second variable in our design space, is determined by the bandwidth of the network that connects this system. Hence, as shown in Figure 1.1, our design space can be represented on a two dimensional space with network bandwidth as its X axis and computational capacity (CPU and RAM) as its Y axis.

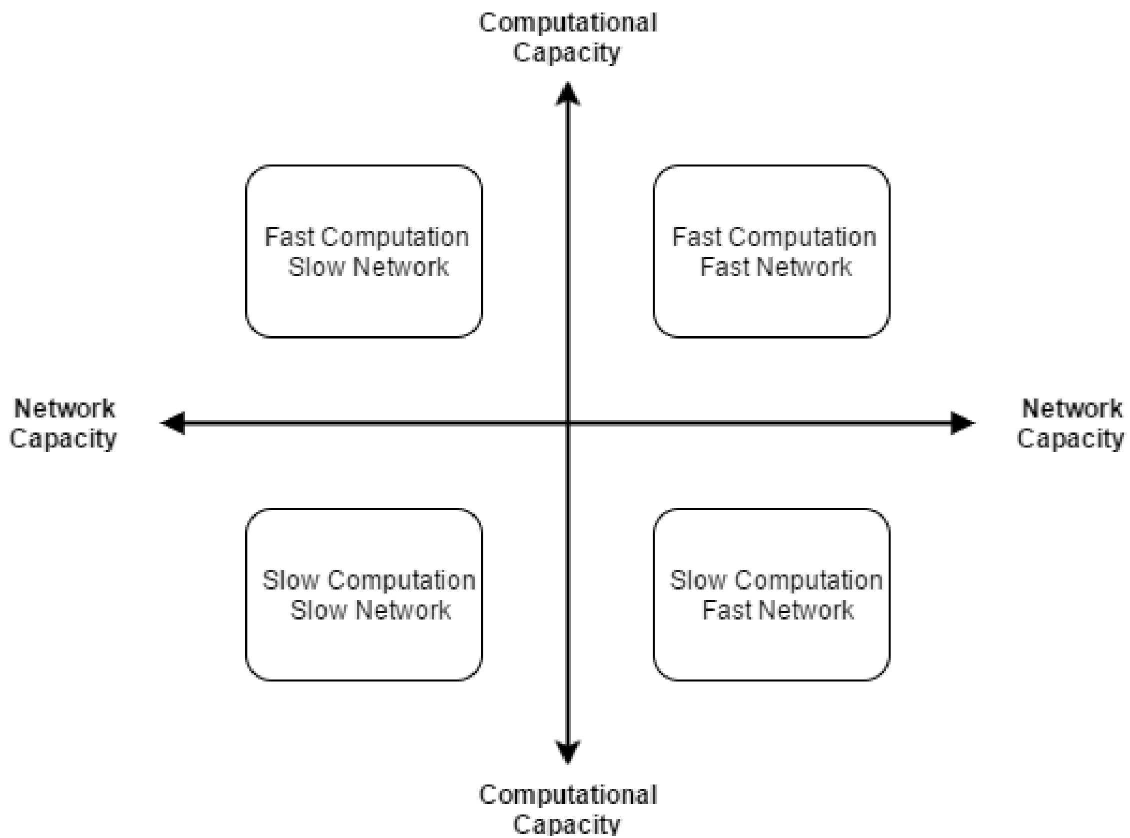

Figure 1.1: Design Space

In this thesis, we focus on four data points in our design space to compare the performance of the two transactional Web archiving solutions. These four data points are: Fast Computation Fast Network, Fast Computation Slow Network, Slow Computation Fast Network, and Slow Computation Slow Network. The data points in our design space were chosen on the basis of the hardware resources available to conduct our performance analysis. All these data points differ only in the computational and network capacities of the server involved in the archival process.

1.6.1 Fast Computation Fast Network

This data point in our design space represents a scenario where both the computation and network capacity for the content Web server are high. An example of this data point is using a c3.8xlarge type of Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instance as the Web server on a 10 Gbps network available for high performance computing using placement groups, because both the computational and network capacity of such a system is high. This data point corresponds to the first quadrant of our design space representation shown in Figure 1.1.

1.6.2 Fast Computation Slow Network

This data point in our design space represents a scenario where the content Web server has a high computational capacity but relatively low network capacity. An example of this data point is using a t2.2xlarge type of AWS EC2 instance as the Web server on a 100 Mbps network because the computational capacity of such a system is comparatively higher than its network capacity. This data point corresponds to the second quadrant of our design space representation shown in Figure 1.1.

1.6.3 Slow Computation Fast Network

This data point in our design space represents a scenario where the content Web server has a high network capacity but relatively lower computational capacity. An example of this data point is using a t2.micro type of AWS EC2 instance as the Web server on a 100 Mbps network because the computational capacity for such systems is lower relative to its network capacity. This data point corresponds to the fourth quadrant of our design space representation shown in Figure 1.1.

1.6.4 Slow Computation Slow Network

This data point in our design space represents a scenario where the computation and network capacity for the content Web server are both low. An example of this data point is using a t2.medium type of AWS EC2 instance as the Web server on a 100 Mbps network because both the computational and network capacity of such a system are low. This data point corresponds to the third quadrant of our design space representation shown in Figure 1.1.

1.7 Hypothesis

In this thesis we claim that there exist alternative transactional Web archiving architectures that perform better than SiteStory in some situations by using system's resources more efficiently. An in-depth explanation of our reasoning behind this claim is provided in Chapter 2.

1.8 Research Questions

A few major research questions that we ask in this thesis are:

1. What is the design space that allows us to understand the performance of transactional Web archiving systems?
2. What are the important metrics to quantify the performance of a transactional Web archiving system?
3. What are the performance concerns of SiteStory in our design space, as quantified by our performance metrics?
4. What alternative solutions can be devised to address these performance concerns?
5. What performance overheads do these solutions impose on the content Web server?
6. Which Web transactional archiving solution (SiteStory or local archiving) uses computing resources more efficiently on the four data points in our design space?

1.9 Contributions

Through this research we make four major contributions to the field of transactional Web archiving and Web service benchmarking. First, we summarize Web server benchmarking practices, such as creating realistic workloads, and use them to demonstrate how to conduct fair benchmarking. Second, we verify the performance claims made by the authors of SiteStory about its minimal usage of server resources. Third, we evaluate the performance of two Web archival systems, SiteStory and our local archiving, across four data points in our design space, which helps us to understand the performance of these systems in the four key regions of our design space. In the end, we present an alternative local archiving solution to SiteStory that performs similarly, but uses roughly half of the network bandwidth. This makes our local archiving technique a more suitable approach for the Fast Computation Slow Network data point in our design space.

1.10 Thesis Roadmap

The rest of the thesis is organized as follows. First, Chapter 2 explains related research work, SiteStory, YCSB, and our design space. Then, Chapter 3 presents the architecture and implementation of our local transactional Web archiving solution. Chapter 4 talks about the dataset used in our experiments and methodologies involved in preparing the benchmarking traces. We also show that the resulting benchmark is an accurate representation of a realistic workload. Then, Chapter 5 outlines the benchmarking practices and demonstrates how we follow them. Additionally, we describe our open source contribution to the YCSB. Afterwards, Chapter 6 expounds on our experimental design, as well as the different modes of operations in our experiments, and also the system configurations in our design space. Chapter 7 then presents the benchmarking results obtained from the different operational modes across the system configurations within our design space. Using these results we delve into a detailed comparison and analysis of SiteStory, along with our local archiving solution. Next, Chapter 8 concludes the thesis by summarizing the takeaways from our results and our proposal for improving our design in the future. At the end, our Appendix constitutes of three chapters – A, B, C. Appendix A describes the additional modes – Mode D and Mode E – and exhibits their benchmarking results. Appendix B shows the Java code that we developed and contributed to the YCSB framework as the REST module. Finally, Appendix C shows the the PHP implementation of our local Web archiving solutions, resource monitoring and visualization tool startup shell scripts, and Logstash configuration files. A Zip archive has been submitted with this thesis that includes all figures, results data, and code files which are described in a readme metadata file.

Chapter 2

Related Work

In this chapter we discuss the related work done in the field of transactional Web archiving, which is of interest to our thesis. First of all, we talk about the related research work and publications on transactional Web archiving. Then, we dive deep into the explanation of SiteStory, a state-of-the-art transactional Web archiving tool. Following this, we will elaborate on the YCSB, an open source benchmarking tool. Finally, we provide a detailed explanation of the design space in the context of this thesis.

2.1 Related Work

The research community in the field of transactional Web archive has been increasingly active in recent years. A variety of research papers, ranging from a novel transaction Web archive implementation to a modified archiving approach for real-time Web archiving, have been published since 2013.

The paper"Evaluating the SiteStory Transactional Web Archive With the ApacheBench Tool" [6] elaborates the performance evaluation of SiteStory using the Apache benchmarking tool with synthetic workloads. Our research is closely related with their paper since both of these works aim at evaluating the performance of SiteStory. However, their research does not evaluate the performance of SiteStory on a realistic Web application with a benchmarking trace that represents a production type workload. Our study addresses this concern by evaluating SiteStory's performance on MediaWiki, which is widely used at production level by many organizations, with a workload generated using real-life benchmarking traces. Additionally, the three experiments in their study were performed on the same experimental setup. This limits the generality of the performance claims made in the paper, since the findings might not apply to other experimental settings with different specifications. For example, different computing power or network bandwidth availability might yield different results. Our study addresses this concern since we conducted all of our experiments in four different settings

where we varied the CPU, RAM, and network bandwidth of our Apache Web server under benchmarking. These additional data points in our design space gives us the scope of a more comprehensive study to analyze the performance of SiteStory.

"Using Transactional Web Archives To Handle Server Errors" [8] is an interesting study that proposes UWS, a novel redirection Apache module that provides uninterrupted access to Web content by serving them from SiteStory. When the content Web server fails due to some internal server fault, each incoming request is redirected to be handled by the SiteStory server. This research demonstrates an excellent application of SiteStory to improve the end user experience. Another interesting work, "Nearline Web Archiving" [9] proposes a modified approach to real-time transactional Web archiving by leveraging Web caching infrastructure already prevalent on Web servers. This research aims at archiving data without incurring any performance overhead on the Web server. Our study makes a different contribution from these research investigations since we analyze the performance of two transactional Web archiving solutions in a broader design space.

2.2 SiteStory

SiteStory is a transactional Web archiving tool capable of capturing every version of a Web resource as it is being requested by a Web browser. It supports the following features:

1. Dynamic archiving of an Apache Web server.
2. Archived data accessible via the Memento protocol [10].
3. Archived data offloaded to WARC files [11].
4. Archived data uploaded into an instance of the Internet Archive's Wayback Machine [12].

2.2.1 Architecture Overview

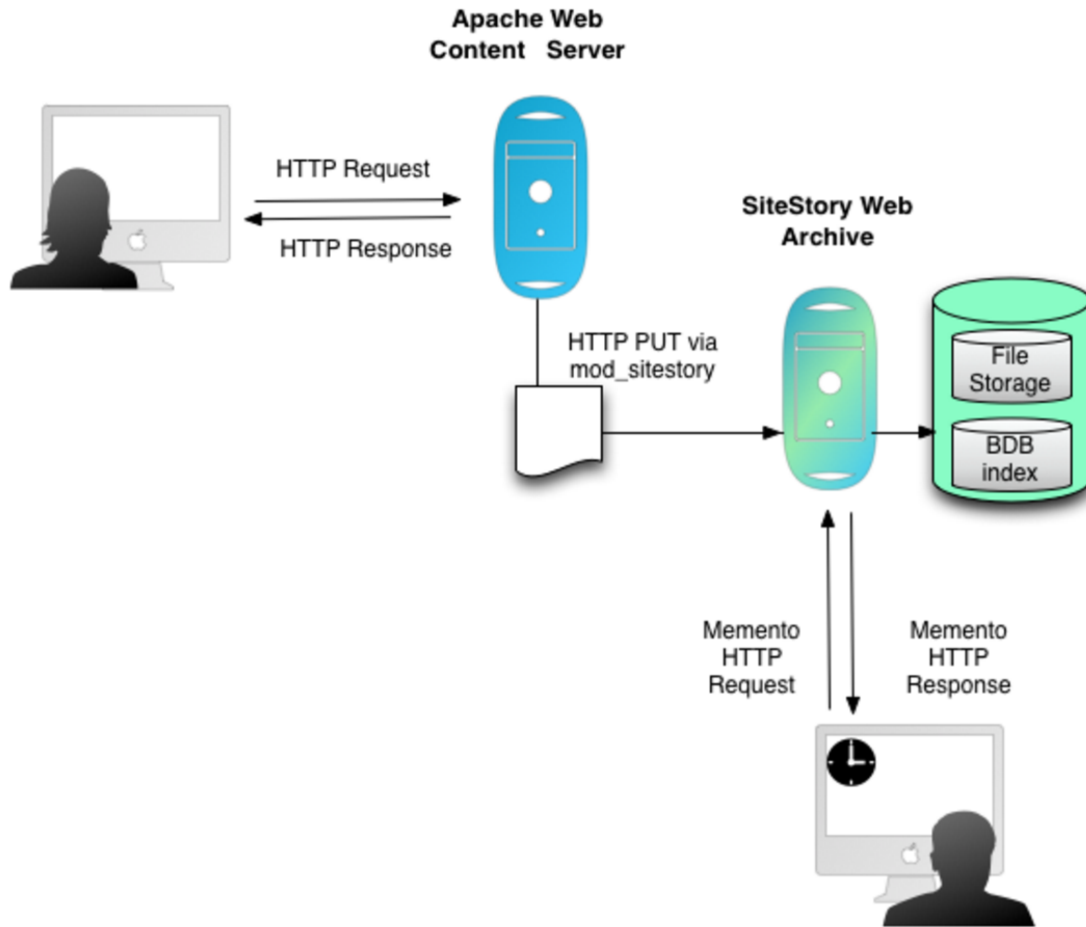

Figure 2.1: Architecture of SiteStory [1]

Figure 2.1 shows an architectural overview diagram of SiteStory and all the important SiteStory components along with a Web archiving request flow. The whole process can be broken down into a series of steps, mentioned below.

An end user requests a Web page from the Apache Web server through his Web browser or Web application. The user's request for data generates an HTTP GET request dispatched to the Apache Web server. The aforementioned server, on receiving the HTTP GET request, processes it to prepare an HTTP response, and returns it back to the end user. In parallel with dispatching the response back to the end user, an installed SiteStory Apache module (mod_sitestory) asynchronously captures this response. The module sends an HTTP PUT request with the captured response as its request body to another SiteStory Web archive server. Upon receiving an archiving HTTP PUT request, the SiteStory Web archive server

store its body. Depending on the user configuration, the archiving request is stored on a file system as an archive file using a Berkeley database (DB) [13] index. Finally, the SiteStory Web archive server contains all the successfully archived Web responses of the resources on the Apache Web server. The SiteStory Web archive server supports Memento, an HTTP based protocol, to access all of the archived responses.

2.2.2 SiteStory Module

The HTTP response captured by the SiteStory module is sent for archiving from the Apache Web server to the SiteStory Web archive server by a special installed filter called mod_sitestory. We refer to this filter as the SiteStory module throughout this thesis. This module is implemented in C and can be installed using the standard Apache apxs [14] program on Linux systems.

2.2.3 SiteStory Web Archive

The SiteStory Web archive is a Java based Web application that is responsible for accepting an HTTP PUT archiving request from the Apache Web server and storing it using Berkeley DB indexes as archive files on the file system. It also implements the HTTP based Memento protocol to provide access to all of the archived Web resources. SiteStory can easily be deployed on many standard Web servers that support Java Web application deployment. We have used Apache Tomcat [15] as the deployment server for SiteStory to conduct our experiments.

2.3 Yahoo Cloud Serving Benchmark

The Yahoo Cloud Serving Benchmark (YCSB) [7] is an open source framework for benchmarking the performance of different key-value stores and databases. It is comprised of three components: YCSB core, module, and module configuration file. The YCSB core[1] is an extensible workload generator capable of generating a variety of workloads required to simulate different benchmarking scenarios. The YCSB module is an extensible request generator responsible for generating workload requests specific to a particular key-value store like Redis [16] or databases like Apache Cassandra [17] or MySQL [18]. The module configuration file defines various input parameters required by the YCSB module. In the subsections below we discuss in detail about the YCSB framework benefits and architecture.

1. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/workloads/CoreWorkload.java

2.3.1 Benefits

Below are the advantages that YCSB has over the other famous benchmarking tools:

1. YCSB is open source and free to use under the Apache License 2.0.
2. It is implemented in Java, which is a platform independent language, hence it is supported on all operating systems like CentOS, Mac OS X, Ubuntu, and Windows.
3. It is capable of generating large workloads to benchmark vast systems.
4. It is more customizable when compared to other popular benchmarking tools like Apache JMeter [19], Apache Benchmark [20], Httperf [21], Tsung [22], HttpLoad[2], etc. It has configurable benchmark properties like: thread concurrency; create, read, update, and delete (CRUD) operation ratio; distribution type; distribution parameters (like Zipf's constant); read and execution timeouts; console logging; number of operations to be performed; maximum time limit for benchmarking; etc.
5. It can be easily modified or extended to implement more features.
6. It is a mature benchmarking framework with over 45 releases and is actively maintained.

2.3.2 Design & Architecture

The YCSB is easily customizable because of its modular design. Its design leads to a separation of concerns across different components, which communicate with each other via interfaces. The three major components of YCSB are discussed below in detail.

YCSB Core

The YCSB core is the first component that is invoked during the benchmarking process. It is responsible for the following tasks:

1. The YCSB core includes the YCSB client, which takes inputs from a user, forks multiple threads, maintains the execution counters, and starts the benchmarking.
2. The YCSB core includes a set of extensible generators – like uniform generator, Zipfian generator, etc. – which produce numbers based on various distribution types required for generating workload.

2. https://github.com/timbunce/http_load

3. The YCSB core includes a set of extensible measurement entities, which keep track of the various benchmarking metrics like average latency, maximum latency, minimum latency, throughput, n^{th} percentile, etc.

4. The YCSB core includes various extensible exporters, which store and preserve the benchmarking results in data representations like JavaScript Object Notation (JSON) [23], simple text, HyperText Markup Language (HTML) [24], etc.

5. The YCSB core has extensible core and REST workloads, which populate every benchmarking thread with a standard YCSB request object using various data generators. The standard YCSB request object contains the benchmarking request information generated at the YCSB core to transfer it to the YCSB module. All the workloads in YCSB must implement the Workload interface[3].

YCSB Module

The YCSB module is invoked by the YCSB core during the benchmarking process. It takes the standard YCSB request object, generated by the core, and makes a service-specific request from it. Two key features of a YCSB module are:

1. A module (or binding) is specific to a service like a database, key-value store, or Web service. Thus, the accumulo module[4] benchmarks only the Apache Accumulo database [25], and the cassandra module[5] benchmarks only the Apache Cassandra database.

2. Each module must implement the DB interface[6] which acts as a standard contract of communication between the YCSB workload generator and the module.

YCSB Module Configuration File

The YCSB workloads parse the YCSB module configuration file to populate various configurable parameters, which are essential for the benchmarking process. The configuration file includes three type of properties: the core properties, workload properties, and custom properties. Some core properties are the workload type, database binding, exporter type, and measurement type. Some workload properties are the CRUD request proportion, request distribution, field lengths, operation counts, and maximum execution time. The core and workload properties are never specific to a particular module, which distinguishes them

3. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/Workload.java
4. https://github.com/brianfrankcooper/YCSB/tree/master/accumulo
5. https://github.com/brianfrankcooper/YCSB/tree/master/cassandra
6. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/DB.java

from the custom properties. All the module-specific properties defined in this configuration file are loaded in the module init method and are referred to as custom properties.

Chapter 3

Local Transactional Archiving

In this chapter we present our theoretical performance concerns with SiteStory and the possible alternative solutions that could address these concerns.

3.1 SiteStory Performance Concerns

This section deals with the performance impacts of SiteStory on the Apache Web server being archived. The task of analyzing the performance overheads incurred by SiteStory and identifying the places at which they occur, is vital to design more efficient alternative archiving solutions. In order to quantify the performance, and hence the performance overhead incurred by the Apache Web server, we use average latency, throughput, and number of HTTP 500 errors received, as our performance metrics [26]. Latency is defined as the absolute time difference between the moment a request is sent out to the Web server from our YCSB benchmarking client to the time when the corresponding response is received by it. Lower average latency corresponds to better Web server performance. Throughput is defined as the number of requests processed successfully by the Apache Web server per second. This is the same as the number of successful responses received by the YCSB benchmarking client per second. A higher throughput value corresponds to better Web server performance. The number of errors is defined as the number of responses received by the YCSB client for which the response status code was HTTP 5XX. These responses correspond to the requests that either failed at the server side or took more than ten seconds to respond.

After carefully analyzing the work flow of a transactional Web archiving request of SiteStory, we expect performance overheads at the two places mentioned below:

1. The SiteStory module installed in the Apache Web server captures every response and sends it to the SiteStory Web archive server as an HTTP request. This whole process consumes processing power of the CPU to perform a series of tasks. Hence, we expect

that this archiving process of the SiteStory module will steal enough cycles so as to be noticeable, when the Apache Web server processes are under heavy workload and bounded by available CPU capacity. If such a situation arises, then we predict that there would be a noticeable degradation in the performance of the Apache Web server while the SiteStory module is enabled on it.

2. To avoid performance degradation of the Apache Web server, SiteStory authors recommend setting up the SiteStory archiving server on a separate machine. In such a setting, the SiteStory module sends every response over the network via TCP/IP to the SiteStory Web archiving server. Thus apart from the HTTP response for every HTTP request to the Apache Web server, an additional HTTP PUT request is sent out from the SiteStory module to the SiteStory Web archiving server. Since this HTTP PUT request contains the response of the HTTP request to the Apache Web server, the network bandwidth used with SiteStory enabled is nearly twice as much as the bandwidth used without it. This additional network traffic could degrade the performance of the Apache Web server by creating a bottleneck in the networking layer. This is extremely likely to happen in scenarios where the average network bandwidth used by the Web server is more than 50 percent of the maximum available bandwidth. Hence, in these scenarios we expect some degradation in the performance of the Apache Web server.

3.2 Local Transactional Archiving

In this section, we discuss two solutions to address the potential performance degradation concerns that we stated in the previous section. In order to improve the performance of the transactional archiving process using SiteStory, we broke the archiving process into two parts:

1. The first part includes archiving the HTTP responses of the Apache Web server locally on the same machine. This eliminates the excessive use of network resources since every archiving response gets sent to the machine itself (through the loop back interface). Hence, such a local archiving solution has half the network bandwidth requirements as for a typical SiteStory setting.

2. The second part includes the transferring and merging of the locally archived Web resources with the already indexed resources on the SiteStory Web archive server. This transferring and merging could be configured to occur at regular intervals like daily, weekly, or monthly, depending on the use case. Since this transferring needs to happen only once in a fixed amount of time, it reduces the redundant usage of the network bandwidth that occurs in a typical SiteStory setting. This is because on websites some resources are accessed repetitively in a short duration of time. In a typical SiteStory setting, this generates extra network traffic, in the form of SiteStory Web archiving

requests, for every request received by the Apache Web server. However, after being archived once at the first archiving request, all the subsequent archiving requests for the same resources will be discarded by SiteStory because of being duplicate. Hence a large portion of the network bandwidth would be effectively wasted by transferring duplicate resources over the network multiple times. However, using a local archiving solution will avoid this redundant network bandwidth usage by transferring the locally archived data over the network only once in a fixed amount of time.

Additionally, a user could configure this transfer process to occur only when the network bandwidth is available for use. This gives a user control over the network bandwidth usage by the Web archiving process, which allows him to efficiently distribute the network bandwidth at his discretion to improve the overall efficiency of the system. Finally, using the local archiving solution would require a separate SiteStory Web archive server only at the time of transferring and merging. Hence it eliminates the continuous need of another server to run a SiteStory archive server that just receives the HTTP PUT requests from the Apache Web server. Such a computing resource could be used for other computational tasks, thereby improving the overall CPU usage of the system.

Due to the aforementioned reasons, we predict that the network bandwidth usage of a Web archiving system can be improved by using a local transactional archiving solution. We now present two possible implementations of the local transactional archiving technique mentioned above. All the local transactional archiving solutions mentioned below involve running a light weight local archiving Web service on the Apache Web server for two reasons:

1. A Web service is required to process the archiving requests sent by the SiteStory module since they are HTTP PUT requests.
2. A light weight Web service would consume minimal CPU and therefore will not impose much CPU overhead on the Apache Web server running on the same machine. This would help us to avoid losing any performance benefits that we gain by efficiently redistributing the network bandwidth of our system with the local archiving solution.

All of the implementations presented below differ only in the way the archiving requests are processed, once they are received by the local archiving Web service.

3.2.1 Local Archiving with Persistent Files

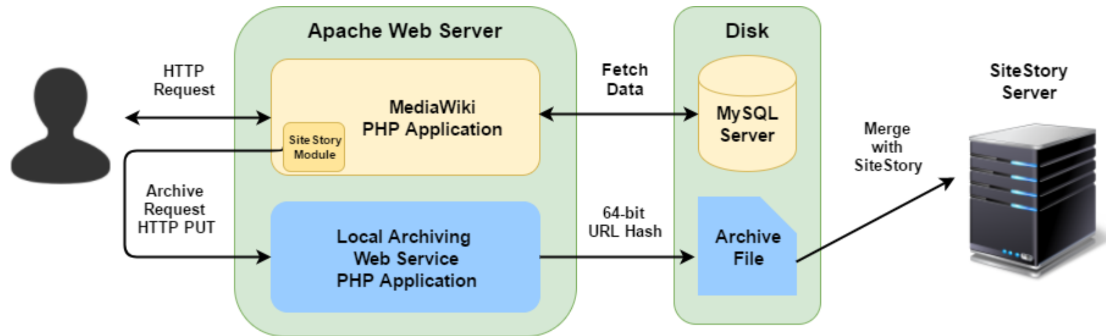

Figure 3.1: Architecture of Local Archiving with Persistent Files

Figure 3.1 shows the architecture of our local archiving with persistent files solution. Here, we process the incoming requests by performing the following steps. First, a 64-bit MD5 hash of the URL for which the archiving request has arrived is calculated. Then a file with the same name as this hash is checked for existence in a predefined folder. If such a file is not present in this folder, then the HTTP PUT request body is read from the archiving request. In such a case, a new file with the URL hash value as its name and HTTP PUT request body as its content is created in the predefined folder. If the file is already present, it means that the HTTP response for this URL is already archived and the service returns an HTTP 200 OK status.

The local archiving with persistent files solution is not the same as running the SiteStory Web archive service locally because the SiteStory Web archiving service needs to be deployed on the Apache Tomcat server. Running an Apache Tomcat server on the same machine would steal CPU cycles from the Apache Web server at heavy workloads thereby degrading its performance. In Appendix A.1 we present the benchmarking results of running SiteStory Web archive service locally on the content Web server and show that it indeed imposes a significant overhead on the Apache Web server. Our local archiving solution, on the other hand, runs a very light weight Web service without the need of an additional Web server, which distinguishes it from running the SiteStory Web archive server locally.

3.2.2 Local Archiving using RAM as Store

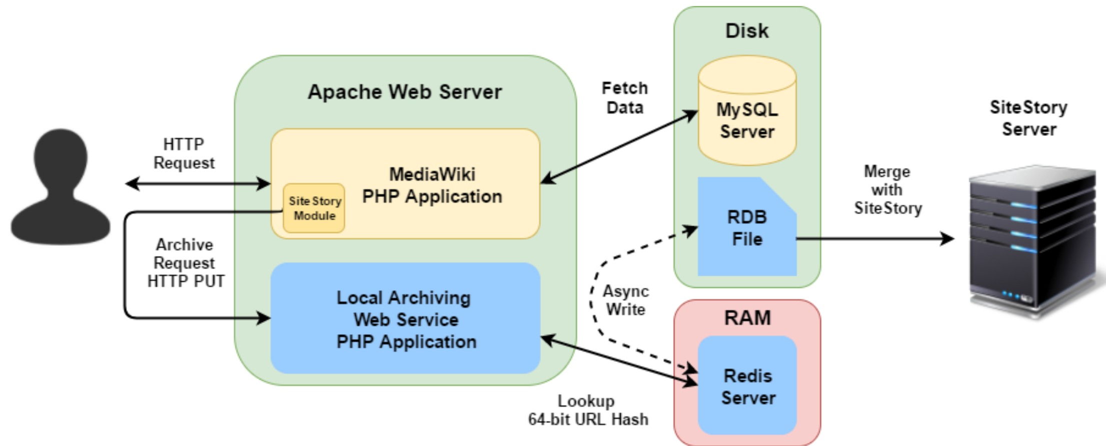

Figure 3.2: Architecture of Local Archiving using RAM as Store

Figure 3.2 shows the architecture of our local archiving service using RAM as store (i.e., in a Redis server). Here, the following steps are performed on an incoming archiving request. First, the archiving request is read and the last modified time-stamp of the HTTP response being archived is extracted from it. Then, a composite key, which is a unique combination of the URL endpoint being archived and this time-stamp value, is formed. Together, the key and the URL endpoint give a unique combination to accurately identify the different versions of same Web resources. Finally, if this key is already present in the Redis server, then a 200 OK HTTP status code is returned. However, if this key is not present in the Redis server, then it is stored in Redis with the archiving HTTP request body as its value. This value includes the HTTP request header, HTTP response header, and the HTTP response body for the URL endpoint being archived. We rely on Redis's automatic persistence to save all the data on the disk in the form of RDB files[1] RDB files can be transferred and read using RDB tools [27] and finally, merged with the SiteStory archive server.

This mode has two benefits over the previous mode. First, the created unique key uses both the URL endpoint to be archived and the last modified time stamp. Hence it will always distinguish between two different versions of the same URL resource. However, the local archiving solution using files, only stores one version of a URL endpoint till the archive folder is cleaned for transfer and merged with the SiteStory server. In our opinion, this is not a major issue since we propose to transfer the data every day, which typically is more frequent that the frequency of change of the Web resources [4]. Second, writing to RAM is faster than writing to disk. Hence this solution seems like a good choice for better performance. However, the overhead of running a Redis server locally can actually degrade the Apache Web server's performance rather than improve it.

1. https://github.com/sripathikrishnan/redis-rdb-tools

We chose to implement and evaluate only the local archiving with persistent files solution, since in our preliminary experiments (Appendix A.2) it performed better than the Local Archiving using RAM as Store solution.

Chapter 4

Data

4.1 Introduction

In this chapter we describe our test data and the process of preparing our benchmarking traces. The choice of test data and accurate benchmarking traces is important to perform fair and insightful benchmarking. To reproduce real-life scenarios that closely represent a production-like environment, we choose to set up a Greek Wikipedia [28] mirror as our Web content server for the performance evaluation of SiteStory and our local archiving solution.

4.2 Greek Wikipedia

4.2.1 Why Use Greek Wikipedia

Using Greek Wikipedia for benchmarking has a few advantages. First, Wikipedia is a widely used Web application that serves millions of requests each day. Hence, the choice of this Web application gives our benchmarking results more credibility, since it is production quality software. Second, the Wikipedia statistics website [29] makes available for public use the necessary information to create benchmarking traces, in the form of page access statistics. This information allowed us to generate the read and write traces to represent the real-life workload on a Greek Wikipedia mirror. Third, Wikipedia runs on MediaWiki [30], open source software written in PHP [31] that can be deployed on the Apache Web server, which makes it possible for us to set up a Greek Wikipedia mirror free of cost. Fourth, since MediaWiki is open source, much documentation is readily available on the Internet, which makes it easier to troubleshoot issues. Finally, since Mediawiki is free software, our setup can be replicated without any costs and our results can easily be reproduced by anyone who wishes to verify them.

At the start of our research, we considered using the English Wikipedia [32] instead of the Greek Wikipedia for our benchmarking. However, we discovered that English Wikipedia is 10 terabytes in size (without images). Importing such an enormous data dump into the database takes a long time with the available MediaWiki data dump import tools like MWDumper [33] and ImportDump [34]. On the other hand, the Greek Wikipedia dump that we chose is 260 megabytes in size and takes only three hours to import into the database. Hence we decided to use the Greek Wikipedia for our benchmarking.

4.2.2 Setup

To set up the Greek Wikipedia mirror, we need to install MediaWiki, a database, and a Web server. We chose MySQL Server 5.0.2 as our database, Apache Web Server 2.3.1 as our Web server, and MediaWiki 1.26.2. We downloaded the full data dump of Greek Wikipedia, for only the current revisions, from their official website [35]. This data dump holds over 180,000 articles [36] and has been edited 6,035,830 times. To import this data into the MySQL database we used MWDumper, the officially recommended tool written and maintained by the development team of Wikipedia.

4.3 Benchmarking Trace

A benchmarking trace is a collection of resources (HTTP URL endpoints in our case) along with a few other workload properties that determine the behavior of a benchmarking workload. It defines a set of resources to be benchmarked, the access pattern of these resources, different operations allowed on them, and the proportion of such operations. A benchmarking tool uses a trace as an input to generate a benchmarking workload.

4.3.1 Why Use Benchmarking Trace

A benchmarking tool user gets control over the type of workload to be generated by defining a benchmarking trace. Hence the user can define a benchmarking trace to generate a workload that closely resembles the real-life access pattern of the system being benchmarked. Such a trace can be created by collecting and analyzing the real-life access statistics for the system. Using such a workload leads to realistic benchmarking, which yields results that are applicable to real-world scenarios. The more realistic a trace, the more realistic will be the benchmarking results. Hence, using accurate benchmarking traces helps in conducting a realistic and relevant benchmarking study.

4.3.2 Preparing Greek Wikipedia Benchmarking Trace

Benchmarking traces for a Web service are composed of HTTP URL endpoints to be used during the benchmarking process. The workload properties should include the ratio of the different HTTP operations, request distributions of these operations, and the parameters defining these distributions. Hence to prepare our benchmarking traces we had to find out the HTTP URL endpoints where these operations took place and the number of times they took place on Greek Wikipedia in some fixed time. We decided to analyze one month of access data for trace preparation as it gives a good sample of the workload behavior and amortizes any erratic access behaviors that might occur just on a few days.

Preparing Read Trace

A read trace file holds a set of HTTP URL endpoints to be benchmarked with the HTTP GET request. To create our read trace and figure out its request distribution, we collected the URL access information of the Greek Wikipedia for the month of November 2015 from the Wikipedia statistics website. The Wikipedia statistics website provides this information every day, in 24 separate page view statistics files (on an hourly basis). Each page view file stores a list of all the HTTP GET requests, separated by a newline, that were received by the Greek Wikipedia during the hour that this file corresponds to. Accordingly, we downloaded 720 files for November 2015, and aggregated their data into a single frequency distribution file (using AnalyzeMonthlyTrace[1]). This file contained a list of unique URL endpoints and the total number of times they were accessed in a month. Then, we sorted this file in descending order of the number of times the URL endpoints were accessed. Once we prepared a frequency distribution file, we ran a linear regression on it to find the request distribution followed by the Greek Wikipedia HTTP GET operations. Performing this analysis helped us to calculate: the set of requested URL endpoints, number of times they were accessed, and the distribution they followed; these we needed to prepare our read benchmarking trace.

Preparing Write Trace

A write trace file holds a set of HTTP URL endpoints to be benchmarked with the HTTP POST request. The Wikipedia statistics website does not provide information for the HTTP POST operations like it does for the HTTP GET operations, in the form of page view files. Hence we set up an additional Greek Wikipedia, along with a corresponding revision history, to generate a write trace. We created a get-monthly-write-trace SQL script[2] to fetch the list of all the URL endpoints for which write operations took place on the Greek Wikipedia during the month of November 2015. Then, we used this list to prepare a frequency distribution file,

1. https://github.com/VTUL/mw-benchmark/blob/master/src/main/java/org/vt/edu/analysis/Analyze-MonthlyTrace.java
2. https://github.com/VTUL/mw-benchmark/blob/master/sql/get-monthly-write-trace.sql

that contained all the unique URL endpoints and the number of times that they received an HTTP POST request. Afterwards, we sorted this file in descending order of the number of times the URL endpoints received an HTTP POST request. Just like for the read trace file, we ran a linear regression to find the request distribution that is followed by Greek Wikipedia HTTP POST operations.

Preparing Write Size Trace

A write size trace file holds a set of HTTP POST request payload sizes to be used for populating an HTTP POST benchmarking request. We created the get-monthly-write-size SQL script[3] to fetch the list of HTTP POST write sizes that took place on the Greek Wikipedia during the month of November 2015. Then, we used this list to prepare a frequency distribution file, that contained all the unique HTTP POST write sizes and the number of times that an HTTP POST request was made with this write size. Afterwards, we sorted this file in descending order of the occurrence frequency of the write data size. Finally, we ran a linear regression to find the request distribution that is followed by Greek Wikipedia HTTP POST operations write sizes.

4.3.3 Zipfian Distribution

The Zipfian distribution belongs to the family of discrete power law probability distributions, which states that the probability of an event is inversely proportional to its rank in the series of such events. While discussing about the distribution pattern of the HTTP requests on the Greek Wikipedia, it is relevant to mention the Zipfian distribution for two reasons. First of all, many studies have shown that most of the website access patterns on the Internet follow a Zipfian distribution [37] [38]. Additionally, a Wikipedia statistics article states that the URL access pattern on Wikipedia does follow a Zipfian distribution [39]. Hence we decided to verify if our read and write traces indeed do follow a Zipfian distribution. To accomplish this task, we used the following mathematical equation that describe the Zipfian distribution:

$$p_i = Cr_i^{-\beta}$$

Here p_i is the occurrence probability of the event at rank r_i and β is the exponent in Zipf's law. By taking the log on both the sides, the equation above can be written as:

$$log(p_i) = log(C) - \beta log(r_i)$$

This equation represents a line in two dimensions, with negative β as its slope and log(C) as its Y axis intercept.

3. https://github.com/VTUL/mw-benchmark/blob/master/sql/get-monthly-write-size.sql

In a Zipfian distribution where an event is described as accessing an HTTP URL endpoint, the occurrence of probability of this event corresponds to the number of times this URL endpoint is accessed (for HTTP GET or HTTP POST method). The rank of such an event in this distribution corresponds to the rank of this URL endpoint in its frequency distribution file. Hence, to find the values of β and verify whether our read and write traces follow the Zipfian distribution, we ran a linear regression on the log of frequency of the URL endpoints versus the log of its rank. As shown in Figures 4.1 and 4.2, the coefficients of determination (R^2) for the read and write traces were 0.9682 and 0.9529, respectively. The closer the value of R^2 is to one, the better is the fit of the data to the distribution used in the regression. Since we received a value of R^2 that is very close to one for both the read and write traces, we verified that our Greek Wikipedia does indeed follow a Zipfian distribution for both reads and writes.

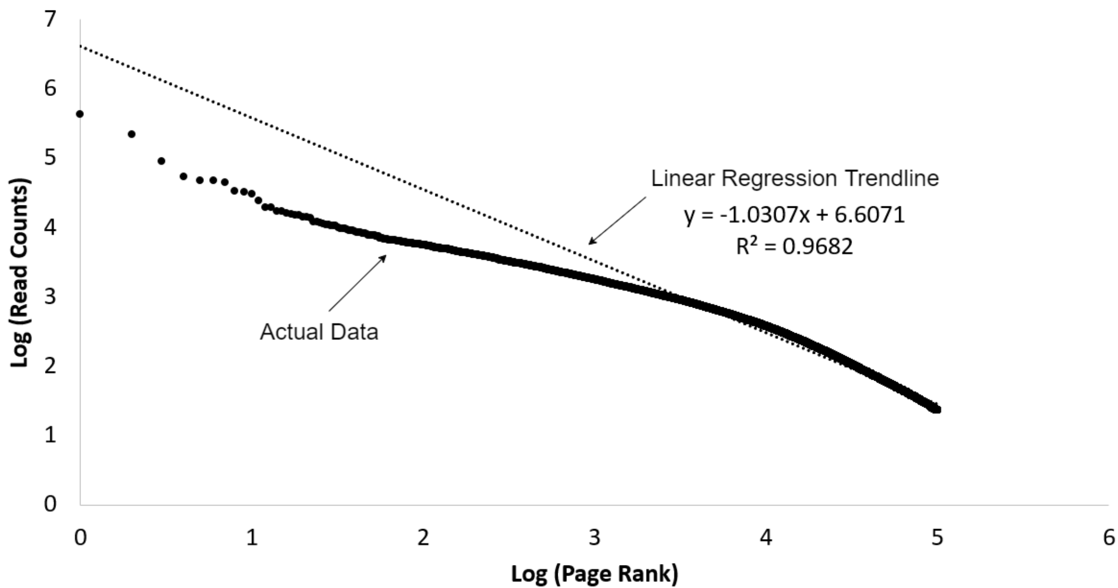

Figure 4.1: Log(Read Counts) versus Log(Page Rank) for all endpoints accessed in November 2015

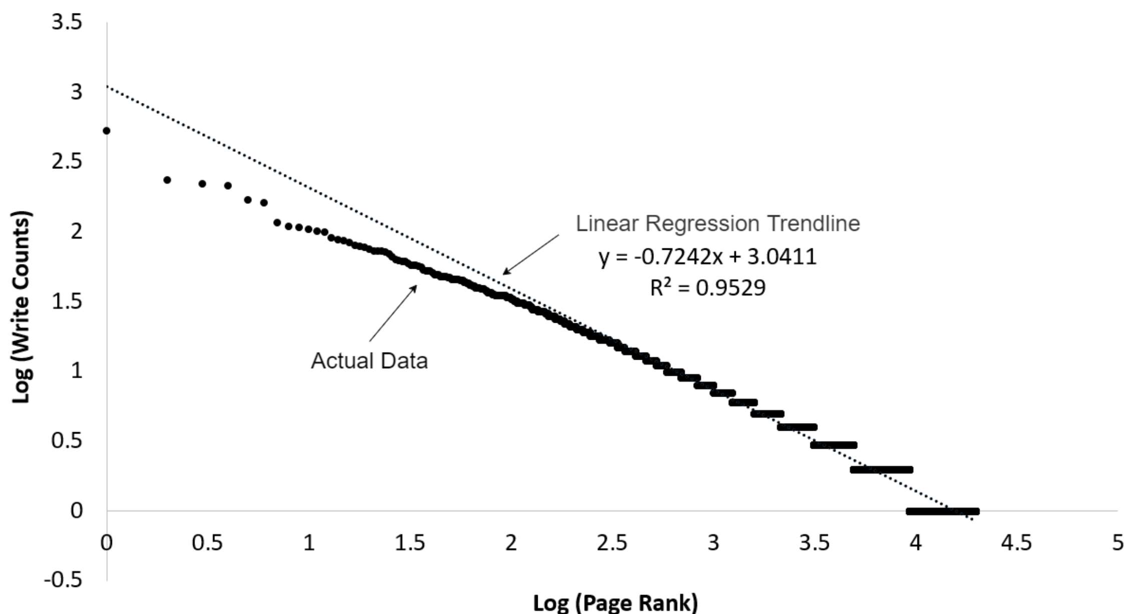

Figure 4.2: Log(Write Counts) versus Log(Page Rank) for all endpoints accessed in November 2015

4.3.4 Read Trace

For the month of November 2015, a total of 2,867,552,286 read operations took place on the Greek Wikipedia. The read trace contained 1,060,114 unique URL endpoints. We calculated the R^2 for the top 10,000 and 20,000 URL endpoints. These came out to be 0.9898 and 0.9772, respectively. Figures 4.3 and 4.4 show the log-log plot for the URL rank versus the URL hit count. Since a higher value of R^2 signifies a better fit, we chose to use the top 10,000 URL endpoints as our final read trace for benchmarking. To verify that the top 10,000 Wikipedia pages have similar page size as the rest of the Wikipedia pages, we calculated the average page size for these top 10,000 URL endpoints and compared it with the value suggested by the Wikipedia community [40]. To calculate the average page size, we captured the amount of network traffic generated by our Wikipedia mirror when these top 10,000 URL endpoints were accessed at the rate of 400 pages per second for ten minutes. We observed that our Wikipedia mirror generated a network traffic of 23,000 KB per second while serving 400 pages per second. This suggested that the average size of a Wikipedia page from the top 10,000 URL of our read trace was 57.5 KB, which is very close to the suggested page size of 50 KB by the Wikipedia community. For the top 10,000 URL endpoints we calculated the value of β to be 0.6175. Our value of β is close to 0.5, which is the value mentioned in a Wikipedia article [39] that states the calculated β for read access pattern on Wikipedia. The

read trace for the top 10,000 URL endpoints, named readtrace.txt[4], is available on GitHub with unrestricted read access.

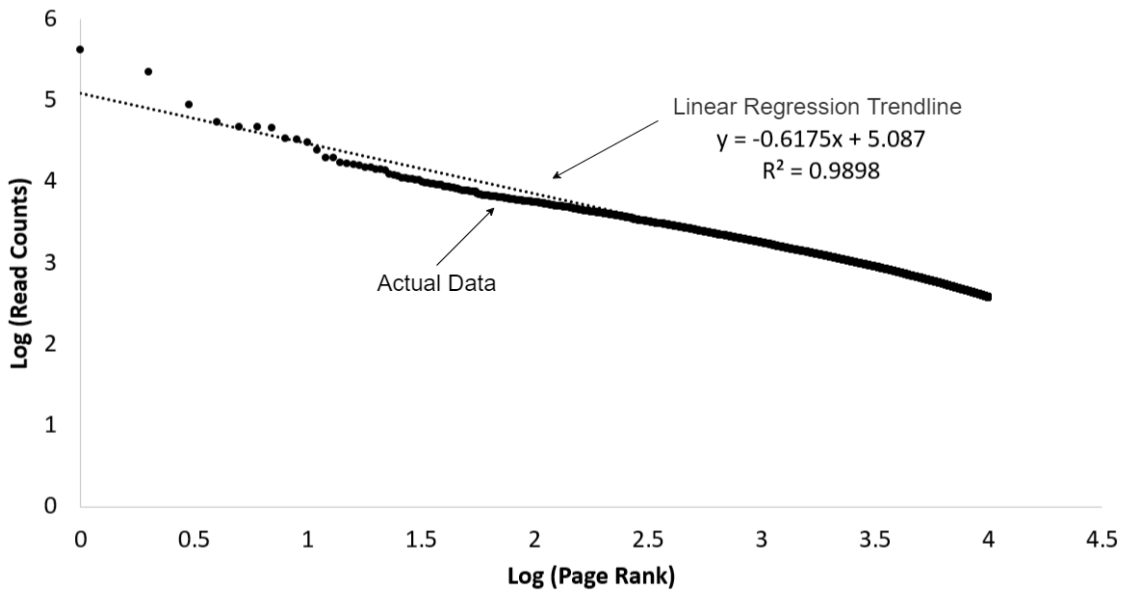

Figure 4.3: Log(Read Counts) versus Log(Page Rank) for top 10K URLs

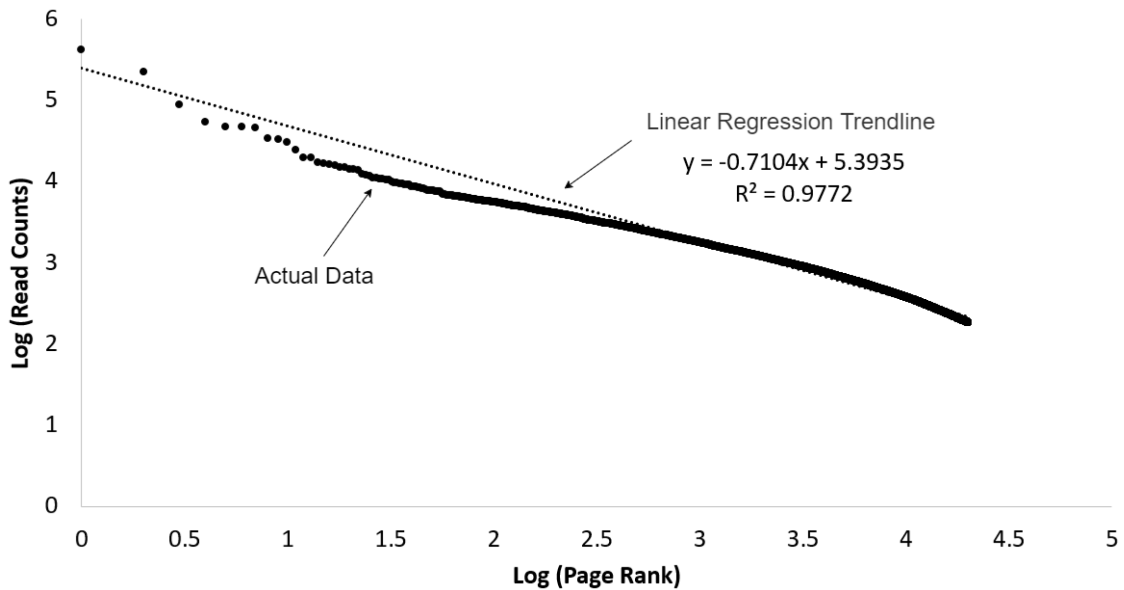

Figure 4.4: Log(Read Counts) versus Log(Page Rank) for top 20K URLs

4. https://github.com/VTUL/mw-benchmark/blob/master/input/readtrace.txt

4.3.5 Write Trace

For the month of November a total of 58,857 write operations took place on the Greek Wikipedia as opposed to 2,867,552,286 read operations. This was expected since Wikipedia is a read heavy website. The write trace contains 24,955 unique URL endpoints. We calculated the R^2 for the top 5,000 and 10,000 URL endpoints. These came out to be 0.9863 and 0.9507, respectively. Figures 4.5 and 4.6 show the log-log plot for the URL rank versus the URL hit count. Since the highest value of R^2 was for the top 5,000 URL endpoints, we chose to use them in our final write trace. For the top 5,000 URL endpoints, we calculated the value of β to be 0.6391. The full write trace, named writetrace.txt[5], is available on GitHub with unrestricted read access.

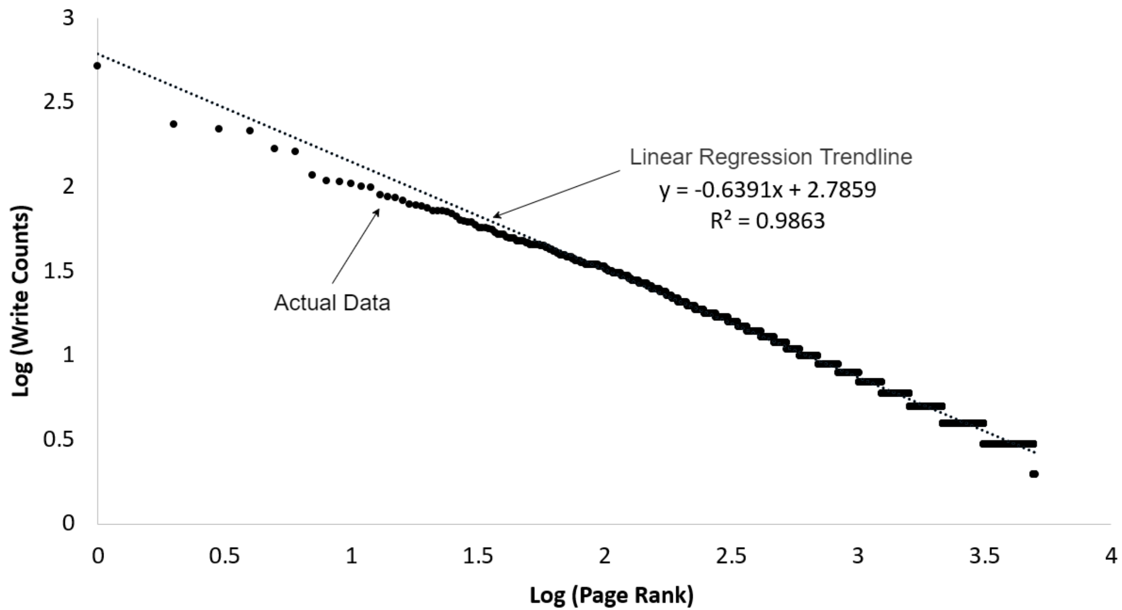

Figure 4.5: Log(Write Counts) versus Log(Page Rank) for top 5K URLs

5. https://github.com/VTUL/mw-benchmark/blob/master/input/writetrace.txt

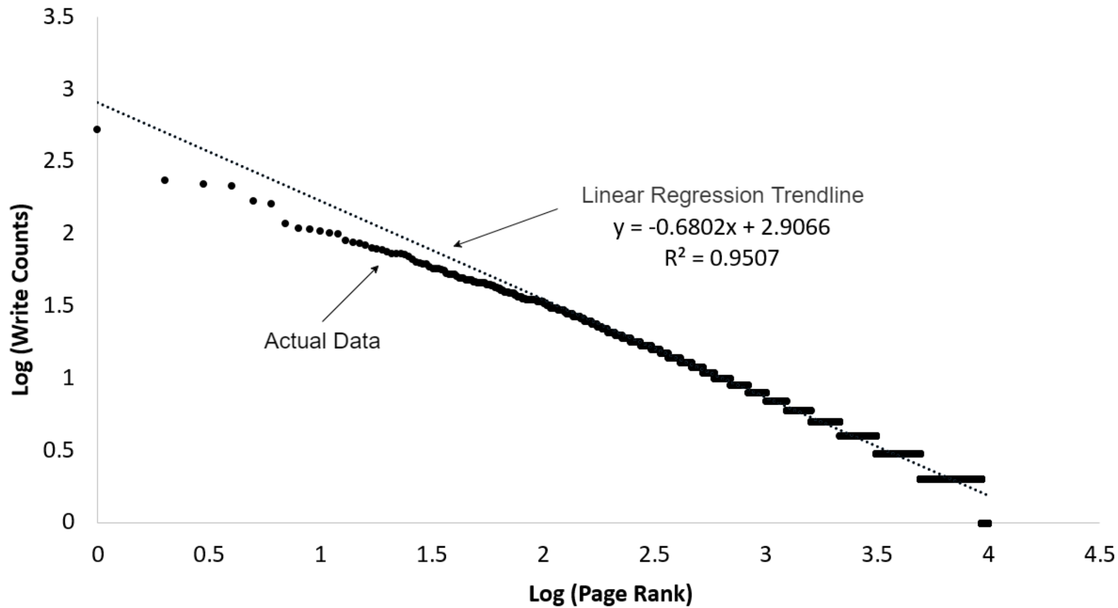

Figure 4.6: Log(Write Counts) versus Log(Page Rank) for top 10K URLs

4.3.6 Write Size Trace

Our analysis showed that the revision size trace, unlike the read or write trace, follows a Uniform distribution [41] instead of a Zipfian distribution. YCSB reads this file and randomly selects values from it for the HTTP POST requests. It creates a dummy write body of the chosen size to prepare an HTTP POST request for benchmarking the Greek Wikipedia. The full write size trace, named writesizetracex.txt[6], is available on GitHub with unrestricted read access.

4.3.7 Trace Cleaning

We verified all the URL endpoints in our read trace by generating HTTP GET requests on them against our Wikipedia mirror. By categorizing the HTTP response status code that we received from their response, we discovered 14 URL endpoints in our read trace that pointed to non-existent articles on the official Wikipedia website. These URL endpoints, absent from Greek Wikipedia, were still present in our read trace because many users requested such URLs on the official Greek Wikipedia website. That is why these URL endpoints were present in the Wikipedia access data and so were included in our read trace file. The list of

6. https://github.com/VTUL/mw-benchmark/blob/master/input/writesizetrace.txt

all these URL endpoints, named removedurls.txt[7], is available on GitHub with unrestricted read access.

7. https://github.com/VTUL/mw-benchmark/blob/master/analysis/removedurls.txt

Chapter 5

Benchmarking

5.1 Introduction

This chapter elaborates on the three aspects of benchmarking that we conduct in this research. They include the benchmarking practices we followed, our open source contribution to the YCSB framework, and the benchmarking specifications that we used to perform our experiments.

5.2 Benchmarking Practices

Following benchmarking practices helps to accurately compare two or more different systems and produce reliable results. Benchmarking experiments that ignore these practices often lead to erroneous, unpredictable, and inconsistent results that do not represent the performance under real-life workloads. In this section, we mention some benchmarking practices that we follow to perform a meaningful benchmarking.

5.2.1 Incremental Workload

Using an incremental workload is helpful to perform an accurate benchmarking study. An incremental workload starts the benchmarking by generating a low number of requests per seconds and then gradually increasing that number with time. The graph of this workload, with requests per seconds plotted on the Y-axis and time on the X-axis, looks like a series of steps. Therefore, this workload is also referred to as a step workload. Two important parameters for step workloads must be chosen carefully.

The first parameter is the time for which a constant number of requests per second are

generated by the benchmarking tool before proceeding to the next step. This is referred to as the step interval, during which the requests per second generated by a tool remain fixed. Each step interval should be large enough to allow stabilization of the system being benchmarked before moving on to the next step. This helps in getting accurate benchmarking results by identifying issues like memory leaks, periodic errors, etc. However, using a large value for step interval can result in long benchmarking experiments that can consume much time, resources, and money. Hence having an appropriate step interval size is important.

The second factor is the step size, which is the number of requests per second with which the generated load is incremented while moving from one step to the next. Just like the step interval, the step size must be carefully selected. If a step size is too small then we would need a lot of steps to hit the breaking point of the system being benchmarked. Just as we discussed for the step interval, this might result in consuming considerable time, resources, and money during our experiments. However, if the step size is too big then we lose the granularity with which we can differentiate between the breaking point for two systems. Hence an appropriate value must be chosen for the step size to maintain a fair balance between time and resources required for benchmarking versus the granularity of the results.

5.2.2 Sufficient Duration

A benchmarking study must run for a sufficient duration to capture a system's stable performance, representing its capability to take a workload for long duration. This is important for two reasons. First, data caches are present in every system on different levels like application level, language level, database level, operating system level, etc. Benchmarking a system for a long time allows for this data caching to take effect, and smooths out its impact over time. This yields accurate results which are consistent, reproducible, and representative of real-life scenarios. Second, systems having resource leaks generally perform well when benchmarked for a small amount of time because the application hasn't used up all available resources yet. However, such a benchmark does not represent the real-world use case. Thus, to get accurate results, benchmarking these systems for a long time helps us to identify such resource leaks and measure accurate performance under resource bottleneck scenarios.

5.2.3 Consistent System Configuration

Configurations play an important role in any system's performance. By tweaking the system configurations for application, database, Web server, or operating system, vast performance gains can be achieved. Any system's optimal performance is accurately marked only when the system is correctly configured. Due to the big impact that configurations have on a system's performance, it is vital to maintain consistent system configurations when conducting benchmarking comparisons between different systems or different modes in the same system. This eliminates any performance differences that could creep into the results due to different

values of system configuration, which might have a significant impact on performance.

5.2.4 Bottleneck Identification

We define a system's breaking point as the load at which its performance starts degrading with a significant increase in errors and average latency (more than an acceptable value), and a stagnant throughput. In a system faced with an increasing workload, a breaking point is achieved due to hitting a bottleneck on one of the various limited resources available to it. Some of the resources in a system that can be the bottleneck are CPU, RAM, networking bandwidth, and disk bandwidth. Identifying a bottleneck during the benchmarking process helps us to validate the results.

For example, let's consider an application that is expected to be bounded by the CPU. Hence, at the breaking point such an application should ideally have 100 percent CPU usage. However, if during benchmarking at the breaking point it shows CPU usage less than 100 percent then we know that this application is not bounded by the CPU. In such a scenario, the server's other limited resources like RAM, network bandwidth, or disk bandwidth could be the bottleneck. Once we identify the bottlenecked resource of this system, we can verify the results by confirming that it saturates at the breaking point of the system. This is why identifying the bottleneck of a system helps us to validate the results and provide an explanation for the errors received.

5.2.5 Multiple Iterations

Since operating systems run multiple processes on limited hardware, these processes share the resources available to them. That is why many processes compete with the process under benchmarking to acquire the system's resources. When these processes acquire the system's resources, the process under benchmarking cannot continue until they relinquish control of the resources required by the process under benchmarking. Operating systems allocate system resources to processes by using a non-deterministic scheduling algorithm. Hence the presence of many competitive processes on a system might erratically degrade the performance of the process that we intend to benchmark. This degradation can sometimes yield unexpected and inconsistent results, which generally manifest themselves as outliers. It is not possible to completely eliminate these undesired results; however, we can still mitigate their impact by running only the absolutely necessary processes on the machine under benchmarking. This will eliminate any additional or unexpected usage of the shared resources by other applications. It also is helpful to mitigate these impacts by running multiple iterations of benchmarking on the system. Generally, three to five is considered an acceptable number of iterations to calculate the final result. The final result can be calculated by taking the average of all the readings obtained in these iterations.

5.2.6 Resource Monitoring

Resource monitoring is an essential part of the benchmarking process to conduct a fair study. It helps us to identify the critical resources at heavy workloads, which gives us the opportunity to improve the performance of systems by carefully tuning and utilizing these resources optimally. Additionally, it allows us to catch any unexpected pattern in the system's resource usage, which might have occurred due to some undesired factors like background processes, intermittent memory errors, etc. By carefully analyzing and eliminating the cause of such undesired factors from the system, we can perform more accurate and fair benchmarking that leads to reproducible and reliable results.

5.3 YCSB REST Workload and REST Module

Previously, the YCSB only supported the benchmarking of certain key-value stores and databases. Our contribution of the REST module[1] to the YCSB adds the support to benchmark RESTful Web services with highly configurable workloads.

5.3.1 Architecture

Figure 5.1 shows the components in the YCSB that are responsible for generating the HTTP workload and the execution flow of this process. All the components shown in this figure in blue represent our contribution to the YCSB. They have been designed and implemented to efficiently generate various types of HTTP workloads. The components in green represent interfaces, while those in yellow represent the default core workload. The latter two types of components were designed and developed by the YCSB authors.

1. https://github.com/brianfrankcooper/YCSB/tree/master/rest

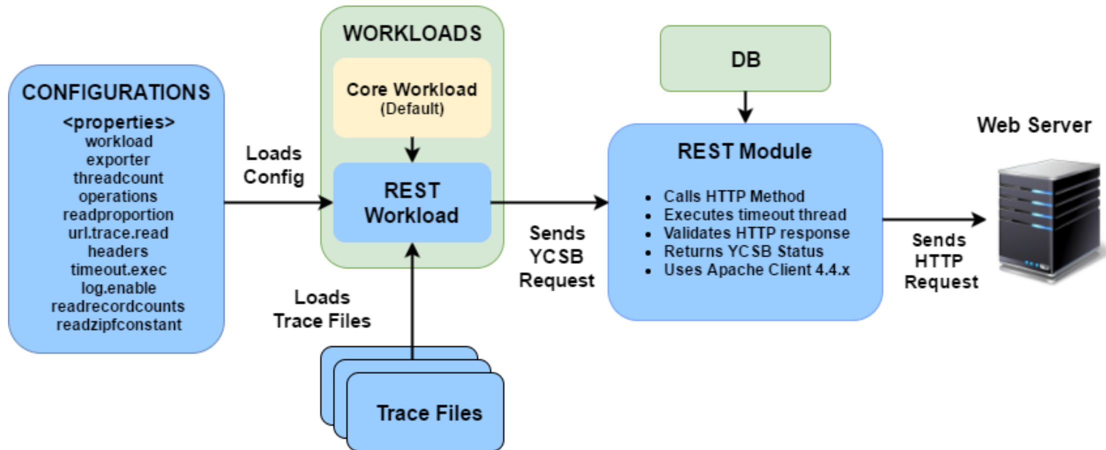

Figure 5.1: YCSB REST Module Architecture

5.3.2 REST Workload

Previously, the core workload was the only workload in the YCSB core, since all the databases and key-value stores generate similar types of benchmarking workloads. This is because CRUD operations on databases and key-value stores are performed on fixed endpoints; however, Web services can have multiple endpoints. Additionally, to simulate real-life workloads for Web services, it is important to accurately characterize Web service workloads using trace files. Due to these reasons, a new workload was required for the REST module. We refer to this as the REST workload[2]. The REST workload extends the core workload since it inherits common functionality from it. However, the REST workload supports a few additional features given below that the core workload does not:

1. It can generate a benchmarking workload using trace files (a collection of HTTP URL resources to be hit for benchmarking) specific to different HTTP CRUD (GET, PUT, POST, DELETE) operations.
2. It can configure custom values for distribution parameters like Zipfian distribution constant, etc.
3. It supports configurable values to load only the top k resources from the trace files in-memory for benchmarking.

2. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/workloads/RestWorkload.java

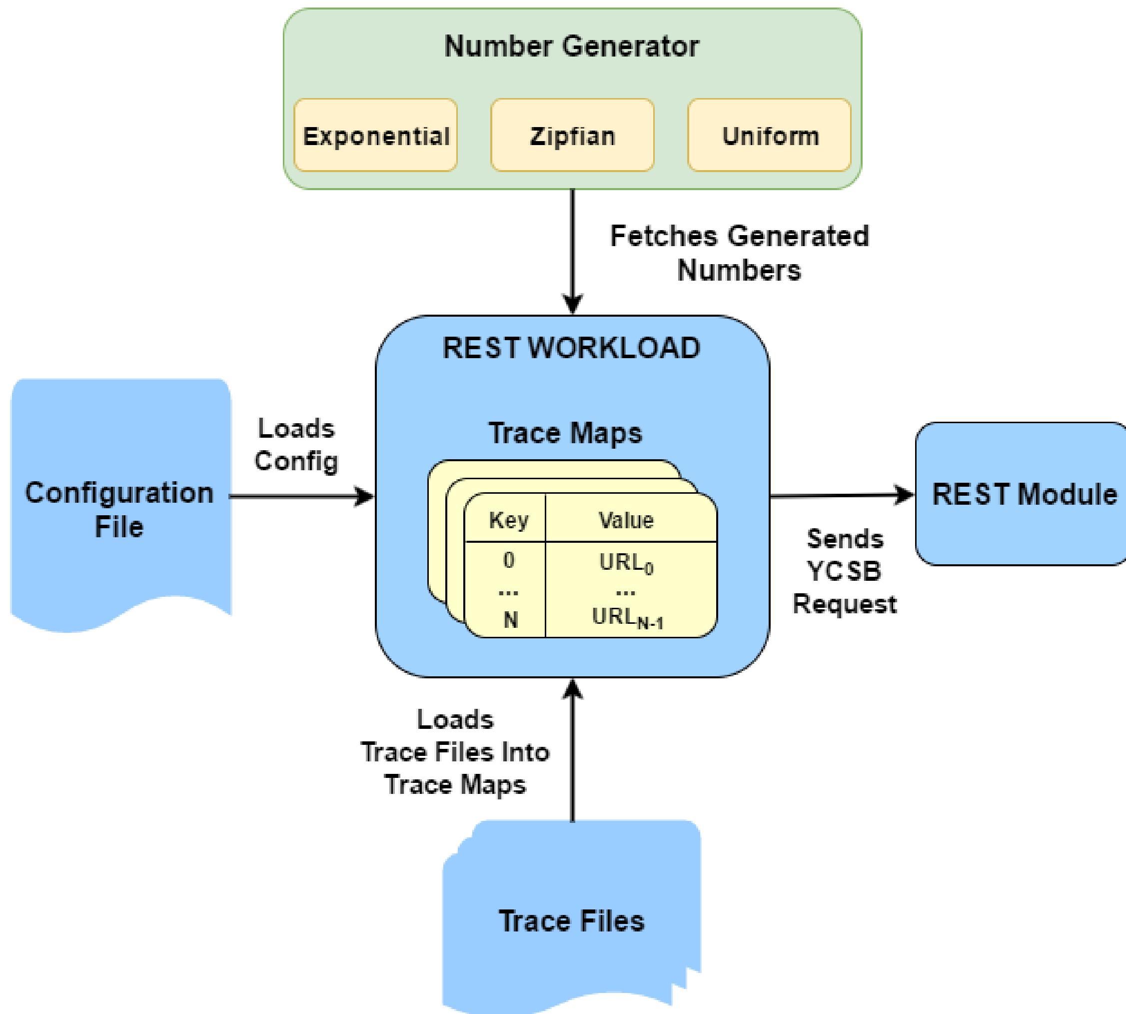

Figure 5.2: YCSB REST Workload

Figure 5.2 shows the steps involved in the creation of a YCSB benchmarking request by the REST Workload. The REST Workload parses the configuration file and fetches the user input value of parameters like benchmarking trace files location, request distribution, etc. Then it loads the trace files into in-memory Hash Maps [42] once, at the initialization phase. These trace maps store sequential numbers (starting from zero) as keys and URL endpoints from the trace files (in-order) as their values. The REST Workload must generate a number to create a YCSB benchmarking request. For this purpose, it uses a number generator that corresponds to the request distribution stated by the user in the configuration file. The NumberGenerator[3] abstract class (represented by the green color) is extended by various types

3. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/generator/Generator.java

of number generators like Exponential, Zipfian, Uniform, etc. These generators generate numbers that follow the distribution represented by them. For example, UniformGenerator[4] generate numbers uniformly in a given range. Similarly, ZipfianGenerator[5] generate numbers using the Zipfian distribution in a given range. Finally, the REST Workload uses the generated number to select the corresponding URL endpoint from the trace hash map and creates a YCSB benchmarking request for this URL. These YCSB benchmarking requests are then sent to the YCSB REST module, which creates HTTP requests to benchmark a Web service.

5.3.3 REST Module

Similar to the other modules, for example accumulo, cassandra, etc., the REST module is also a child module of the parent YCSB Maven project. The REST module's main class, RestClient[6], implements the DB interface of the core module. However, instead of making database CRUD calls for benchmarking, the REST module makes HTTP CRUD calls since the endpoints are HTTP URLs. Hence the REST module can be considered as a module that takes the standard YCSB request object and converts it to an HTTP GET, PUT, POST or DELETE call according to the received request. The Apache HTTP Client 3.0 library [43] is used in the RestClient class to generate these HTTP requests for benchmarking.

5.3.4 REST Configurations

The benchmarking specification file contains all the properties required to correctly set up the benchmarking process in the YCSB. The REST configuration includes all the specifications required for the REST module. These configurations give users the ability to generate flexible benchmarking workloads. In this section, we will describe all the custom properties that we have defined and implemented for the REST module.

url.prefix

This is the base endpoint URL where the Web service to be benchmarked is running. URL endpoints from the trace files (DELETE, GET, POST, PUT) will be prefixed with this value before making an HTTP request. A common usage value would be http://127.0.0.1:8080/webservice. The default value is http://127.0.0.1:80/.

4. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/generator/UniformGenerator.java
5. https://github.com/brianfrankcooper/YCSB/blob/master/core/src/main/java/com/yahoo/ycsb/generator/ZipfianGenerator.java
6. https://github.com/brianfrankcooper/YCSB/blob/master/rest/src/main/java/com/yahoo/ycsb/webservice/rest/RestClient.java

url.trace.read

This is the path to a trace file that holds the URL endpoints to be invoked for the HTTP GET method. The URL endpoints must be separated by a newline.

url.trace.insert

This is the path to a trace file that holds the URL endpoints to be invoked for the HTTP POST method. The URL endpoints must be separated by a newline.

url.trace.update

This is the path to a trace file that holds the URL endpoints to be invoked for the HTTP PUT method. The URL endpoints must be separated by a newline.

url.trace.delete

This is the path to a trace file that holds the URL endpoints to be invoked for the HTTP DELETE method. The URL endpoints must be separated by a newline.

headers

These are the HTTP request headers used for all requests. Headers must be separated by space as a delimiter. The default value is: Accept */* Accept-Language en-US,en;q=0.5 Content-Type application/x-www-form-urlencoded user-agent Mozilla/5.0.

timeout.con

This is the HTTP connection timeout in seconds. If the client fails to connect with the server within this time limit, that will be considered as an error. The default value is 10 seconds.

timeout.read

This is the HTTP connection timeout in seconds. If the client fails to read from the server within this time limit, that will be considered as an error. The default value is 10 seconds.

timeout.exec

This is the HTTP connection timeout in seconds. If the client fails to serve a request with the server within this time limit, that will be considered as an error. The default value is 10 seconds.

log.enable

This is a Boolean value to enable console status logs. When true, it will print all the HTTP requests being made and their response status on the YCSB console window. The default value is false.

readrecordcount

This is an integer value that signifies the top k URL endpoints to be picked from the url.trace.read file for making the HTTP GET requests. It must have a value greater than 0. If this value exceeds the number of entries present in the file, then k will be set to the number of entries in the file. The default value is 10000.

insertrecordcount

This is an integer value that signifies the top k URL endpoints to be picked from the url.trace.write file for making the HTTP POST requests. It must have a value greater than 0. If this value exceeds the number of entries present in the file, then k will be set to the number of entries in the file. The default value is 5000.

deleterecordcount

This is an integer value that signifies the top k URL endpoints to be picked from the url.trace.delete file for making the HTTP DELETE requests. It must have a value greater than 0. If this value exceeds the number of entries present in the file, then k will be set to the number of entries in the file. The default value is 1000.

updaterecordcount

This is an integer value that signifies the top k URL endpoints to be picked from the url.trace.update file for making the HTTP PUT requests. It must have a value greater than 0. If this value exceeds the number of entries present in the file, then k will be set to the number of entries in the file. The default value is 1000.

readzipfconstant

This is a double precision value of the Zipf constant to be used for read requests. Applicable only if the request distribution is equal to Zipfian. The default value is 0.9.

updatezipfconstant

This is a double precision value of the Zipf constant to be used for update requests. Applicable only if the request distribution is equal to Zipfian. The default value is 0.9.

deletezipfconstant

This is a double value of the Zipf's constant to be used for delete requests. Applicable only if the request distribution is equal to Zipfian. The default value is 0.9.

insertzipfconstant

This is a double value of the Zipf's constant to be used for insert requests. Applicable only if the request distribution is equal to Zipfian. The default value is 0.9.

Chapter 6

Experimental Design

6.1 Introduction

In this chapter we provide a description of our experimental design. First, we explain the overall architecture and the request flow for our benchmarking setup. Then we present the three benchmarking modes of our experiments that helped us to quantify the performance concerns of SiteStory and compare its performance with our local archiving solution.

6.2 Experimental Setup

Depending upon the benchmarking mode, our experimental setup consists of two or three dedicated servers or machines. These servers communicate with each other on a private network connected by an Ethernet switch. More details on these benchmarking modes is provided in the subsections below.

6.2.1 YCSB Client

The YCSB client is responsible for benchmarking the performance of the Apache Web server. It does so by generating HTTP Web requests and measuring the performance evaluation metrics including throughput, average latency, and errors of the responses received. As mentioned before in Chapter 4, benchmarking traces and workload configuration files are used to generate the HTTP workload. Apart from the benchmarking task, this machine is also responsible for the real time monitoring and visualization of resource usage – like CPU, RAM, network bandwidth, and disk bandwidth – of the Apache Web server. It indexes the incoming resource monitoring data from the Apache Web server into an Elastic Search service [44], which is then visualized by Kibana [45] on a Web browser.

Specification

Table 6.1 shows the specifications of the YCSB client machine that we use for our experiments.

Property	**Value**
Operating System	Mac OS X El Capitan - 10.11.13
Central Processing Unit	2.66 GHz Intel Core i5
Cores	4 (2 processors and 2 hyper-threaded)
Random Access Memory	4 GB 1067 MHz Double Data Rate 3
Storage	1 TB SATA Disk
Software	Java 1.7.0, Elastic Search, Kibana

Table 6.1: System specification of YCSB Client

6.2.2 Apache Web Server

The Apache Web server machine runs MediaWiki, the Web application which serves the workload generated by the YCSB client. By comparing the performance of this server in different modes of operations, we evaluate the performance overhead of SiteStory and our local archiving solution. In order to monitor its resources we used collectl [46], which captures the CPU, RAM, disk and network usage every ten seconds and appends it to a file. This file is then used by a locally running Logstash [47] service, which sends this resource usage data for real time monitoring to the remote Elasticsearch service, running on the YCSB Client.

As mentioned in Chapter 4, the Web application running on this server is MediaWiki, which hosts a Greek Wikipedia mirror. The software installed on this server includes the Apache HTTP server, Java, Logstash, PHP, MySQL, and PHP APCu [48].

Specification

To cover the four data points in our design space, we ran the experiments on two separate machines with different specifications. One of the machines has high computational capacity while the other has relatively lower computational capacity. Tables 6.2 and 6.3 show the specifications for the high and low computational capacity machines that we used for our experiments, respectively.

Property	**Value**
Operating System	CentOS - 6.6
Central Processing Unit	2.66 GHz Intel Core 2 Duo
Cores	2 (1 processor and 1 hyperthreaded)
Random Access Memory	4 GB 1333 MHz Double Data Rate 3
Storage	250 GB SATA Disk

Table 6.2: System specification of the low computational capacity Web server

Property	**Value**
Operating System	CentOS - 6.6
Central Processing Unit	2.8 GHz Intel Core i7
Cores	8 (4 processors and 4 hyper-threaded)
Random Access Memory	32 GB 1333 MHz Double Data Rate 3
Storage	1 TB SATA Disk

Table 6.3: System specification of the high computational capacity Web server

6.2.3 SiteStory Server

The SiteStory server is responsible for receiving and processing the HTTP PUT Web archiving requests sent by the Apache Web server over the private network. This server runs the SiteStory Web application deployed on an Apache Tomcat server. Every Web archiving request received by the SiteStory server is processed, and if the response is not already archived, it is stored persistently using the Berkeley database indexes. A few of the most important software components installed on this machine include Apache Tomcat and Java.

Specification

Table 6.4 shows the specifications of the SiteStory server that we used for our experiments.

Property	**Value**
Operating System	Mac OS X El Capitan - 10.11.13
Central Processing Unit	3.1 GHz Intel Core i5
Random Access Memory	32 GB 1333 MHz Double Data Rate 3
Storage	4 TB SATA Disk

Table 6.4: System specification of the SiteStory server

6.2.4 Ethernet Switch

We set up a dedicated private network to run all our benchmarking experiments by using an Ethernet switch to create this network and connect all the machines together. Using a dedicated private network helped us to avoid any network inconsistencies that may creep into our experiments due to shared usage of network bandwidth in shared network architectures.

Specification

To cover the four data points in our design space, we ran our experiments on two separate networks of different speeds. We varied the network speed by using two Ethernet switches of different specifications. One of the network switches supported a maximum bandwidth of 100 Megabits per second (Mbps) while the other one was 10 times faster, and supported a maximum bandwidth of 1 Gigabit per second (Gbps). Tables 6.5 and 6.6 show the specifications for the low and high network bandwidth switches that we used for our experiments, respectively.

Property	**Value**
Company	Cisco
Model	8 Port Workgroup Ethefast 10/100 LinkSys Switch
Speed	100 Mbps

Table 6.5: Specification of the low network bandwidth switch

Property	**Value**
Company	Netgear
Model	GS108 ProSafe 8 Port Gigabit Switch
Speed	1 Gbps

Table 6.6: Specification of the high network bandwidth switch

6.3 Modes

In this section we describe our three different benchmarking modes – Mode A, Mode B, and Mode C. Evaluating the performance of the Apache Web server in all these modes helped us to quantify the overhead incurred by SiteStory on the Web server it archives. It also

helped us to evaluate the performance of our local archiving solution and compare it with the SiteStory performance.

6.3.1 Mode A

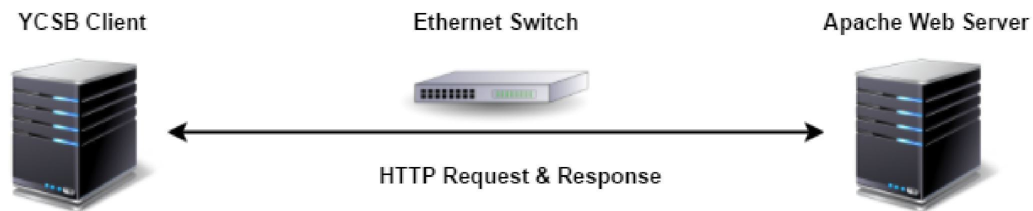

Figure 6.1: Architecture of Mode A

Figure 6.1 shows the architecture diagram of Mode A. In this mode, we benchmark the performance of the Apache Web server by generating requests from the YCSB client. As mentioned in Chapter 3, the YCSB client generates HTTP GET and PUT requests for the Greek Wikipedia mirror from the benchmarking traces, which were prepared using the access pattern data from the Wikipedia statistics website. Figure 6.1 shows the flow of a typical request from the YCSB client. The YCSB client sends an HTTP request to the Wikipedia mirror and records the performance metrics (average latency, throughput, response status) when the corresponding response is received. The aim of benchmarking this mode is to evaluate the performance of the bare Wikipedia server on our dedicated hardware without any archiving taking place. These results were compared with the results of Mode B and Mode C, which helped us to calculate the performance implications of using SiteStory and our local archiving solution for archiving Web requests, respectively.

6.3.2 Mode B

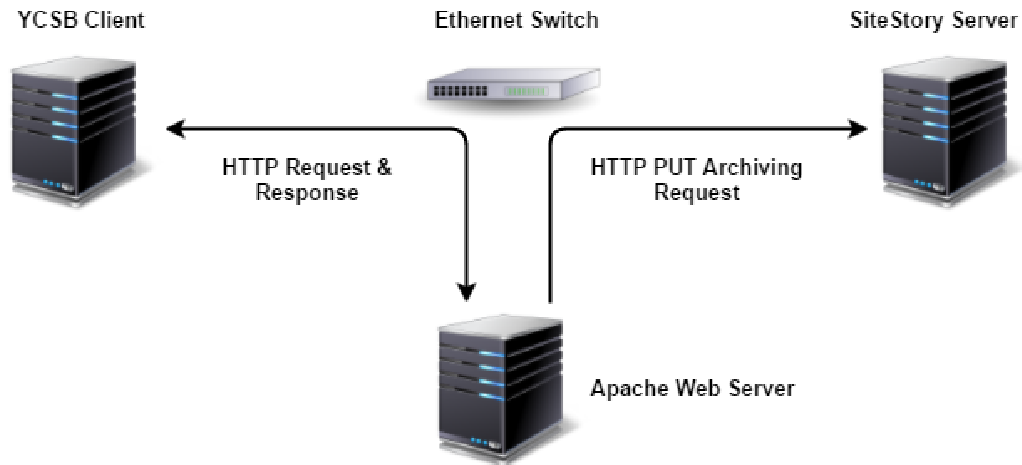

Figure 6.2: Architecture of Mode B

Figure 6.2 shows the architecture diagram of Mode B. Just like Mode A, in this mode we benchmarked the performance of the Apache Web server from the YCSB client, with just one notable difference. Here we enabled the SiteStory module on the Apache Web server. Hence for every request served by the Apache Web server, one HTTP response and one HTTP PUT request was generated, as opposed to only one HTTP response in Mode A. As shown in Figure 6.2, the HTTP response was sent to the YCSB client with the requested data and the HTTP PUT request was sent to the SiteStory archiving server. The HTTP PUT request body contained the HTTP response and the HTTP request for which this response was generated.

The aim of benchmarking this mode was to evaluate the performance of the Wikipedia server with SiteStory enabled. The performance differences captured by the benchmarking results of this mode versus Mode A represent the overhead incurred by the Apache Web server due to enabling Web archiving using SiteStory. This helped us to verify the claims made by SiteStory's authors by comparing the performance metrics captured in this mode versus Mode A.

6.3.3 Mode C

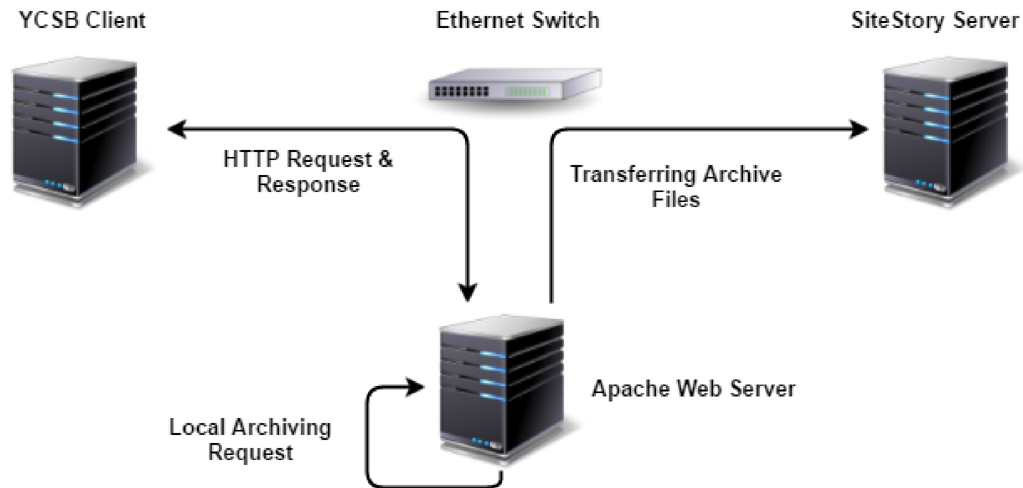

Figure 6.3: Architecture of Mode C

Figure 6.3 shows the architecture diagram of Mode C. This mode is similar to Mode B, with one difference. Instead of sending the Web archiving HTTP PUT requests to a remote SiteStory server, we configured the Apache SiteStory module to send these requests to a local archiving Web service. As mentioned in Chapter 2, a light weight local archiving Web service, written in PHP, that runs on the Apache Web server, processes these HTTP requests and persists them as individual files on the hard disk. The aim of benchmarking this mode was to evaluate the performance of the Wikipedia server with our local Web archiving solution and to capture its resource utilization. This helped us in understanding the performance overheads incurred by our local archiving solution on the Apache Web server and to compare it with SiteStory. Figure 6.3 shows a typical request and response flow of Mode C.

Chapter 7

Results

7.1 Introduction

In this chapter we present the results of our benchmarking experiments for the four data points in our design space. We compare the average latency for the three modes – A, B, and C – that were obtained in our experiments.

7.2 Fast Computation Fast Network

In this subsection we present the results of our experiments conducted in the Fast Computation Fast Network data point in our design space.

7.2.1 Benchmarking Results

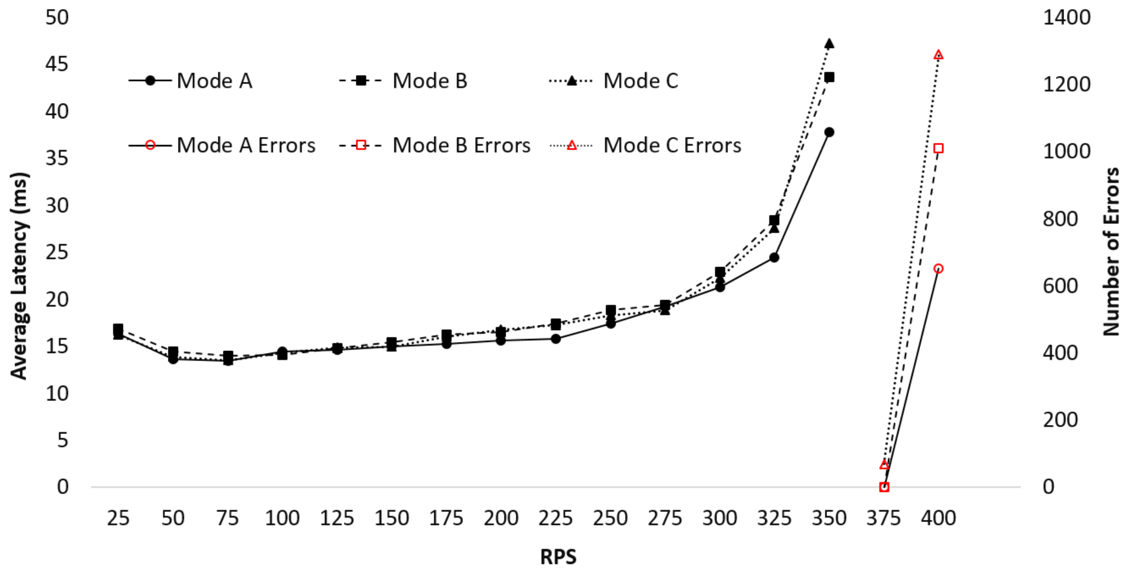

Figure 7.1: Average Latency and Errors versus RPS for Fast Computation Fast Network data point

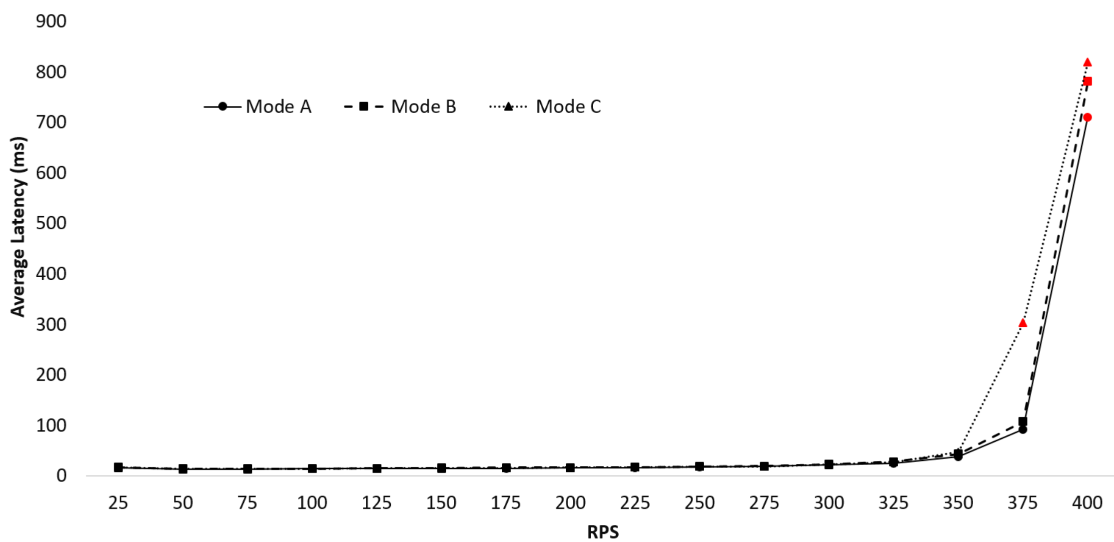

Figure 7.2: Average Latency versus RPS for Fast Computation Fast Network data point

Figure 7.1 shows the average latency and the number of errors versus the number of requests per second (RPS) received by the Apache Web server. The errors are shown in red, and with open shapes. Figure 7.2 shows the average latency graph till 400 RPS. The points in

red signify the occurrence of errors in the corresponding mode. We observed that Mode C breaks at 375 RPS. In contrast, both Mode A and Mode B break at 400 RPS.

Resource Utilization of the Web server

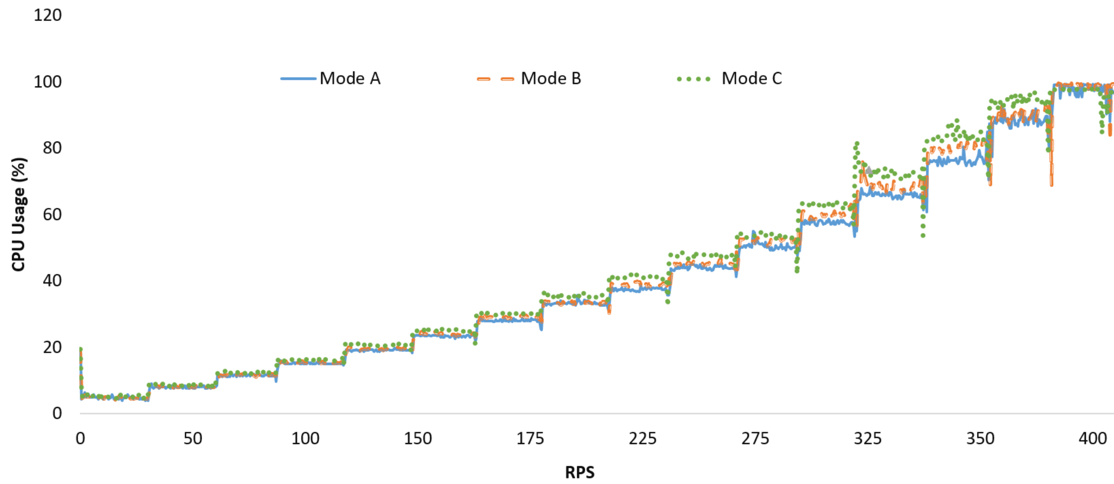

Figure 7.3: CPU Usage for Fast Computation Fast Network data point

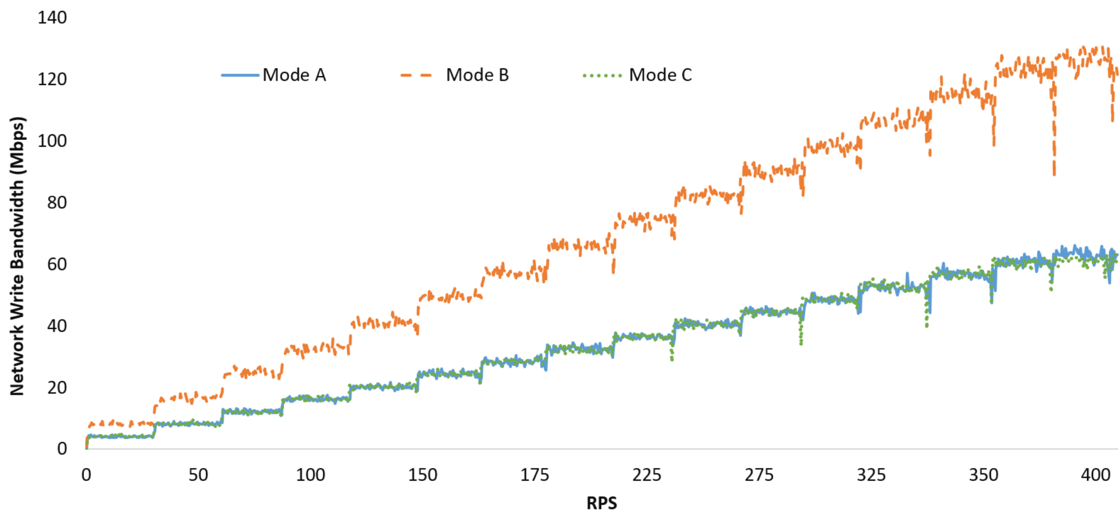

Figure 7.4: Network Write Bandwidth Usage for Fast Computation Fast Network data point

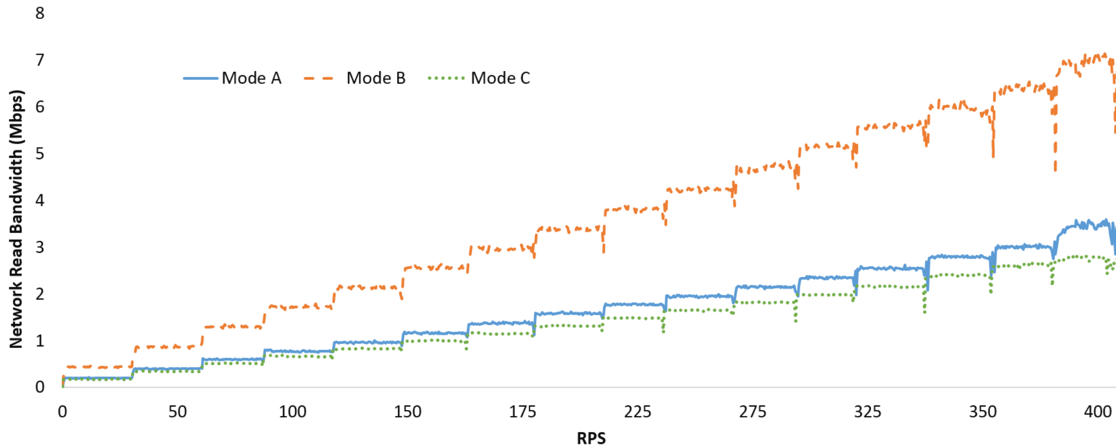

Figure 7.5: Network Read Bandwidth Usage for Fast Computation Fast Network data point

Figures 7.3, 7.4 and 7.5 show the CPU, network write bandwidth, and network read bandwidth used by the Web server with respect to the RPS received by the Apache Web server, respectively. The network write bandwidth corresponds to the rate at which the network traffic is sent out to the connected network by a server whereas the network read bandwidth corresponds to the rate at which network bandwidth is read by it.

During the experiments, a noteworthy observation was that the performance of all the modes was bounded by the CPU because, at 375 RPS for Mode C and 400 RPS for Mode A and Mode B, the system reached 100 percent CPU usage and errors started to occur. Also, the CPU usage increases in steps with time because we used a step benchmarking workload for all of our experiments. In addition to that, the vertical lines in the graph occur at a regular interval corresponded to a sudden dip in the CPU usage. They occur because after the end of every benchmarking step, YCSB halts to prepare itself for the next step, thereby generating no benchmarking requests for a small duration. Since the Apache Web server receives no requests during this time period, its CPU usage suddenly goes down.

The results also indicate that the network write and read bandwidth used in Mode B were twice the bandwidth used in Mode A and Mode C. This finding is in accordance with our expectations since Mode B sends an additional copy of the HTTP response to the SiteStory archiving server for every HTTP request, thereby doubling the network write bandwidth consumed. The network write bandwidth for Mode A and Mode C were similar because Mode C used the loop-back interface to send the archiving request to the local archiving service, thereby consuming the same network bandwidth as Mode A. The network read bandwidth for Mode C was double because for every archiving request sent by the Apache Web server, the SiteStory server sends an HTTP 204 No Content as the archiving response. Hence in addition to the benchmarking requests from the YCSB client, the Apache Web server in Mode B also receives the archiving response from the SiteStory server.

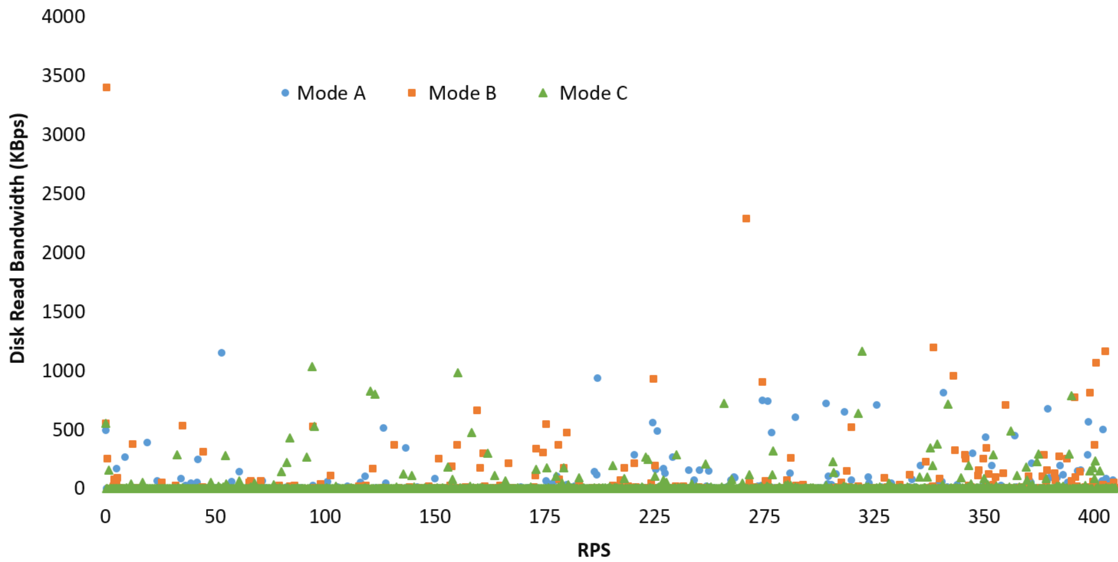

Figure 7.6: Disk Read Bandwidth Usage for Fast Computation Fast Network data point

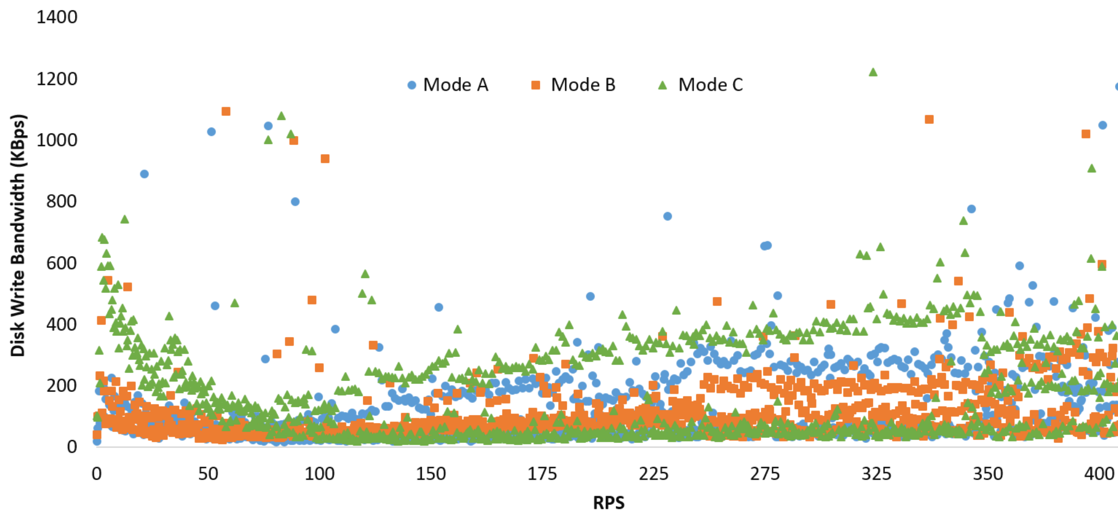

Figure 7.7: Disk Write Bandwidth Usage for Fast Computation Fast Network data point

Figures 7.6 and 7.7 show the disk read and write bandwidth used by the Web server versus the RPS received by it, respectively. The usage pattern for all the modes remained approximately the same, except in Mode C, regarding disk write bandwidth used. In this, consumption is consistently higher than for the other two modes, as depicted in Figure 7.7. This is because our local archiving solution performs disk write operations to archive the data locally, which the other modes do not. Additionally, the write bandwidth used by Mode C is higher at the beginning because no archived URL endpoints exist in the archive folder. Hence

every URL endpoint is written to files on the hard disk, thereby increasing the disk write bandwidth consumed. With time, some of the incoming URL endpoints will already be archived, eliminating the need to write data to files; this reduces the disk write bandwidth consumption.

7.2.2 Performance Evaluation

With the help of all the noted observations for the Fast Computation Fast Network data point in our design space, we can successfully conclude that computational availability determines the performance of the Web server in our archiving system, because all the modes are bounded by the CPU. In addition to this, SiteStory does not create any noticeable performance overhead on the Apache Web server as both Mode A and Mode B break at 400 RPS. Also, our local Web archiving solution lowers the breaking point by only 6.25 percent from 400 RPS for Mode A and Mode B to 375 RPS for Mode C.

7.3 Fast Computation Slow Network

In this section we present the results of our experiments for the Fast Computation Slow Network data point in our design space. Following the similar outline as for the Fast Computation Fast Network data point in our design space, we prepared graphs and noted observations for the average latency, number of errors, and system's resource usage.

7.3.1 Benchmarking Results

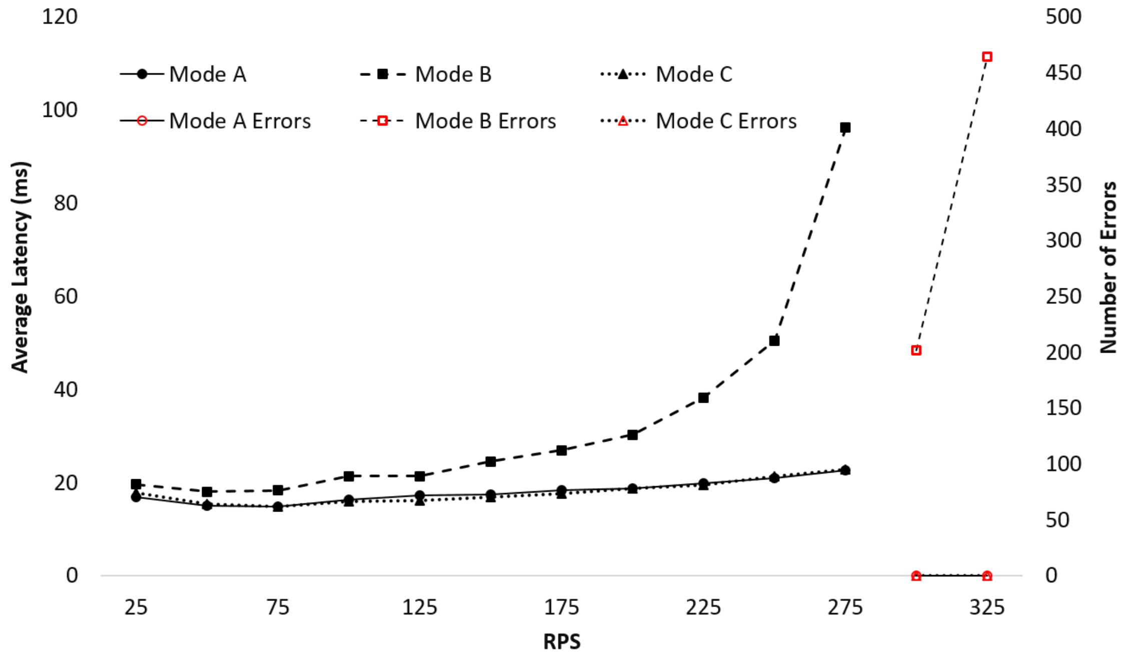

Figure 7.8: Average Latency versus Request Per Second for Fast Computation Slow Network data point

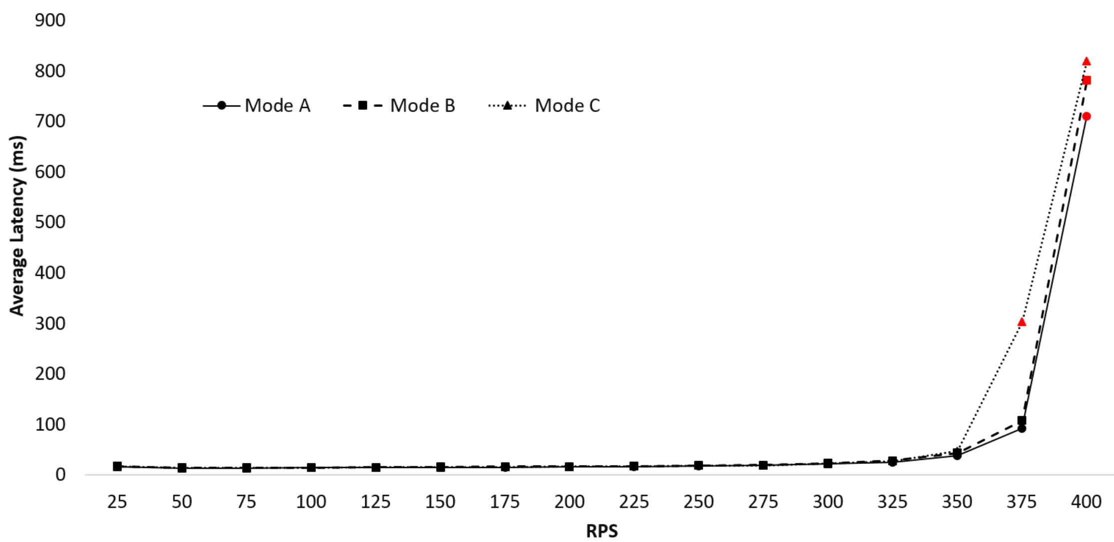

Figure 7.9: Average Latency versus RPS for Fast Computation Slow Network data point

Figure 7.8 shows the average latency versus the number of RPS received by the Apache Web server. Figure 7.9 shows the average latency till 400 RPS. We observed that Mode B breaks at 300 RPS, before Mode A and Mode C. Also, the average latency for Mode B is notably higher than that of Mode A and Mode C before its breaking point. This is as per our expectations since Mode B uses twice the network bandwidth as in Mode A or Mode C, which increases the network traffic in this data point of our design space.

Resource Utilization

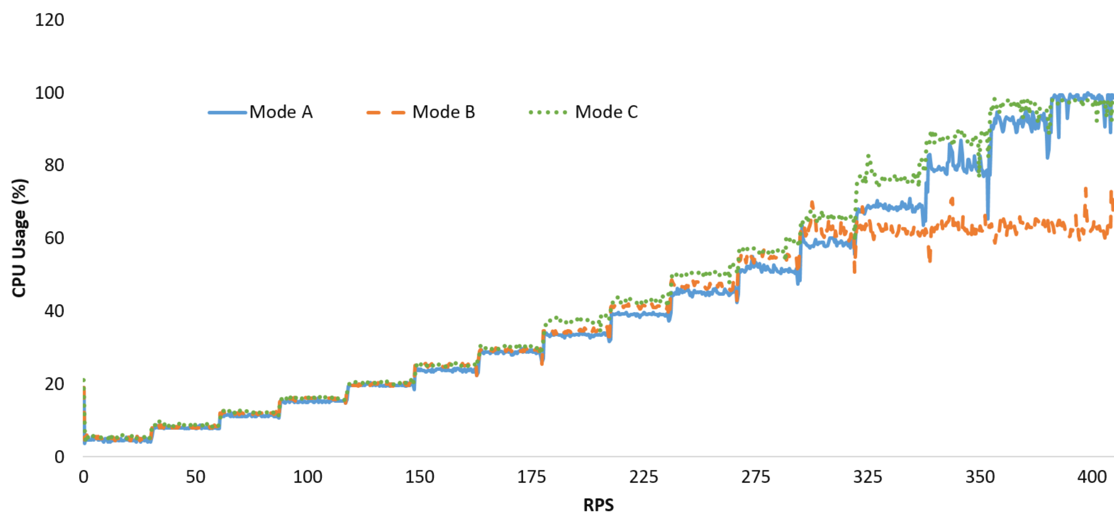

Figure 7.10: CPU Usage for Fast Computation Slow Network data point

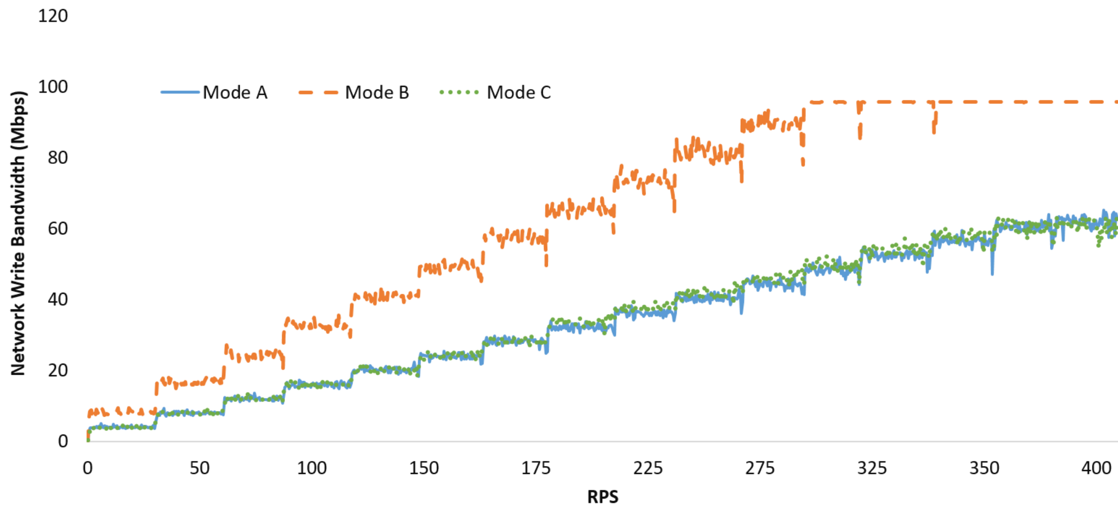

Figure 7.11: Network Write Bandwidth Usage for Fast Computation Slow Network data point

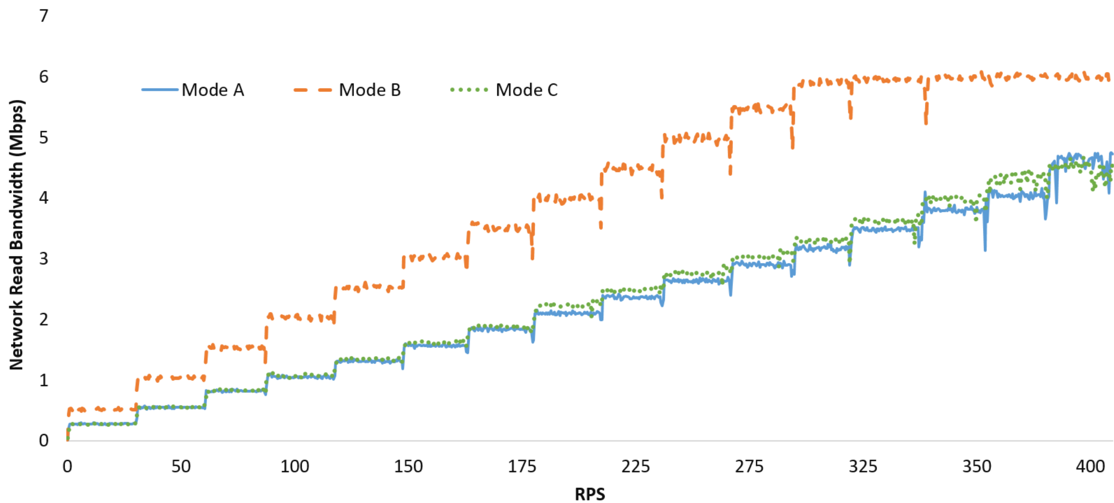

Figure 7.12: Network Read Bandwidth Usage for Fast Computation Slow Network data point

Figures 7.10, 7.11, and 7.12 show the CPU, network write bandwidth, and network read bandwidth used by the Web server with respect to the RPS received by the Apache Web server, respectively. We observed that at 300 RPS the CPU usage, network write bandwidth and network read bandwidth stay constant for Mode C at 70 percent, 94 Mbps, and 6 Mbps, with further increase in the benchmarking workload. However, Mode A and Mode C show a 100 percent CPU usage with an increasing network bandwidth usage. This points out

that at 300 RPS, Mode B is bounded by the network because the read and write network bandwidth totals to 100 Mbps, which is the network bandwidth of this data point in our design space.

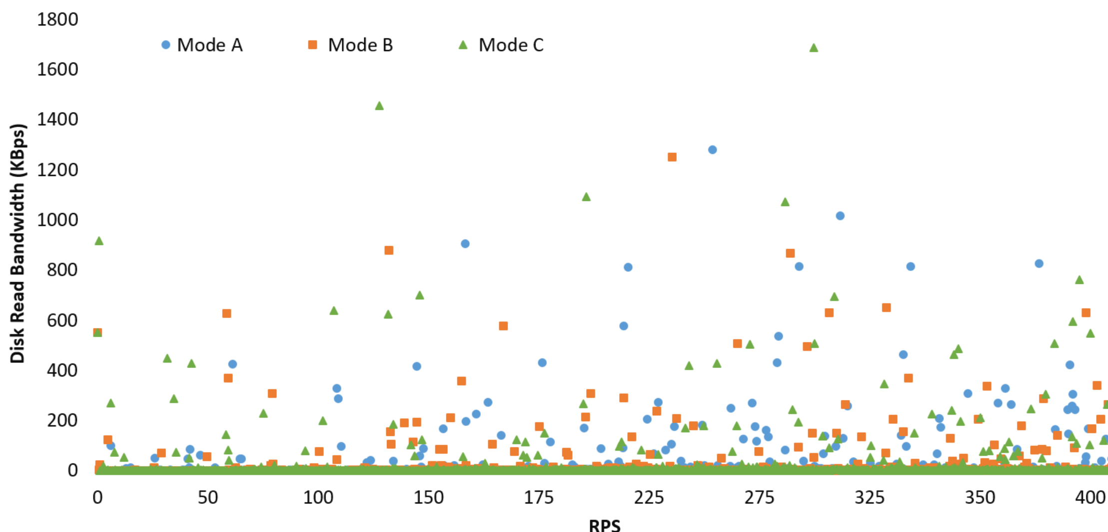

Figure 7.13: Disk Read Bandwidth Usage for Fast Computation Slow Network data point

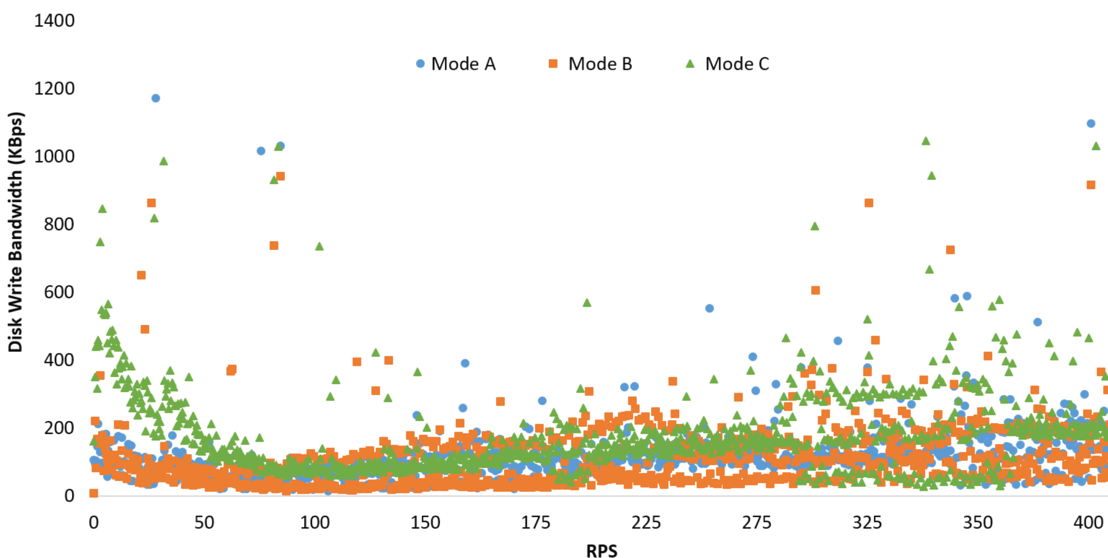

Figure 7.14: Disk Write Bandwidth Usage for Fast Computation Slow Network data point

Figurea 7.13 and 7.14 show the disk read and write bandwidth used by the Web server versus the RPS received by it, respectively. The usage pattern for all the modes remained approximately the same.

7.3.2 Performance Evaluation

Using all the noted observations for the Fast Computation Slow Network data point, we can effectively conclude that network availability for Mode B and computational availability for Mode A and Mode C determine the performance of the Web server in our archiving system. This is because Mode B is bounded by the network whereas Mode A and Mode C are bounded by the CPU.

7.4 Slow Computation Fast Network

This section covers the results of our experiments for the Slow Computation Fast Network data point in our design space. Following the similar outline as the previously discussed data points in our design space, we prepared graphs and noted observations for the average latency, number of errors, and system's resource usage.

7.4.1 Benchmarking Results

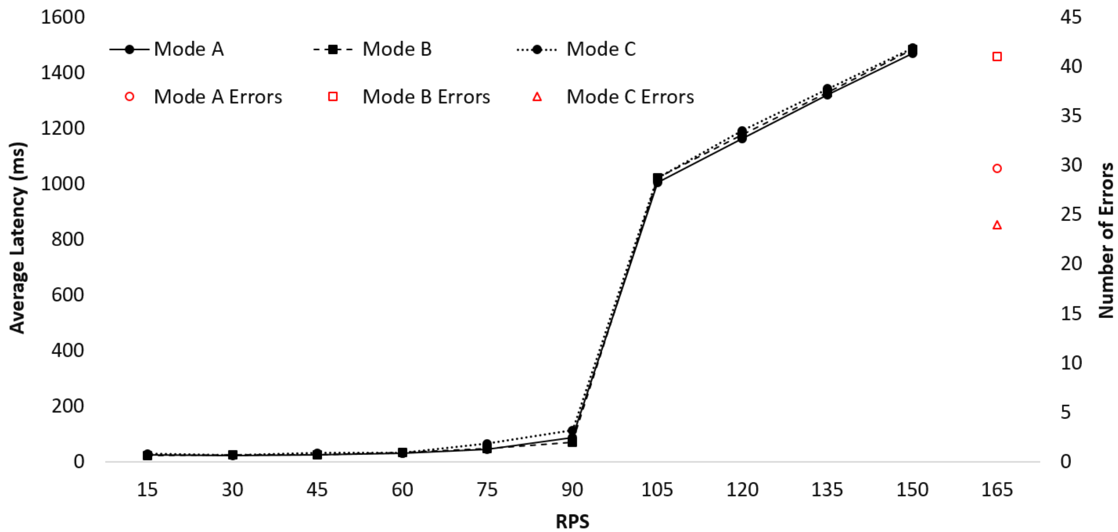

Figure 7.15: Average Latency versus Request Per Second for Slow Computation Fast Network data point

Figure 7.15 shows the the average latency versus the number of requests per second (RPS) generated by the YCSB benchmark. It was noted that all the modes break at 165 RPS with similar average latency.

Resource Utilization

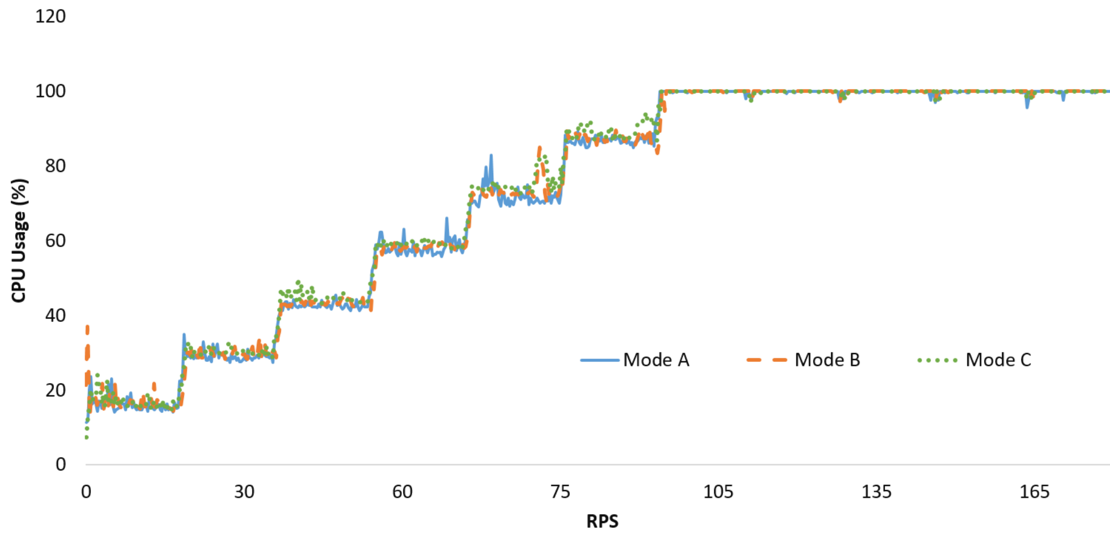

Figure 7.16: CPU Usage for Slow Computation Fast Network data point

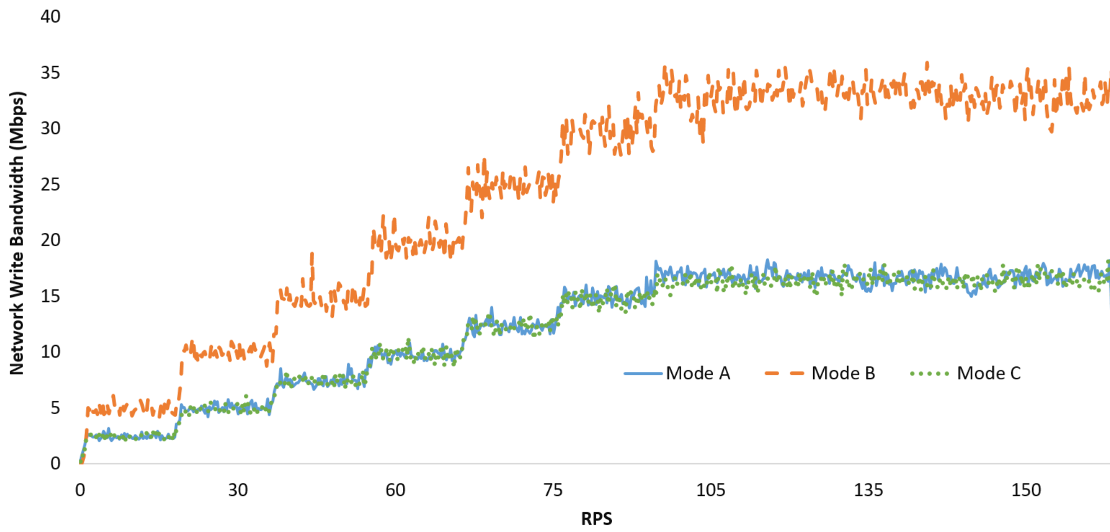

Figure 7.17: Network Write Bandwidth Usage for Slow Computation Fast Network data point

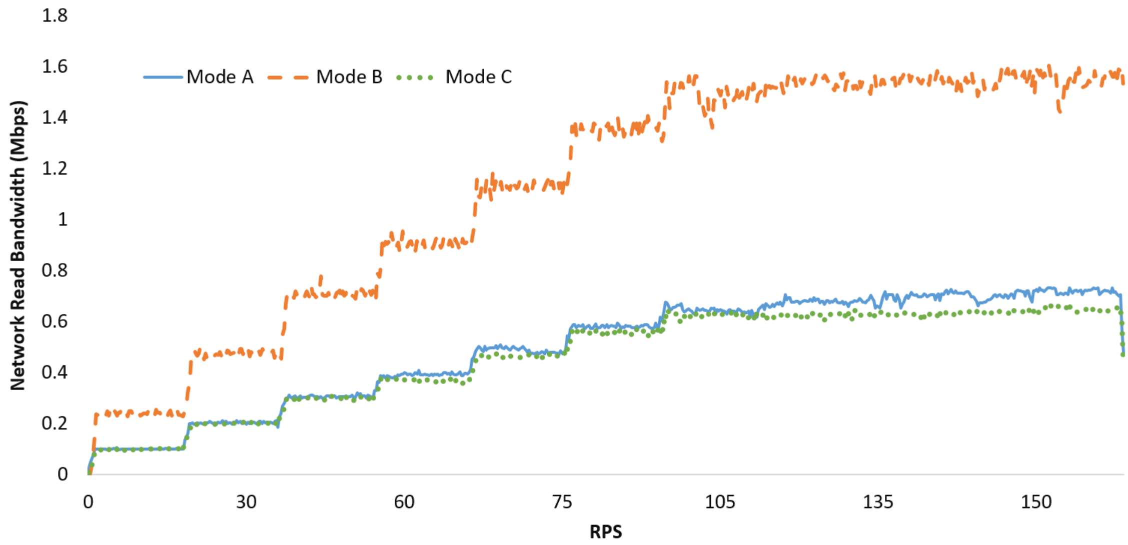

Figure 7.18: Network Read Bandwidth Usage for Slow Computation Fast Network data point

Figures 7.16, 7.17 and 7.18 show the CPU, network write bandwidth, and network read bandwidth used by the Web server with respect to the RPS received by the Apache Web server, respectively. We observed that for 100 RPS onwards, the CPU usage stays constant at 100 percent for all modes, with further increase in the benchmarking workload. At the same time, the network write bandwidth remains constant at 16 Mbps for Mode A and Mode C, and 34 Mbps for Mode B. On similar lines, the network read bandwidth stays consistent at 0.6 Mbps for Mode A and Mode C, and 1.6 Mbps for Mode B. This showed that all the modes are bounded by the CPU since the network bandwidth of this data point in our design space is 1 Gbps, which is more than the maximum bandwidth used in our experiments.

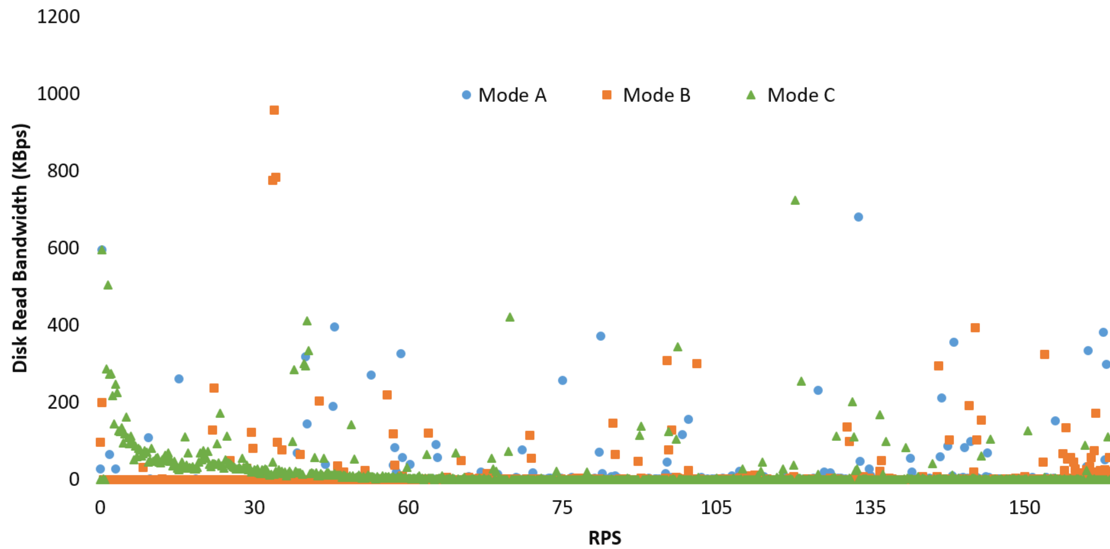

Figure 7.19: Disk Read Bandwidth Usage for Slow Computation Fast Network data point

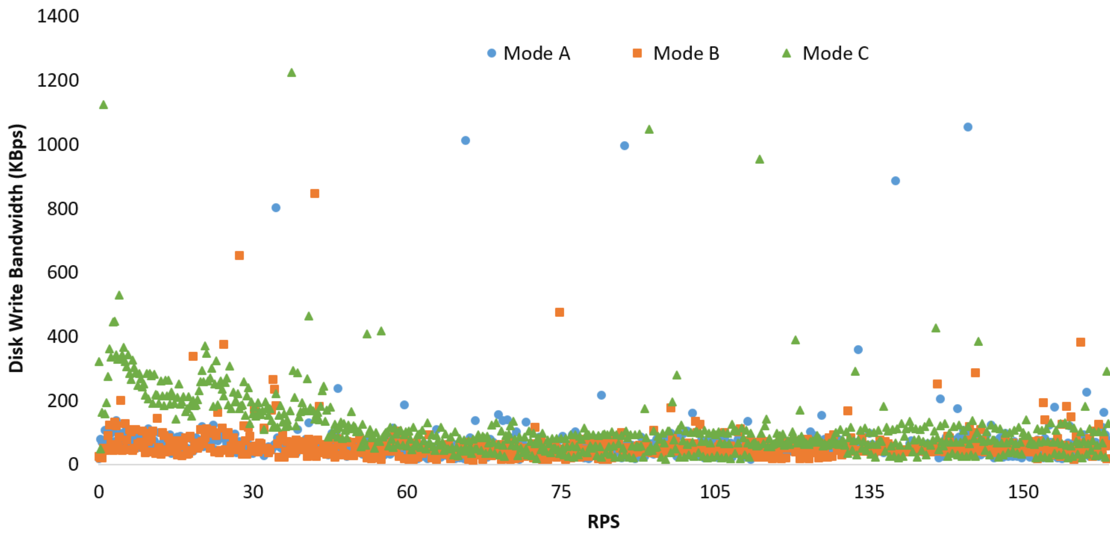

Figure 7.20: Disk Write Bandwidth Usage for Slow Computation Fast Network data point

Figures 7.19 and 7.20 show the disk read and write bandwidth used by the Web server versus the RPS received by it, respectively. The usage pattern for all the modes remained approximately the same.

7.4.2 Performance Evaluation

With the help of mentioned observations of the Slow Computation Fast Network data point in our design space, we can conclude that computational availability determines the performance of the Web server for all the modes. This is because all the modes are bounded by the CPU.

7.5 Slow Computation Slow Network

In this section we present the results of our experiments for the Slow Computation Slow Network data point in our design space. Like for the previously discussed data points in our design space, we prepared graphs and noted observations for the average latency, number of errors, and system's resource usage.

7.5.1 Benchmarking Results

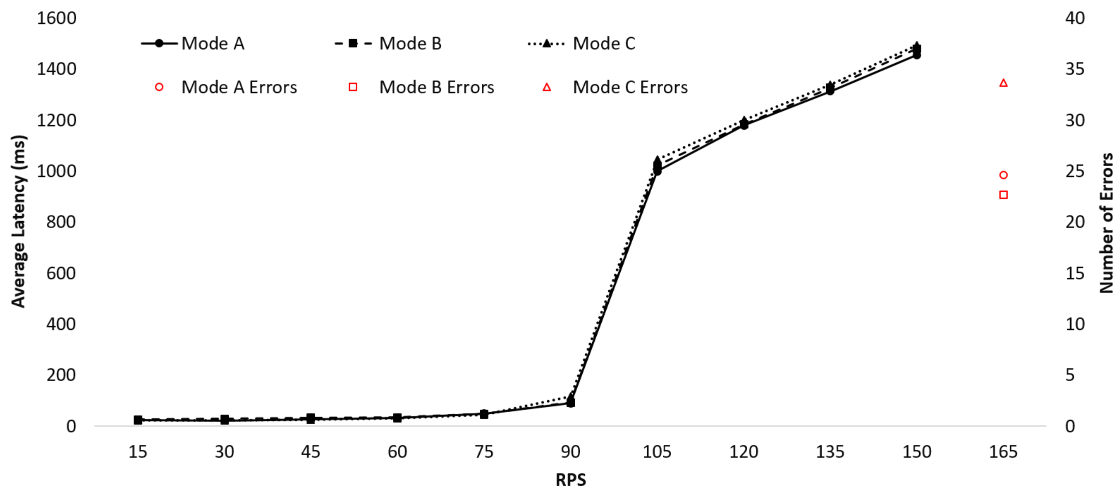

Figure 7.21: Average Latency versus Request Per Second for Slow Computation Slow Network data point

Figure 7.21 shows the the average latency versus the number of requests per second (RPS) generated by the YCSB benchmark. All the observations of this data point in our design space were noted to be the same as for the Slow Computation Fast Network data point, with the same breaking point of 165 RPS.

Resource Utilization

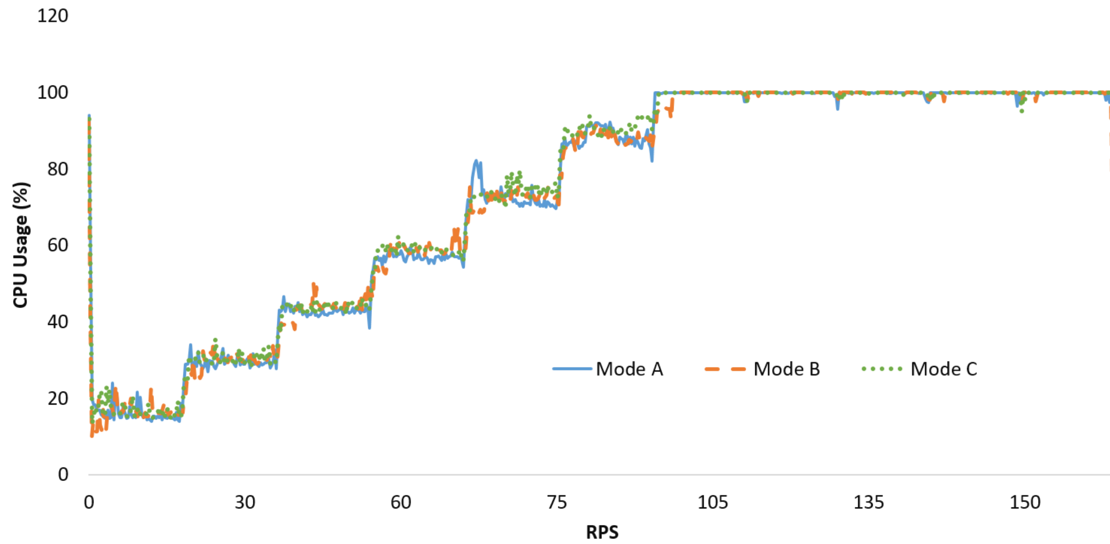

Figure 7.22: CPU Usage for Slow Computation Slow Network data point

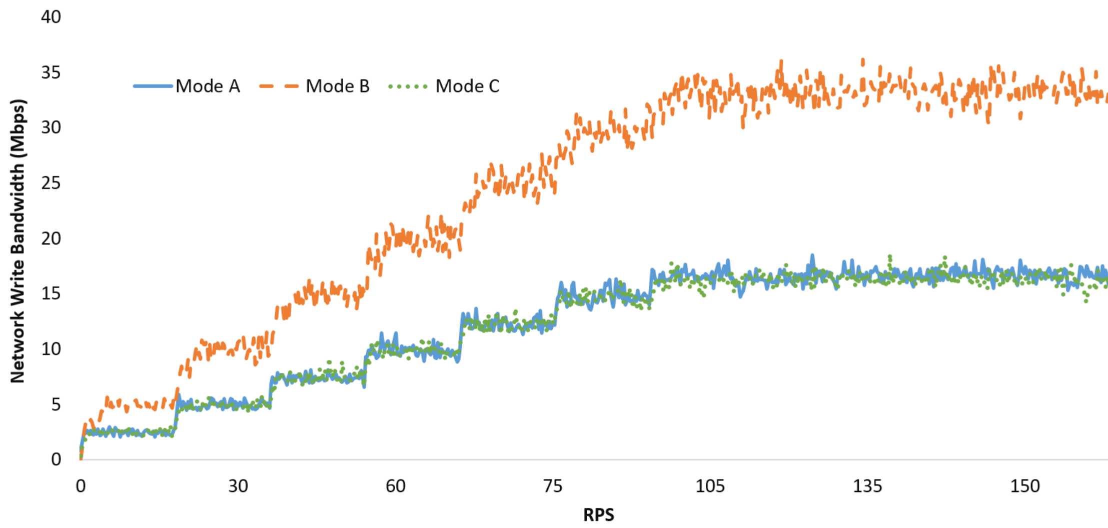

Figure 7.23: Network Write Bandwidth Usage for Slow Computation Slow Network data point

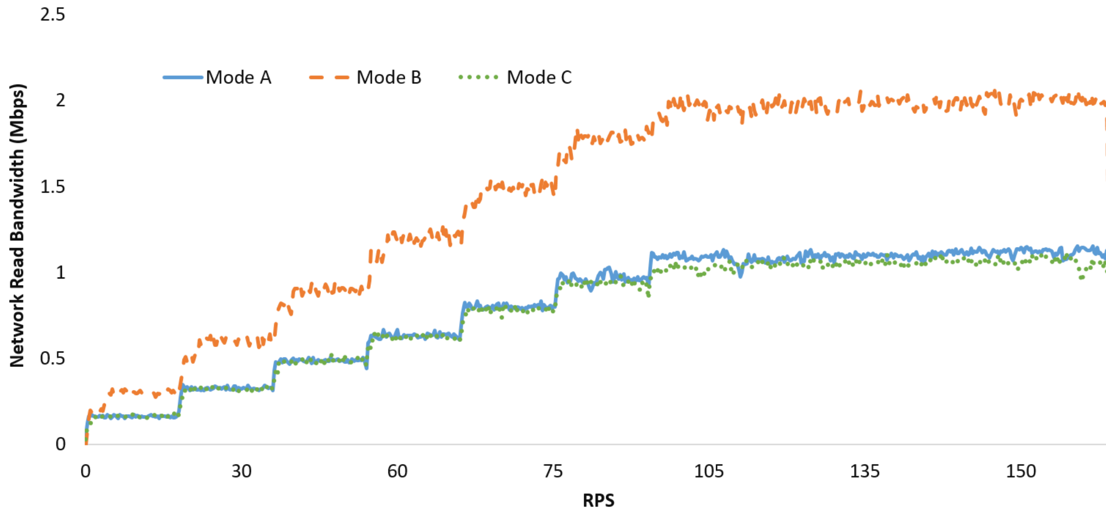

Figure 7.24: Network Read Bandwidth Usage for Slow Computation Slow Network data point

Figures 7.22, 7.23, and 7.24 show the CPU, network write bandwidth, and network read bandwidth used by the Web server with respect to the RPS received by the Apache Web server, respectively. Similar to the Slow Computation Fast Network data point in our design space, the CPU usage stays constant at 100 percent for all modes from 100 RPS onwards. The network write bandwidth remains constant at 16 Mbps for Mode A and Mode C, and 34 Mbps for Mode B. Similarly, the network read bandwidth stays consistent at 0.6 Mbps for Mode A and Mode C, and 1.6 Mbps for Mode B.

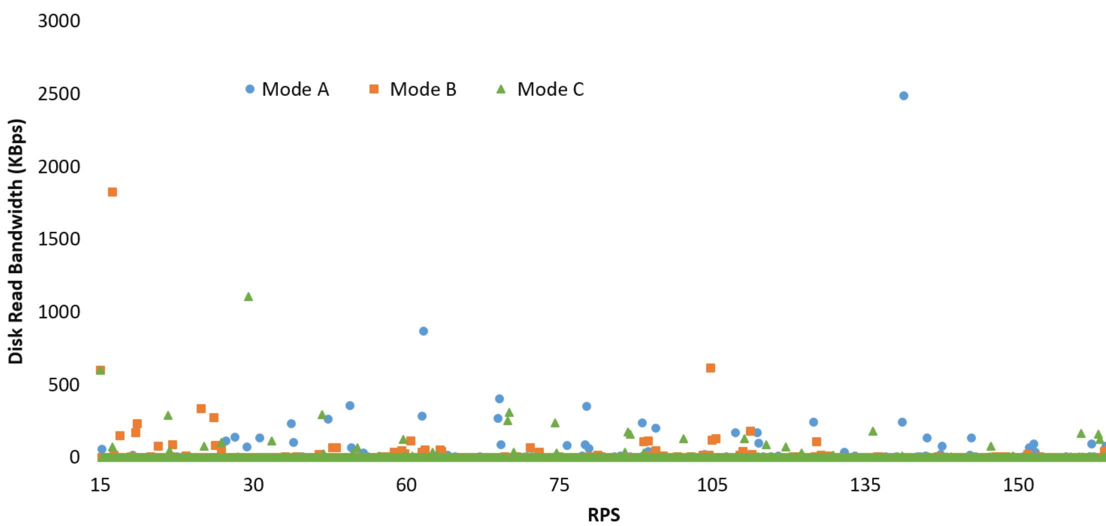

Figure 7.25: Disk Read Bandwidth Usage for Slow Computation Slow Network data point

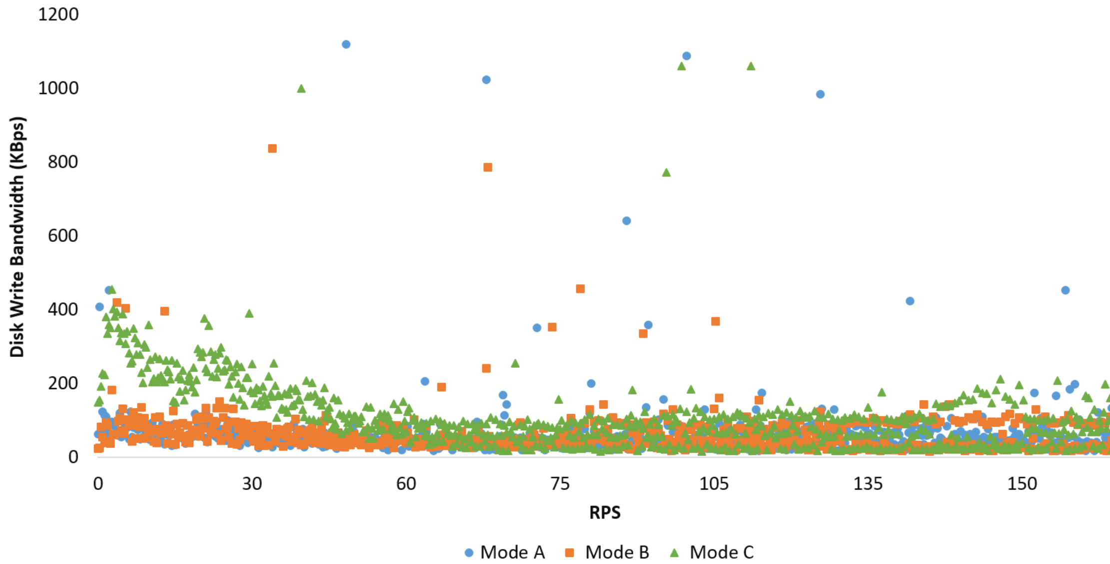

Figure 7.26: Disk Write Bandwidth Usage for Slow Computation Slow Network data point

Figures 7.25 and 7.26 show the disk read and write bandwidth used by the Web server versus the RPS received by it, respectively. The usage pattern for all the modes remained approximately the same.

7.5.2 Performance Evaluation

Using the aforementioned observations for the Slow Computation Slow Network data point in our design space, we can conclude that computational availability determines the performance of the Web server for all the modes. This is because all the modes are bounded by the CPU.

Chapter 8

Conclusions

We conclude this thesis by stating the results obtained from the detailed performance analysis and discussing the ideas for future extension of our research.

8.1 Conclusion

In this thesis, we illustrated the process that is followed to conduct a realistic benchmarking of Web services. During our research, we made an open source contribution to the YCSB framework by designing and implementing a REST module that is capable of generating workloads to benchmark Web services. The YCSB REST module allows researchers to efficiently generate various realistic workloads for Web services using trace files. By using this, we conducted our benchmarking experiments and a detailed performance analysis of two transactional Web archiving solutions – SiteStory and local transactional archiving – described in this thesis. The results show that our proposed system, with local archiving, uses resources more efficiently than SiteStory in the Fast Computation Slow Network data point of our design space. At this data point in our design space, SiteStory breaks at 300 RPS while our local archiving solution breaks at 375 RPS, which is very close to the breaking point of Mode A – 400 RPS. Hence this proves our hypothesis for the Fast Computation Slow Network data point in our design space when Greek Wikipedia is used as the Web Service and step workload is used for the performance benchmarking. Conclusively, we demonstrate that our local archiving solution does not impose any significant performance overheads across the four data points in our design space.

8.2 Future Work

There are many promising, unexplored directions on analyzing and improving the performance of transactional Web archiving systems. In this section we mention a few potential extensions of our research. To begin with, we could implement the *Local archiving using RAM as a Store* solution, which was proposed in Chapter 2. Since Redis is a highly efficient in-memory key-value store, the expected performance overhead of this solution should be minimum. An analysis of this solution in our design space would be interesting to conduct with respect to RAM.

Also, our experiments were confined due to hardware limitations. Thus, a broader design space could show other interesting results. One such design space can be represented by a three dimensional space where CPU, RAM, and network bandwidth are its deciding parameters Another such extension could be to use a variety of Web applications for the performance analysis. For example, instead of using a Web encyclopedia, audio or video streaming, or static resource applications, can be used. These applications have different access patterns and resource usage from Wikipedia. Hence an analysis of these applications could provide a holistic perspective about the implications of SiteStory and our local archiving solution on a Web server.

Bibliography

[1] J. F. Brunelle, M. L. Nelson, L. Balakireva, R. Sanderson, and H. V. de Sampel, "SiteStory." [Online]. Available: `https://mementoweb.github.io/SiteStory/`, 2013. [Accessed: 1 April, 2017].

[2] J. Masanes, *Web Archiving*. Springer Science and Business Media, 2006. ISBN 978-3-540-46332-0.

[3] M. Consalvo and C. Ess, *The Handbook of Internet Studies*. Wiley-Blackwell, 2011. ISBN: 978-1-4051-8588-2.

[4] M. Pennock, *Web-Archiving*. Digital Preservation Coalition, 2011. ISSN: 2048-7916.

[5] D. Gomes, J. Miranda, and M. Costa, "A Survey on Web Archiving Initiatives," in *Proceedings of TPDL 2011*, vol. 6966 of *Lecture Notes in Computer Science*, (Berlin, Germany), Springer, 2011.

[6] J. F. Brunelle1, M. L. Nelson, L. Balakireva, R. Sanderson, and H. V. de Sompel, "Evaluating the SiteStory Transactional Web Archive with the ApacheBench Tool," in *Proceedings of TPDL 2011*, vol. 416 of *Communications in Computer and Information Science*, (Valletta, Malta), Springer, 2013.

[7] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the 1st ACM symposium on Cloud computing 2010*, (Indianapolis, Indiana, USA), Associate for Computing Machinery, 2010.

[8] Z. Xie, P. Chandrasekar, and E. A. Fox, "Using Transactional Web Archives To Handle Server Errors," in *Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries*, (Knoxville, Tennessee, USA), ACM, 2015.

[9] Z. Xie, K. Nayyar, and E. A. Fox, "Nearline Web Archiving." [Online]. Available: `https://vtechworks.lib.vt.edu/bitstream/handle/10919/71648/2016-WADL-nearline.pdf?sequence=1`, 2016. [Accessed: 1 June, 2016].

[10] H. V. de Sompel, M. Nelson, and R. Sanderson, "Memento protocol." [Online]. Available: `http://mementoweb.org/guide/rfc/`, 2013. [Accessed: 1 April, 2017].

[11] WARC, "WARC." [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000236.shtml, 2009. [Accessed: 1 April, 2016].

[12] Archive.org, "WayBack machine." [Online]. Available: https://archive.org/web/, 2014. [Accessed: 1 April, 2017].

[13] Oracle Corporation, "Berkeley database." [Online]. Available: http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html, 1994. [Accessed: 1 April, 2017].

[14] The Apache Software Foundation, "Apache APXS." [Online]. Available: https://httpd.apache.org/docs/2.4/programs/apxs.html, 1995. [Accessed: 1 April, 2017].

[15] The Apache Software Foundation, "Apache Tomcat." [Online]. Available: http://tomcat.apache.org/, 1999. [Accessed: 1 April, 2017].

[16] S. Sanfilippo, "Redis." [Online]. Available: https://redis.io/, 2009. [Accessed: 1 April, 2017].

[17] A. Lakshman and P. Malik, "Apache Cassandra." [Online]. Available: http://cassandra.apache.org/, 2008. [Accessed: 1 April, 2017].

[18] Oracle Corporation, "MySQL." [Online]. Available: https://www.mysql.com/, 1995. [Accessed: 1 April, 2017].

[19] The Apache Software Foundation, "Apache Jmeter." [Online]. Available: http://jmeter.apache.org, 1998. [Accessed: 1 April, 2017].

[20] The Apache Software Foundation, "Apache Benchmark." [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html, 2004. [Accessed: 1 April, 2017].

[21] D. Mosberger and T. Jin, "A Tool for Measuring Web Server Performance," in *Proceedings of the Internet Server Performance Workshop 1998*, vol. 26 of *Performance Evaluation Review*, (Madison, Wisconsin, USA), ACM, 1998.

[22] N. Niclausse, "Tsung." [Online]. Available: http://tsung.erlang-projects.org, 2002. [Accessed: 1 April, 2017].

[23] E. T. Bray, "JavaScript Object Notation." [Online]. Available: http://www.json.org, 2013. [Accessed: 1 April, 2017].

[24] World Wide Web Consortium and Web Hypertext Application Technology Working Group, "Hypertext Markup Language." [Online]. Available: https://www.w3.org/html/, 1993. [Accessed: 1 April, 2017].

[25] YCSB, "Apache Accumulo." [Online]. Available: https://accumulo.apache.org, 2008. [Accessed: 1 April, 2017].

[26] Wikipedia.com, "Computer performance." [Online]. Available: https://en.wikipedia.org/wiki/Computer_performance. [Accessed: 1 April, 2017].

[27] S. Sanfilippo, "Redis RDB Tools." [Online]. Available: https://redis.io/topics/persistence, 2012. [Accessed: 1 April, 2017].

[28] Wikimedia Foundation, "Greek Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Greek_Wikipedia, 2002. [Accessed: 10 Dec, 2015].

[29] Wikimedia Foundation, "Wikipedia Statistics." [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:Statistics. [Accessed: 10 Dec, 2015].

[30] M. Manske, L. D. Crocker, W. Foundation, and M. volunteers, "MediaWiki." [Online]. Available: https://www.mediawiki.org/wiki/MediaWiki, 2002. [Accessed: 10 Dec, 2015].

[31] Z. T. Rasmus Lerdorf, The PHP Development Team, "PHP." [Online]. Available: http://php.net, 1995. [Accessed: 1 June, 2016].

[32] Wikimedia Foundation, "English Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/English_Wikipedia, 2001. [Accessed: 10 Dec, 2015].

[33] Wikimedia Foundation, "MediaWiki Dumper." [Online]. Available: https://www.mediawiki.org/wiki/Manual:MWDumper, 2012. [Accessed: 10 Dec, 2015].

[34] Wikimedia Foundation, "MediaWiki Import Dump." [Online]. Available: https://www.mediawiki.org/wiki/Manual:ImportDump.php, 2012. [Accessed: 10 Dec, 2015].

[35] Wikimedia Foundation, "Greek Wikipedia data dump." [Online]. Available: http://archive.org/download/elwiki-20151201/elwiki-20151201-pages-meta-current.xml.bz2, 2015. [Accessed: 10 Dec, 2015].

[36] Wikimedia Foundation, "Greek Wikipedia Statistics." [Online]. Available: https://stats.wikimedia.org/EN/TablesWikipediaEL.htm. [Accessed: 10 Dec, 2015].

[37] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications," in *Proceedings of the IEEE Computer and Communications Societies. 1998*, vol. 26 of *Performance Evaluation Review*, (San Francisco, CA, USA), IEEE Computer Society Press, 1999.

[38] L. A. Adamic and B. A. Huberman, "Zipf's law and the Internet," in *Proceedings of the Glottometrics. 2002*, vol. 3, (Ludenscheid, Germany), RAM-Verlag, 2002.

[39] Wikimedia Foundation, "Wikipedia:Does Wikipedia traffic obey Zipf's law?." [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:Does_Wikipedia_traffic_obey_Zipf%27s_law%3F. [Accessed: 10 Dec, 2015].

[40] Wikimedia Foundation, "Wikipedia article size." [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:Article_size. [Accessed: 10 Dec, 2015].

[41] Wikimedia Foundation, "Uniform distribution." [Online]. Available: https://en.wikipedia.org/wiki/Uniform_distribution_(continuous). [Accessed: 10 Dec, 2015].

[42] Sun Microsystems, "HashMap Java implementation." [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html. [Accessed: 1 Dec, 2016].

[43] The Apache Software Foundation, "Apache HTTP Client 3.x API." [Online]. Available: https://hc.apache.org/httpclient-3.x/apidocs/, 2011. [Accessed: 1 March, 2016].

[44] S. Banon, "Elasticsearch." [Online]. Available: https://www.elastic.co/, 2010. [Accessed: 1 June, 2016].

[45] Elastic Search Development Community, "Kibana." [Online]. Available: https://www.elastic.co/products/kibana, 2013. [Accessed: 1 June, 2016].

[46] M. Seger, "Collectl." [Online]. Available: http://collectl.sourceforge.net/index.html, 2007. [Accessed: 1 June, 2016].

[47] Elastic Search Development Community, "Logstash." [Online]. Available: https://www.elastic.co/products/logstash, 2010. [Accessed: 1 June, 2016].

[48] J. Watkins, "PHP APC user cache." [Online]. Available: http://php.net/manual/en/book.apcu.php, 2013. [Accessed: 1 June, 2016].

Appendix A

Results for Additional Modes

A.1 Running SiteStory Locally - Mode D

Appendix A.1 shows the benchmarking results of Mode D for all data points in our design space. This mode is similar to Mode C, with one difference. Instead of sending the Web archiving HTTP PUT requests to our local archiving service, we run the SiteStory Web archiving Apache Tomcat server locally and configure the Apache SiteStory module to send these requests to it. The aim of benchmarking this mode was to compare the performance of our local transactional Web archiving with persistent files solution with the SiteStory Web archiving service running locally. We predicted that running an Apache Tomcat server on the same machine would steal CPU cycles from the Apache Web server at heavy workloads thereby degrading its performance. In this section we show that running the SiteStory service locally indeed imposes a significant overhead on the Apache Web server that our local archiving solution does not.

A.1.1 Fast Computation Fast Network

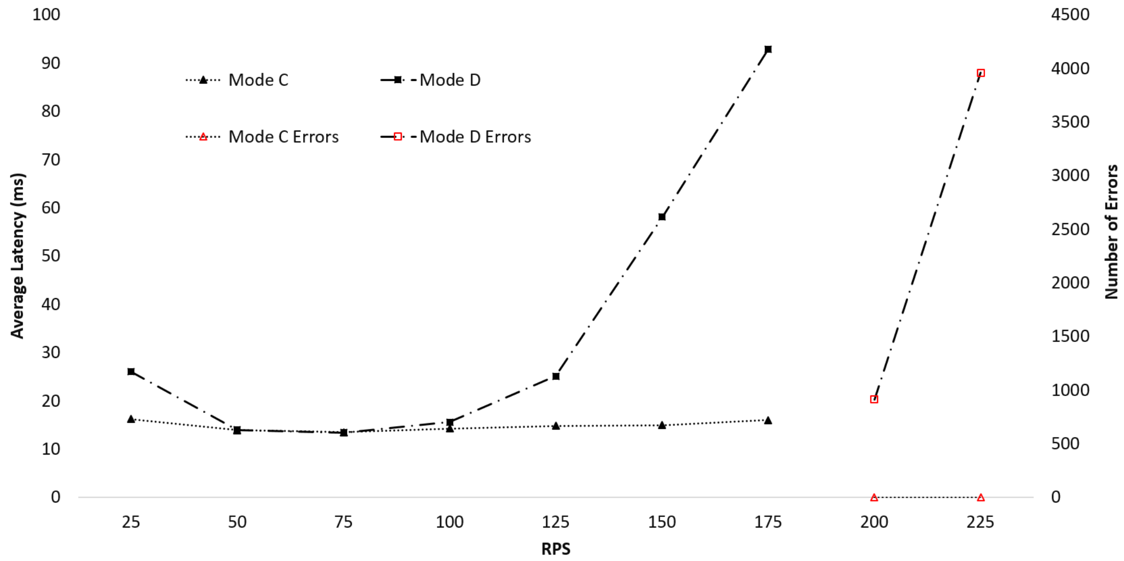

Figure A.1: Average Latency and Errors versus RPS for Fast Computation Fast Network data point

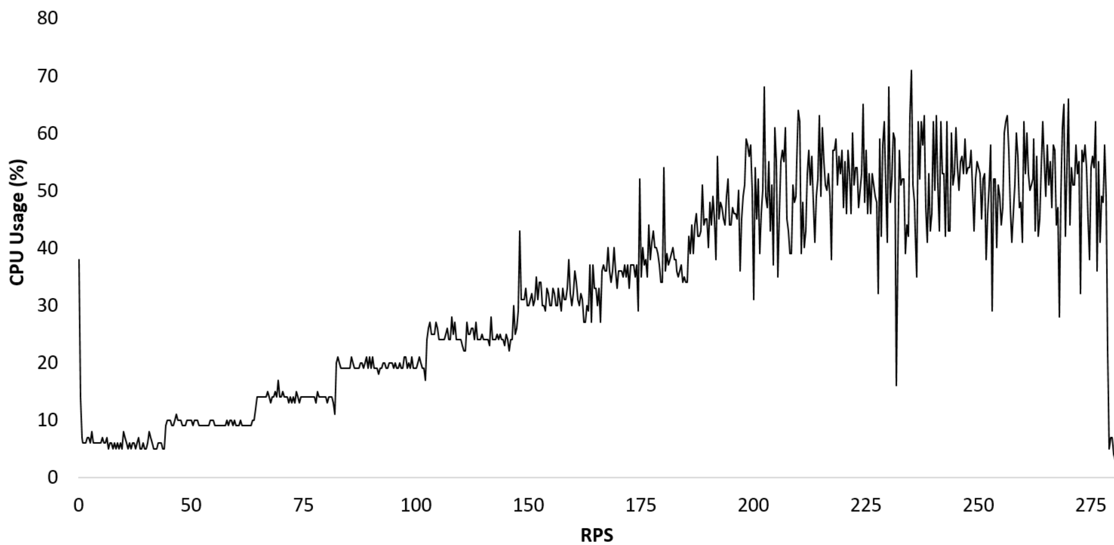

Figure A.2: CPU Usage for Fast Computation Fast Network data point

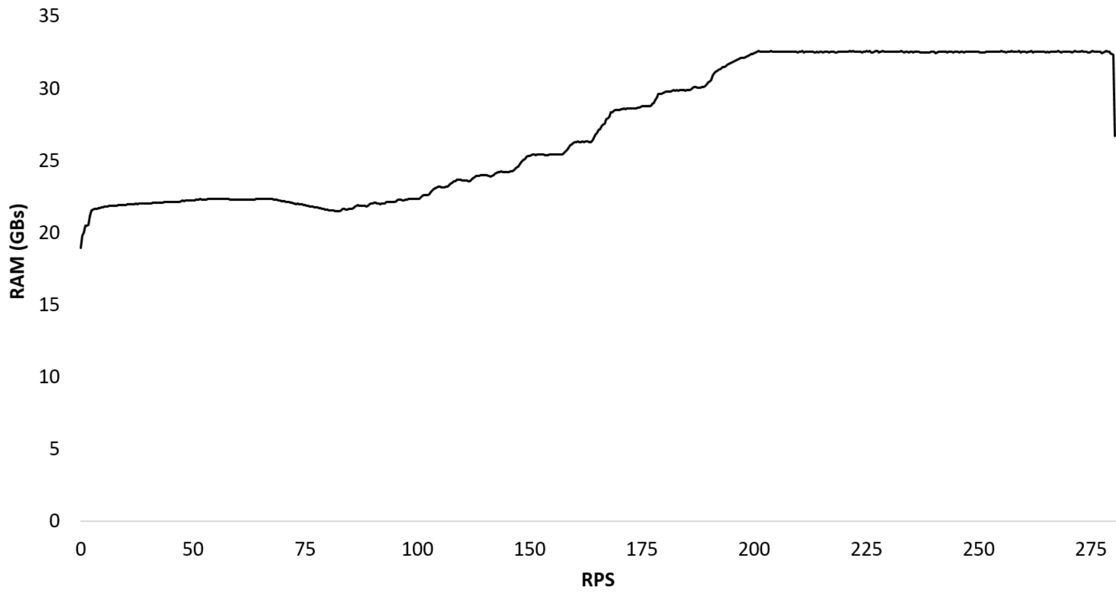

Figure A.3: RAM Usage for Fast Computation Fast Network data point

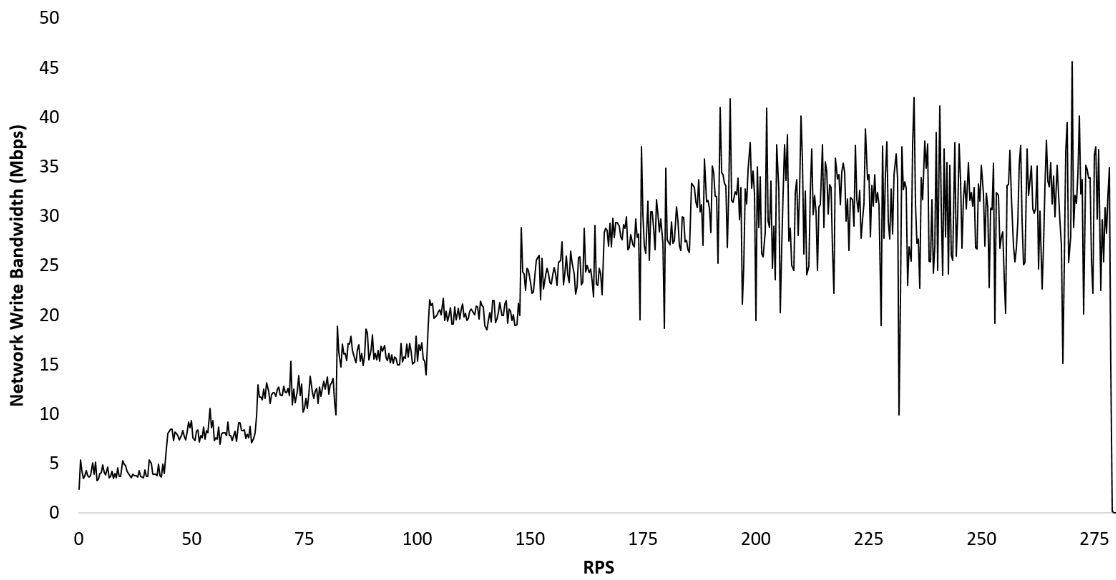

Figure A.4: Network Write Bandwidth Usage for Fast Computation Fast Network data point

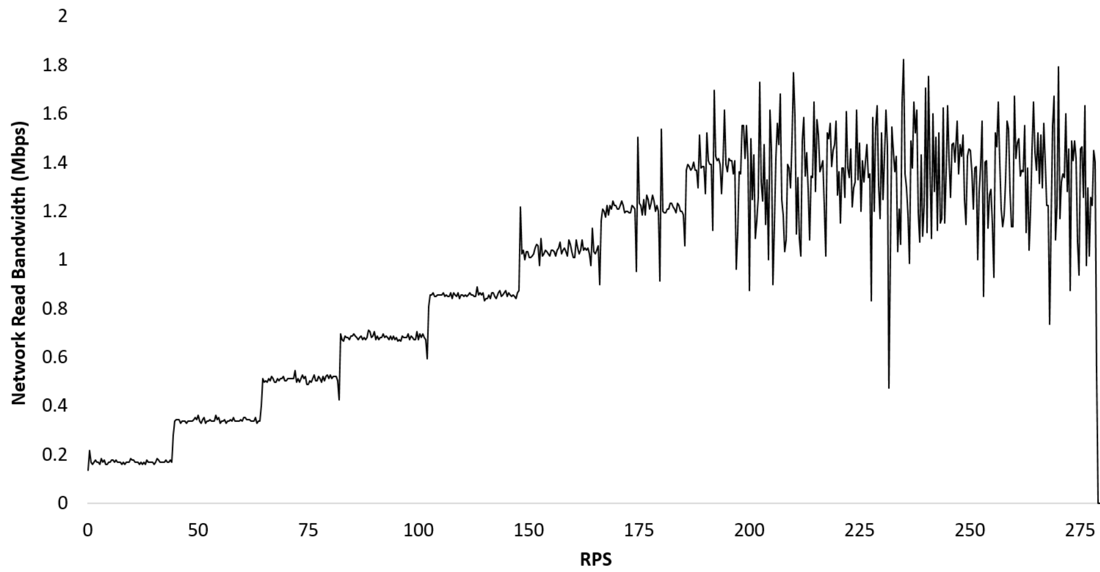

Figure A.5: Network Read Bandwidth Usage for Fast Computation Fast Network data point

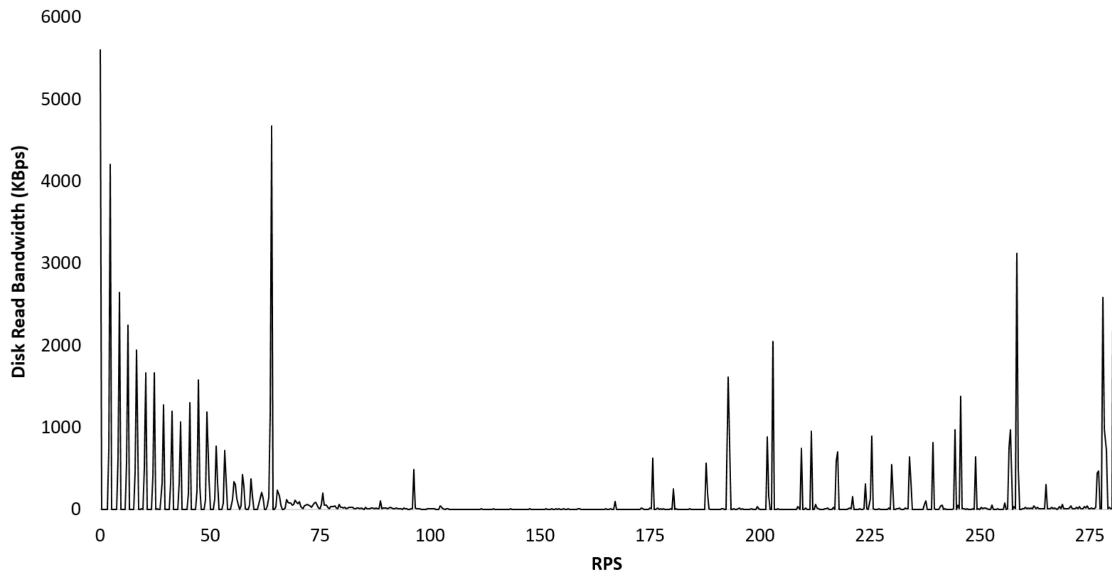

Figure A.6: Disk Read Bandwidth Usage for Fast Computation Fast Network data point

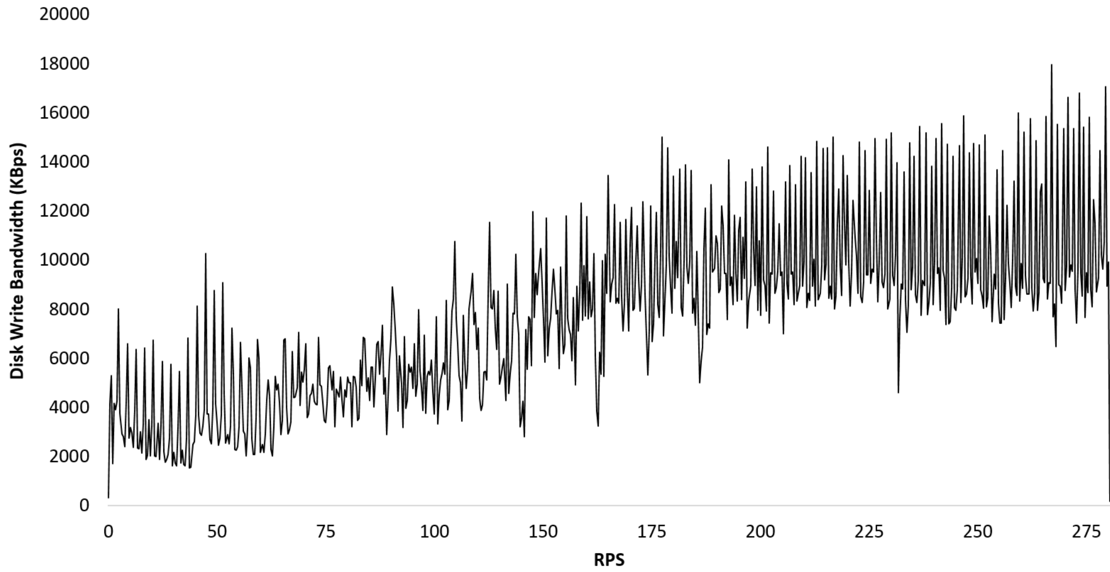

Figure A.7: Disk Write Bandwidth Usage for Fast Computation Fast Network data point

A.1.2 Fast Computation Slow Network

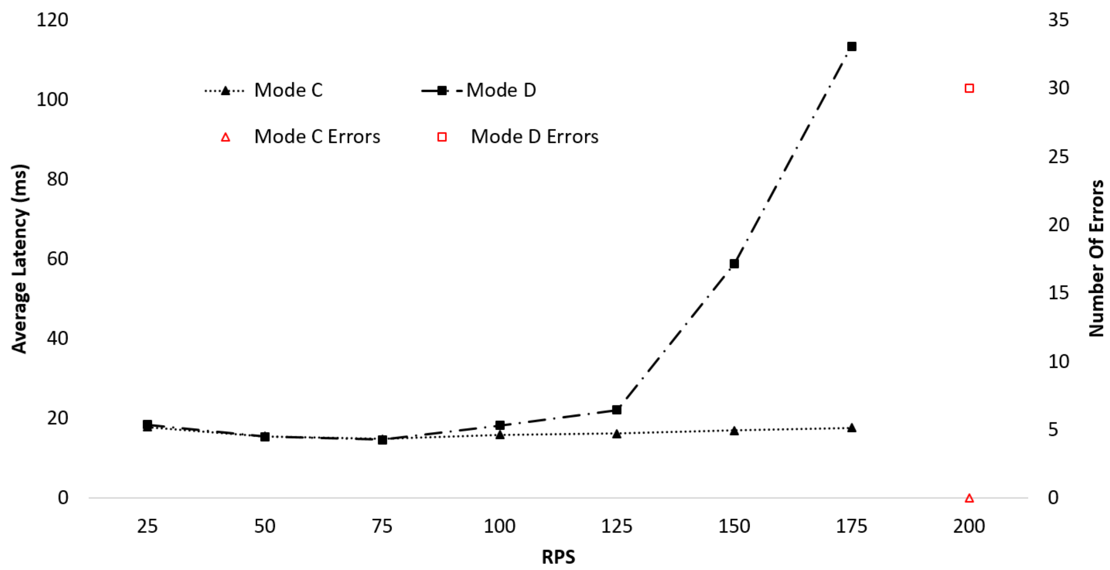

Figure A.8: Average Latency and Errors versus RPS for Fast Computation Slow Network data point

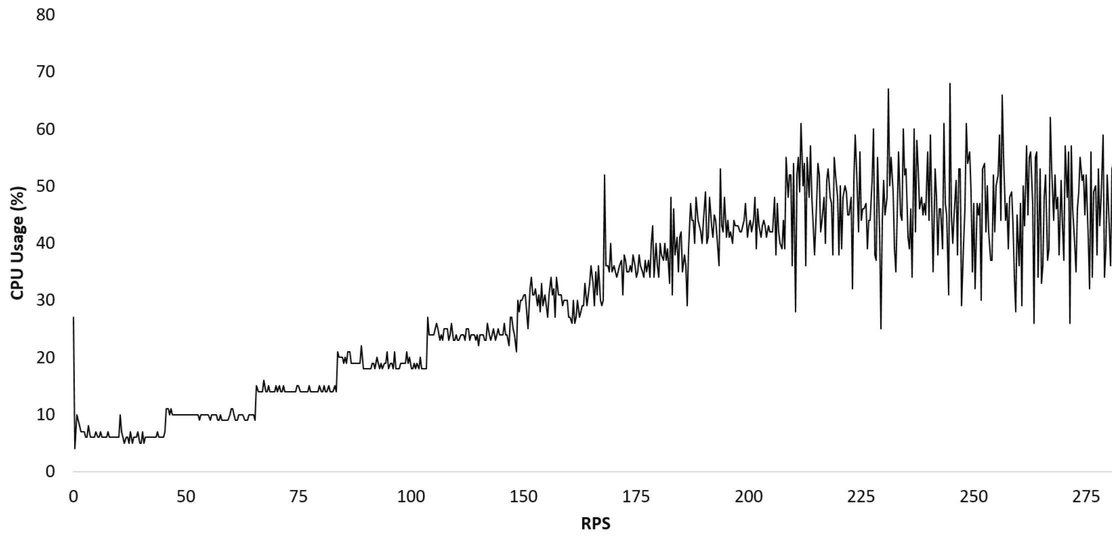

Figure A.9: CPU Usage for Fast Computation Slow Network data point

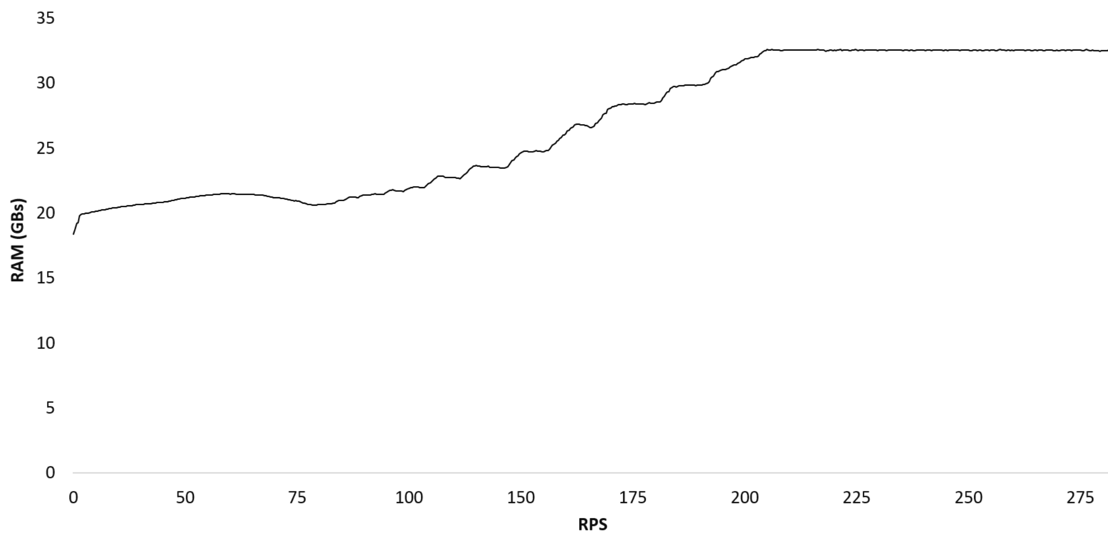

Figure A.10: RAM Usage for Fast Computation Slow Network data point

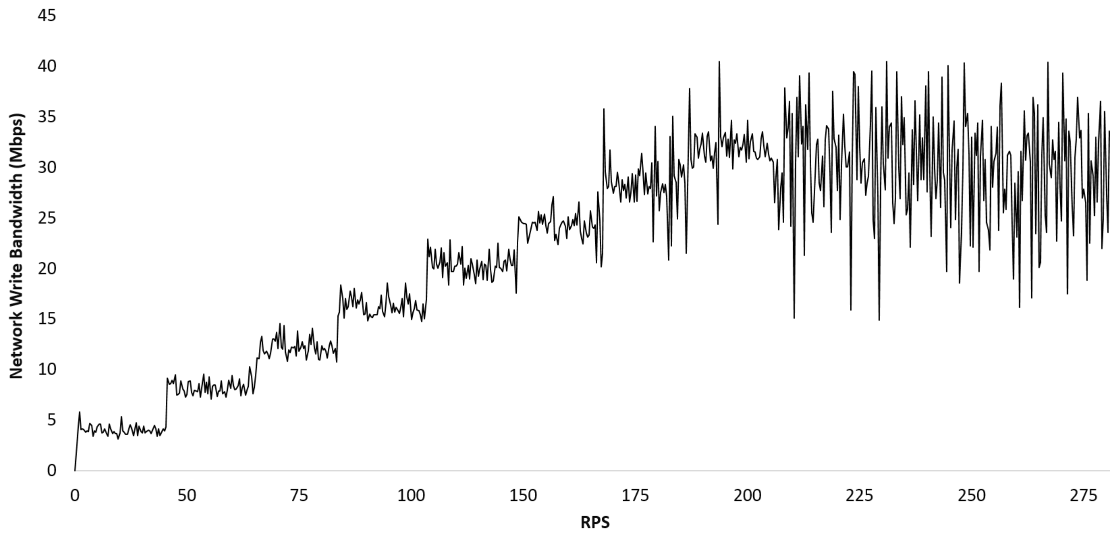

Figure A.11: Network Write Bandwidth Usage for Fast Computation Slow Network data point

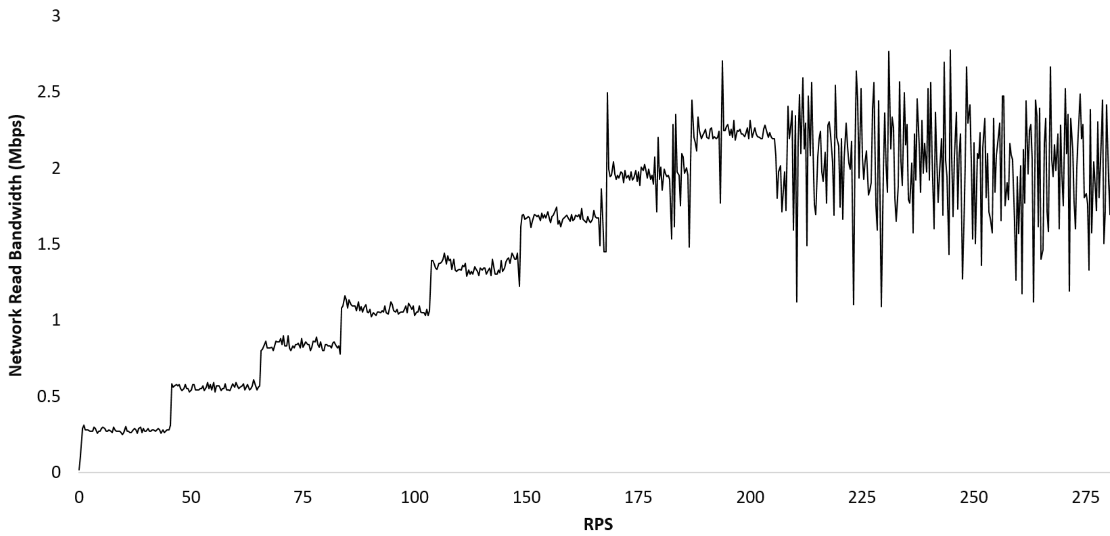

Figure A.12: Network Read Bandwidth Usage for Fast Computation Slow Network data point

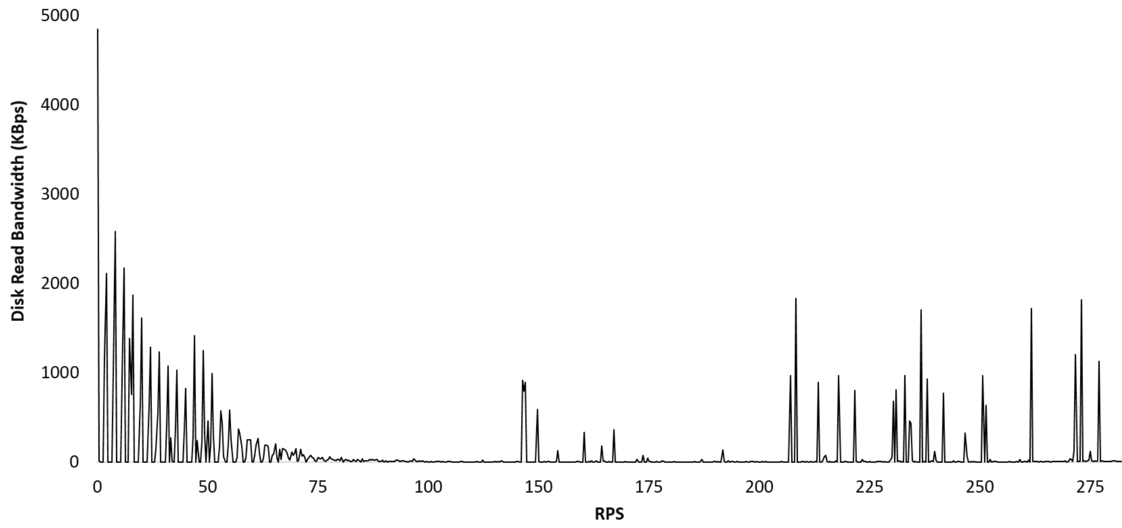

Figure A.13: Disk Read Bandwidth Usage for Fast Computation Slow Network data point

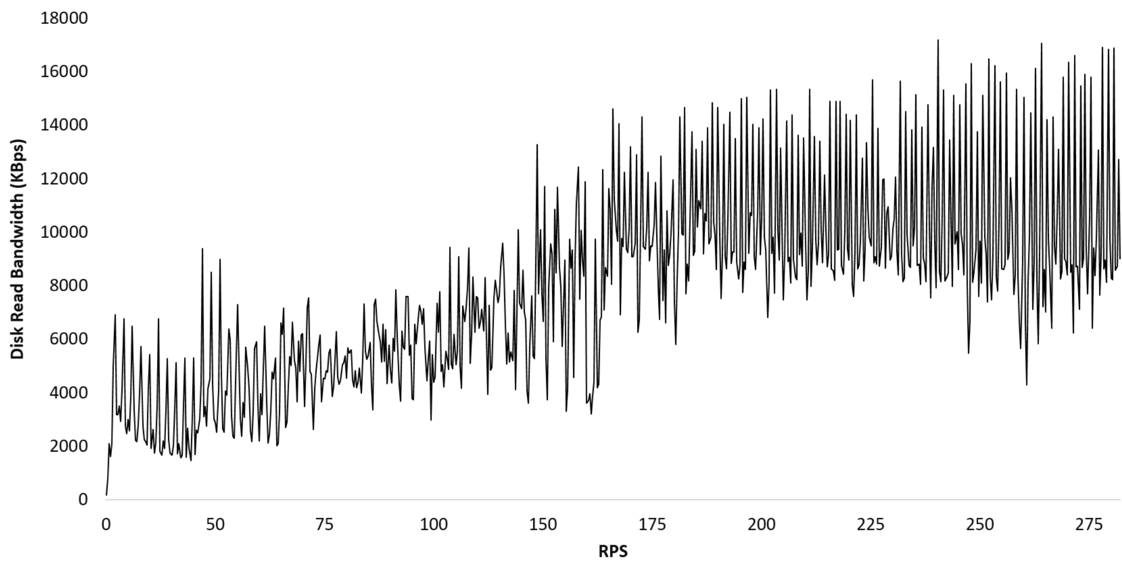

Figure A.14: Disk Write Bandwidth Usage for Fast Computation Slow Network data point

A.1.3 Slow Computation Fast Network

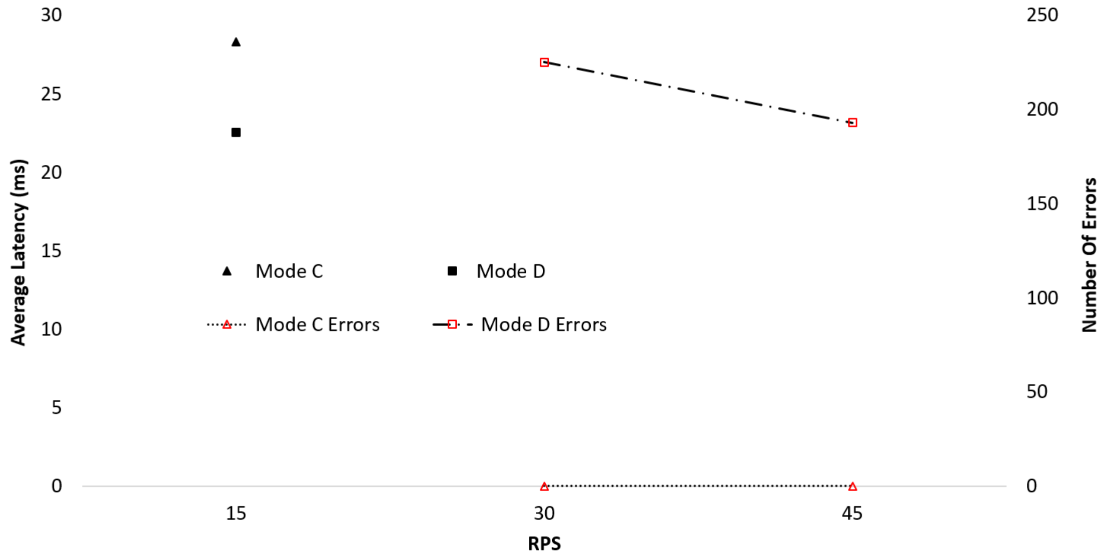

Figure A.15: Average Latency and Errors versus RPS for Slow Computation Fast Network data point

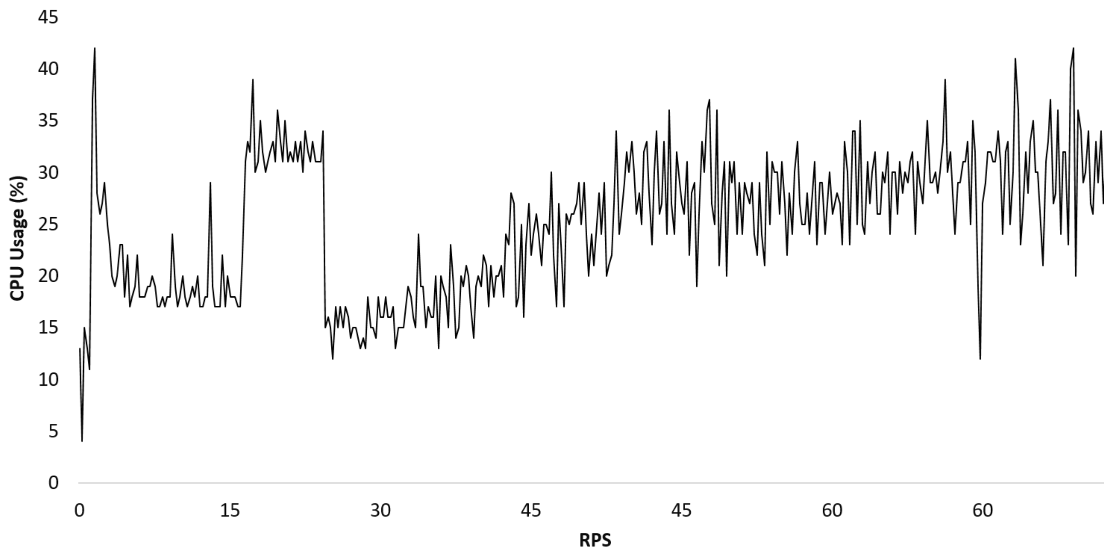

Figure A.16: CPU Usage for Slow Computation Fast Network data point

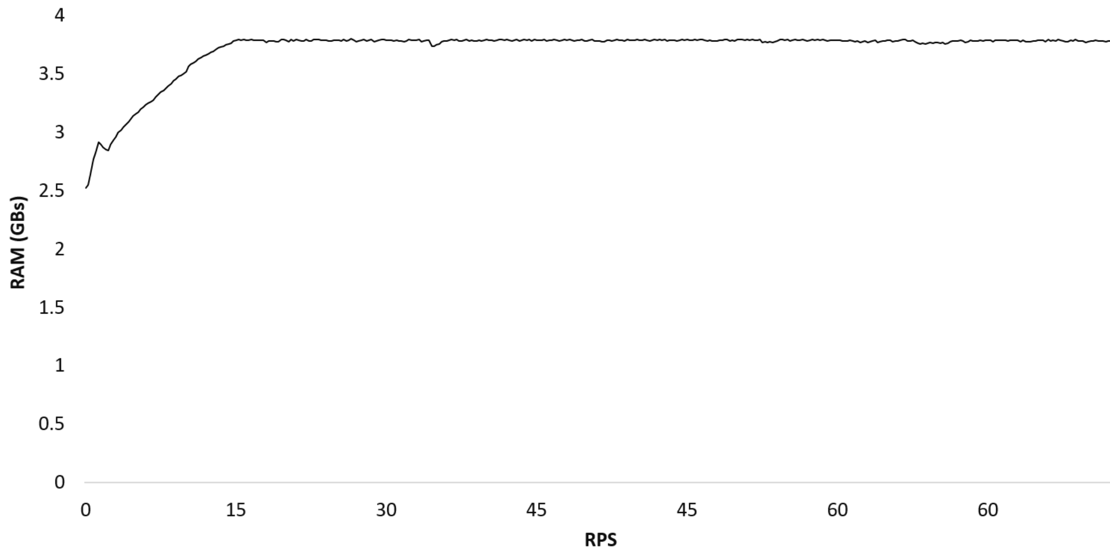

Figure A.17: RAM Usage for Slow Computation Fast Network data point

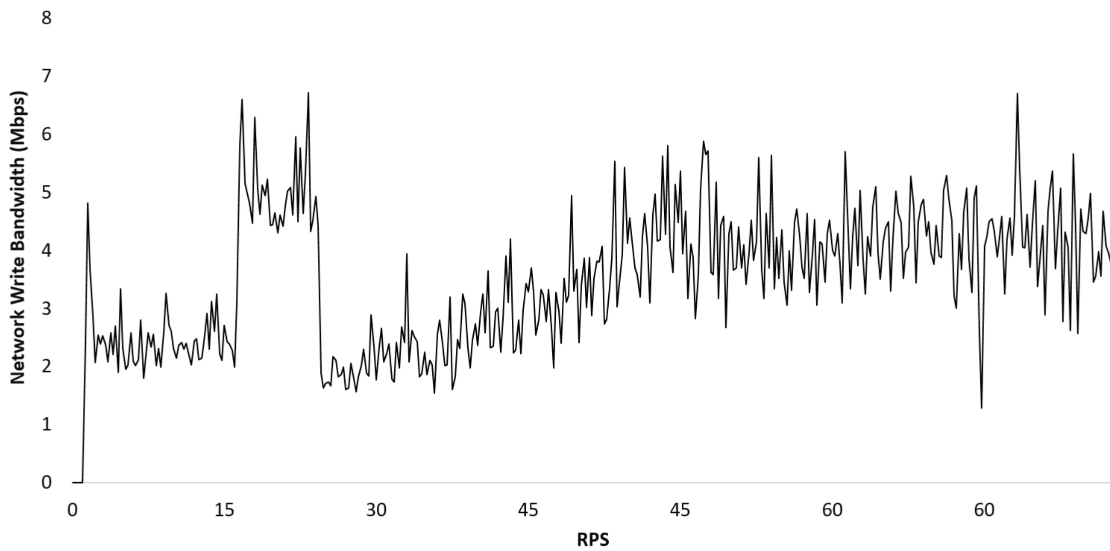

Figure A.18: Network Write Bandwidth Usage for Slow Computation Fast Network data point

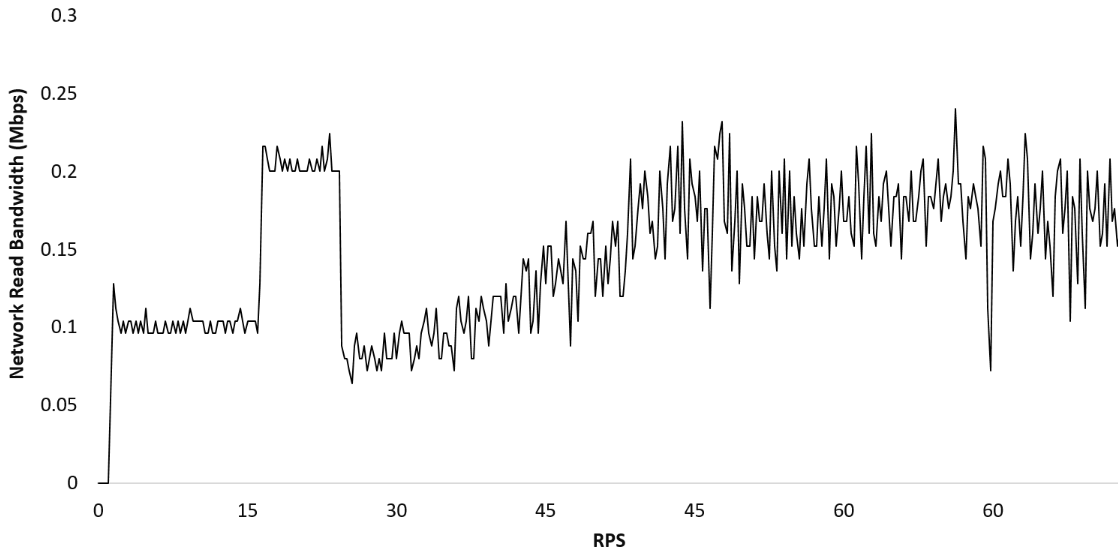

Figure A.19: Network Read Bandwidth Usage for Slow Computation Fast Network data point

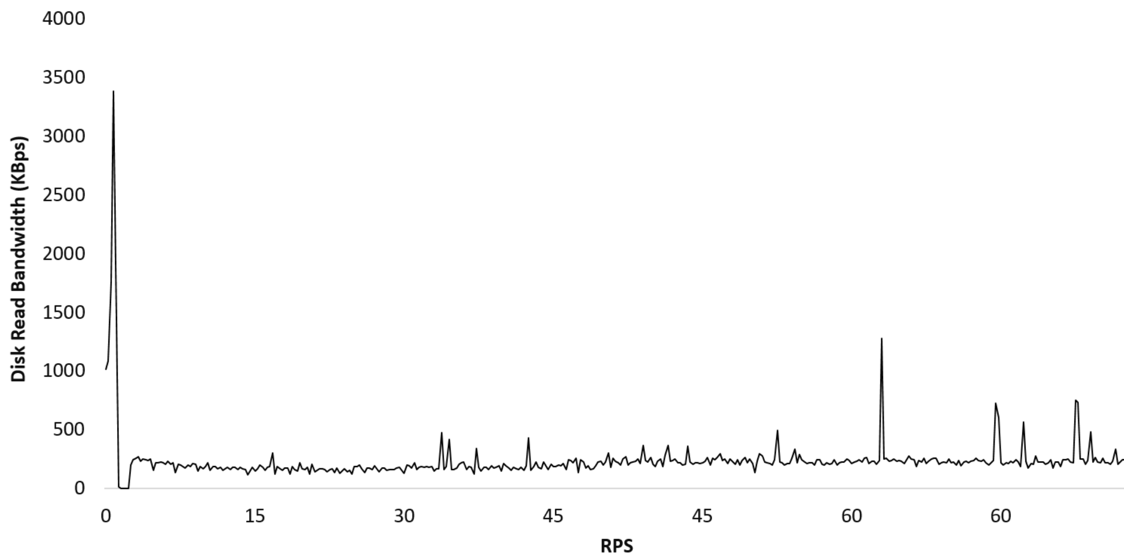

Figure A.20: Disk Read Bandwidth Usage for Slow Computation Fast Network data point

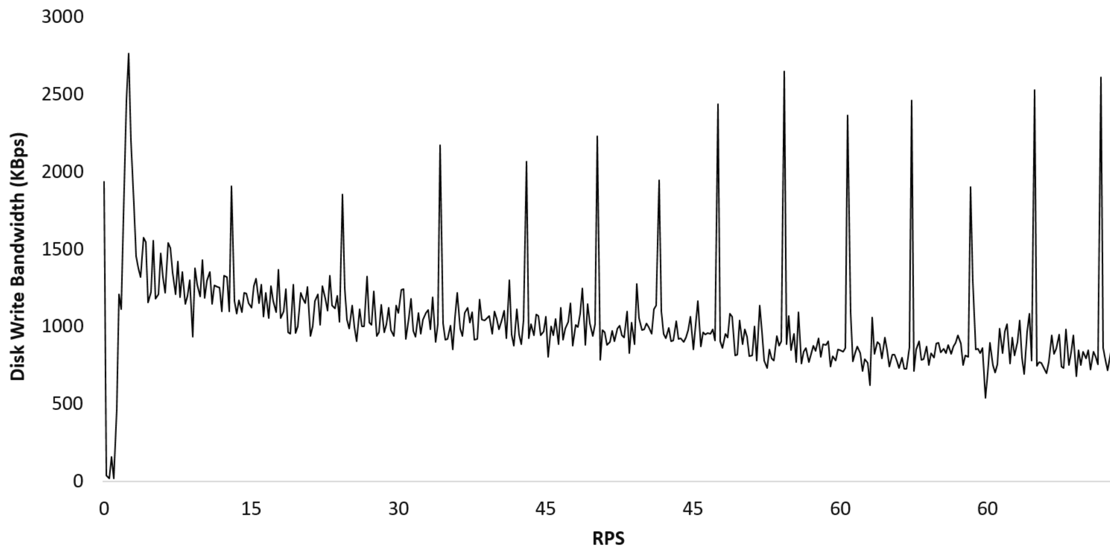

Figure A.21: Disk Write Bandwidth Usage for Slow Computation Fast Network data point

A.1.4 Slow Computation Slow Network

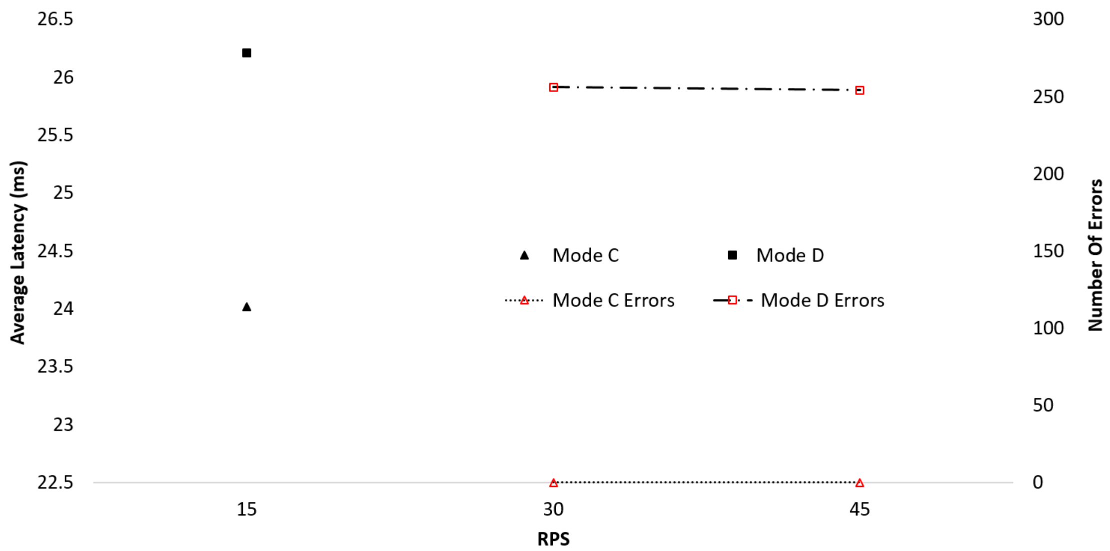

Figure A.22: Average Latency and Errors versus RPS for Slow Computation Slow Network data point

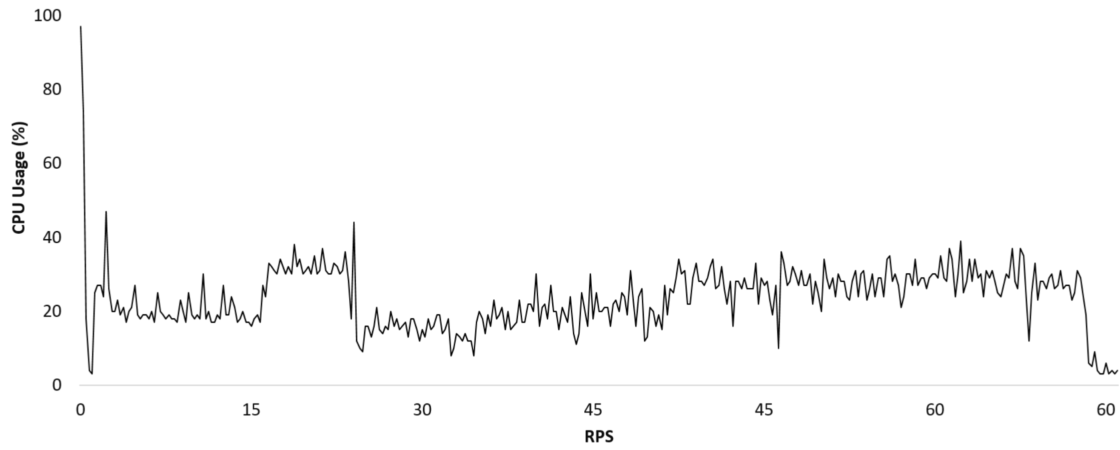

Figure A.23: CPU Usage for Slow Computation Slow Network data point

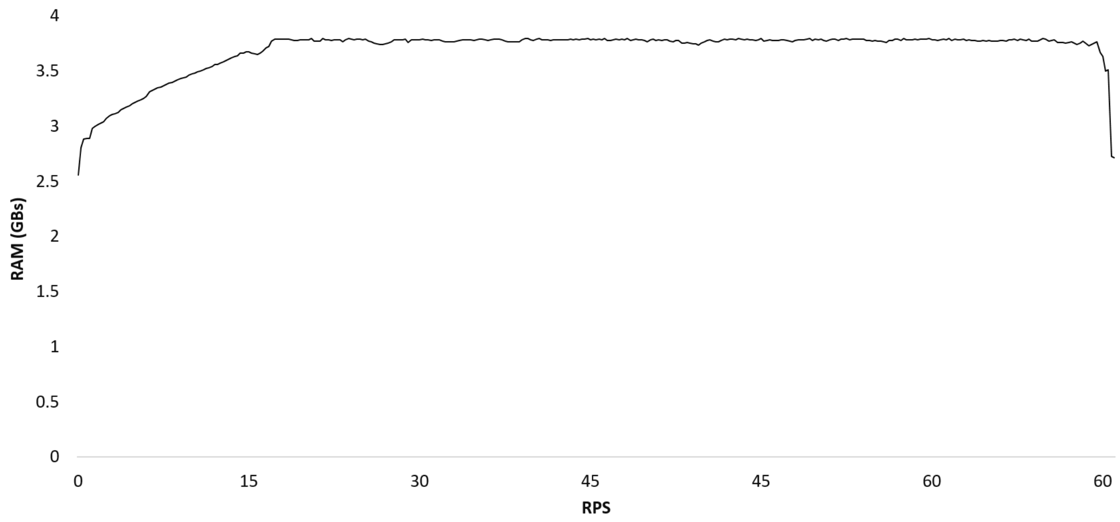

Figure A.24: RAM Usage for Slow Computation Slow Network data point

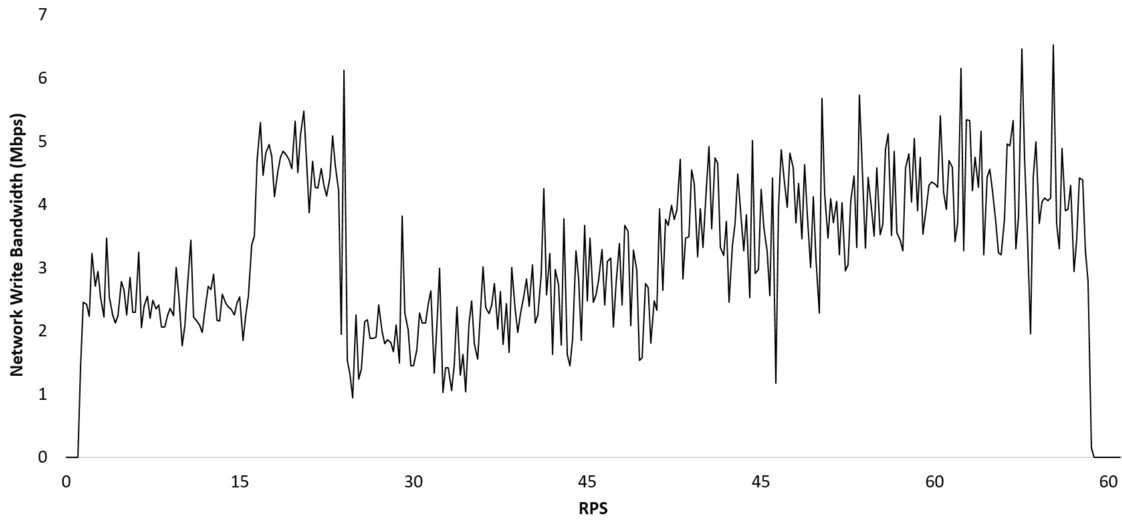

Figure A.25: Network Write Bandwidth Usage for Slow Computation Slow Network data point

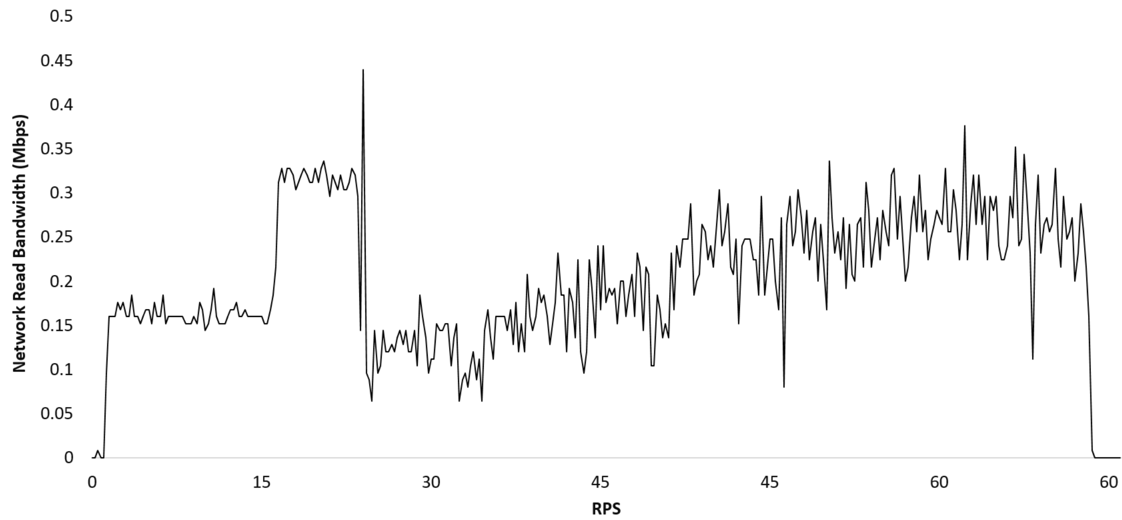

Figure A.26: Network Read Bandwidth Usage for Slow Computation Slow Network data point

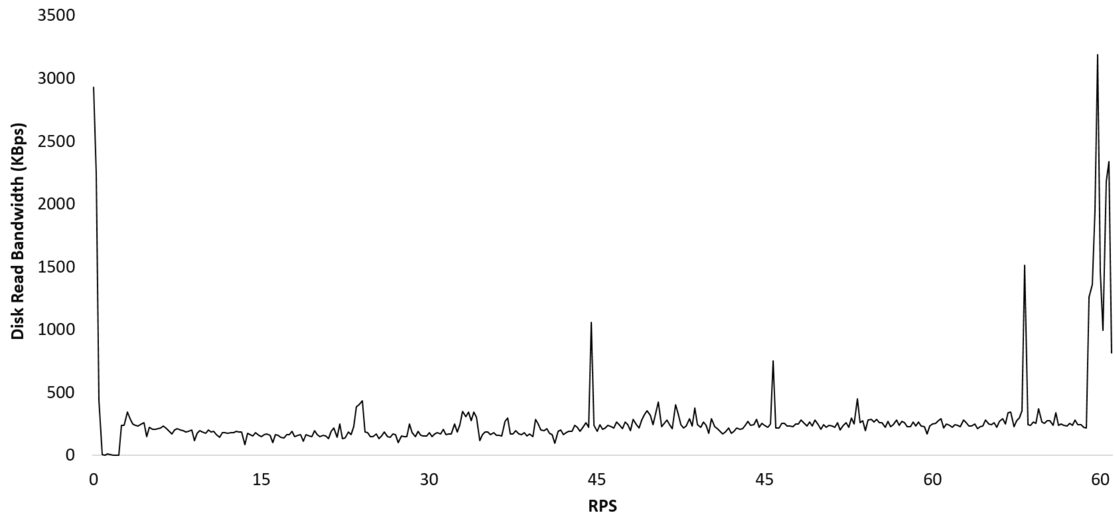

Figure A.27: Disk Read Bandwidth Usage for Slow Computation Slow Network data point

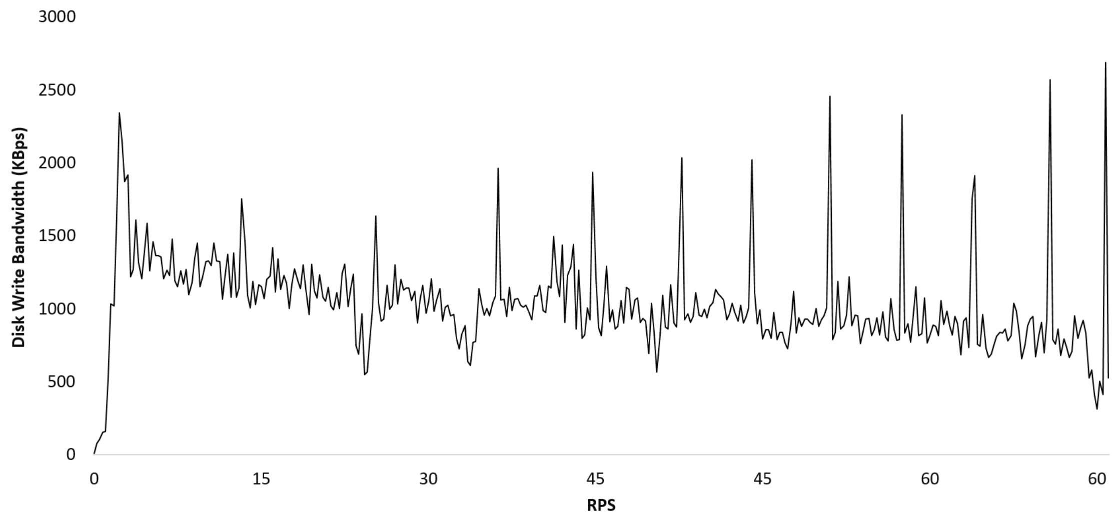

Figure A.28: Disk Write Bandwidth Usage for Slow Computation Slow Network data point

A.2 Using RAM as a Store Local Archiving Technique - Mode E

Appendix A.2 shows the benchmarking results of Mode E for all data points in our design space. This mode is similar to Mode C, with the difference that here instead of using our local transactional Web archiving with persistent files solution we use the RAM as a store implementation to archive the incoming HTTP PUT requests. The aim of benchmarking

this mode was to compare the performance of these two local transational Web archiving techniques and choose the more optimal implementation for performance comparison with Mode A and Mode B. In this section we show that using RAM as a store local archiving technique imposes more overhead on the Apache Web server than the local archiving using the persistent files solution.

A.2.1 Fast Computation Fast Network

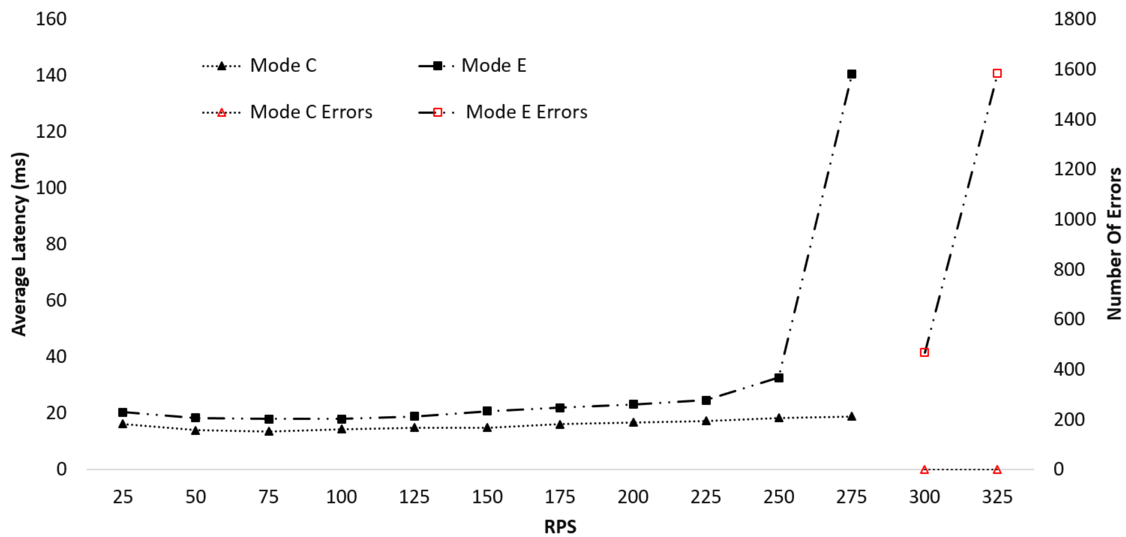

Figure A.29: Average Latency and Errors versus RPS for Fast Computation Fast Network data point

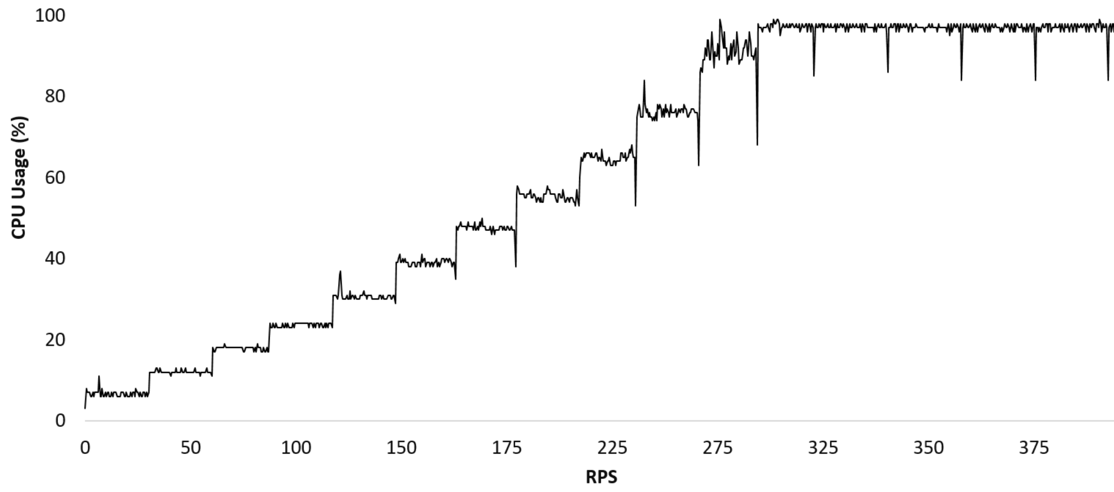

Figure A.30: CPU Usage for Fast Computation Fast Network data point

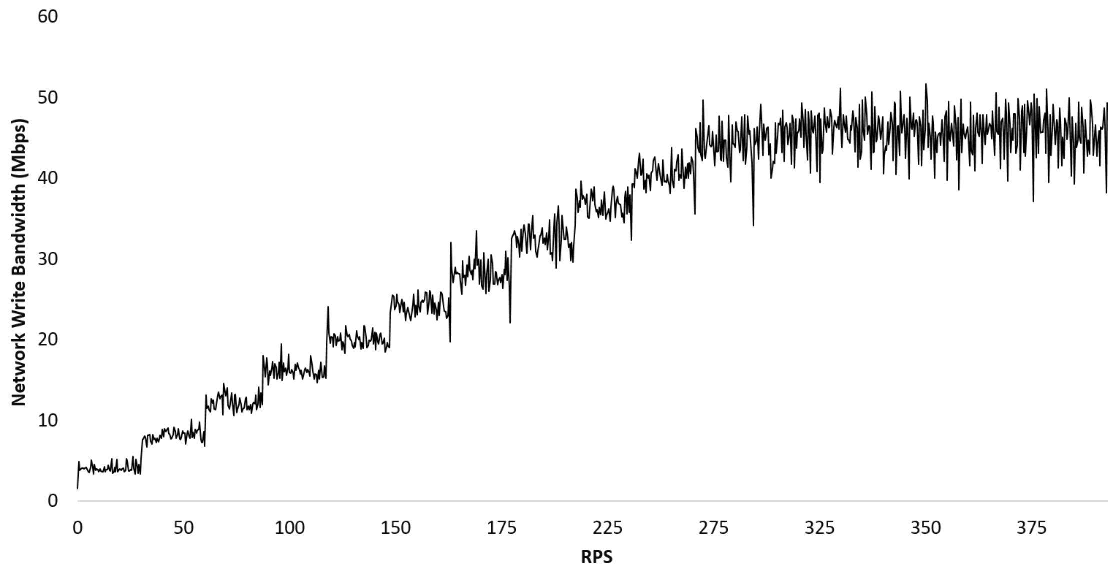

Figure A.31: Network Write Bandwidth Usage for Fast Computation Fast Network data point

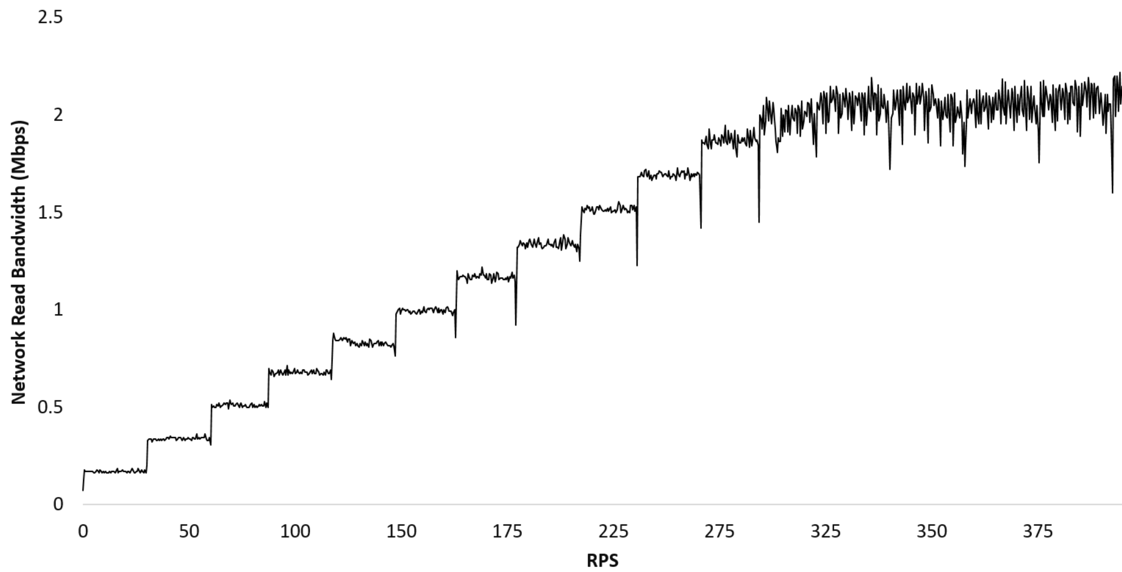

Figure A.32: Network Read Bandwidth Usage for Fast Computation Fast Network data point

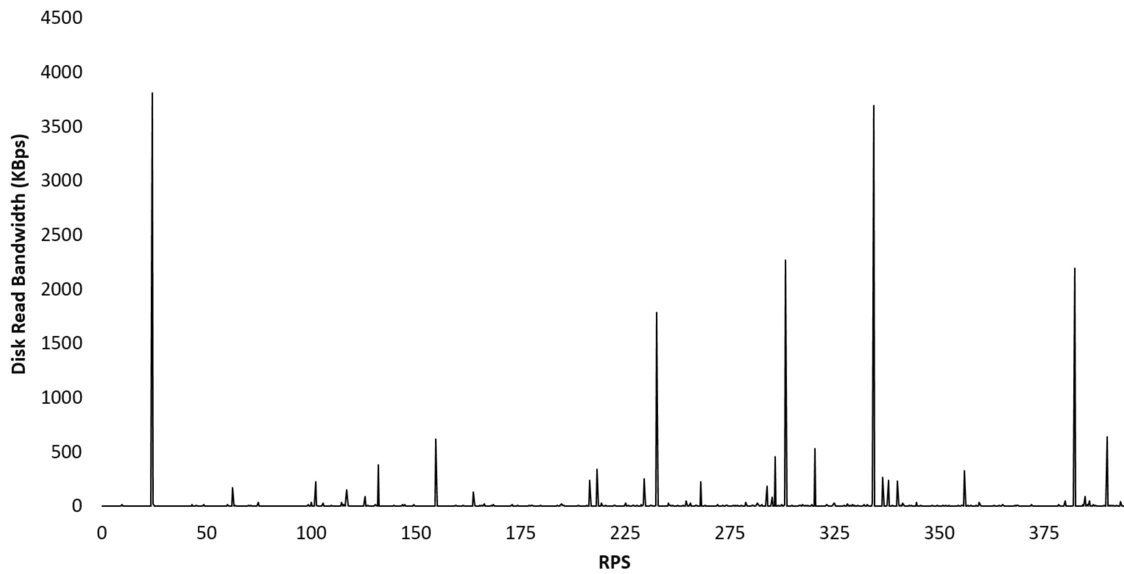

Figure A.33: Disk Read Bandwidth Usage for Fast Computation Fast Network data point

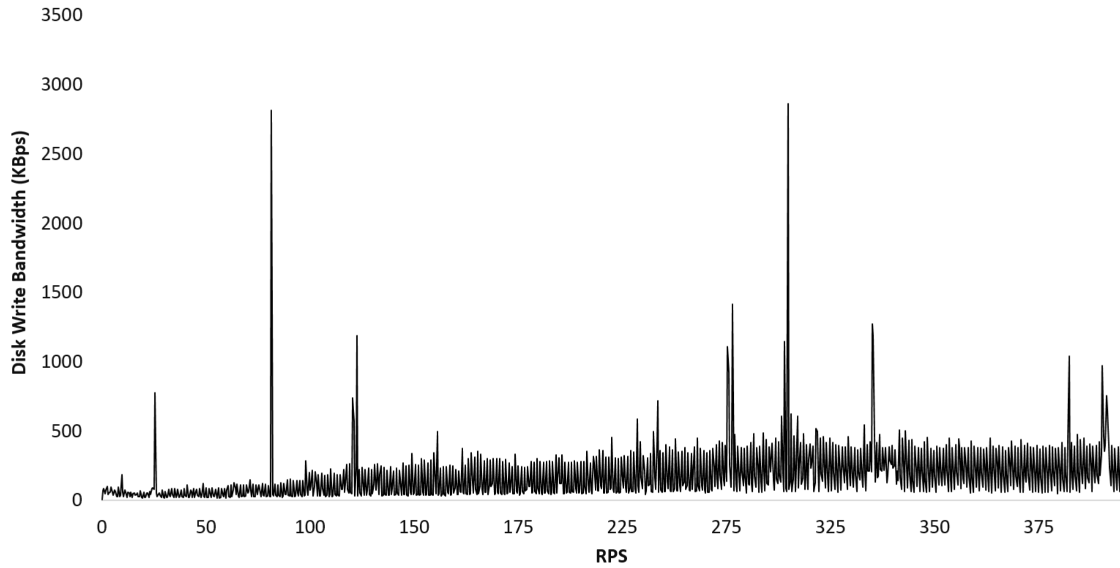

Figure A.34: Disk Write Bandwidth Usage for Fast Computation Fast Network data point

A.2.2 Fast Computation Slow Network

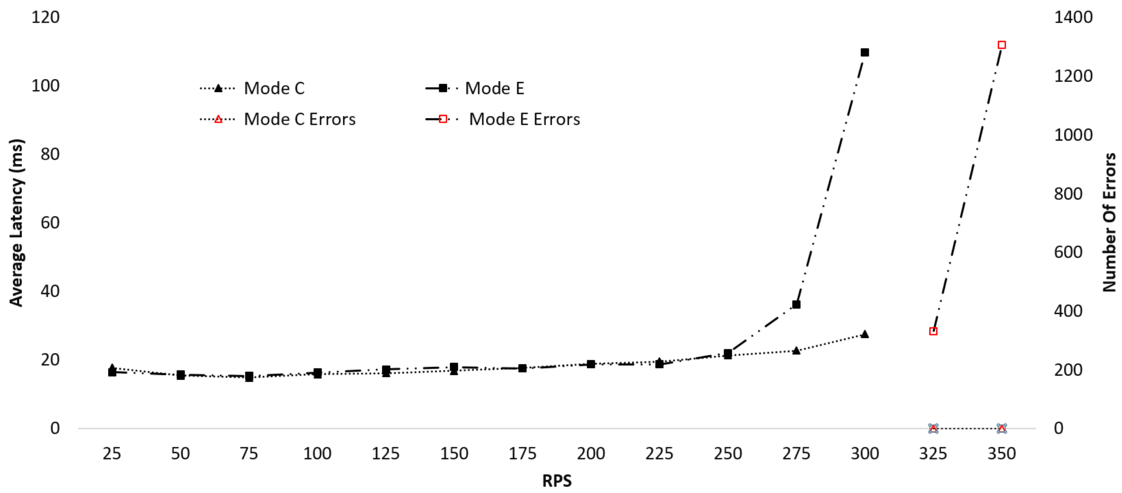

Figure A.35: Average Latency and Errors versus RPS for Fast Computation Slow Network data point

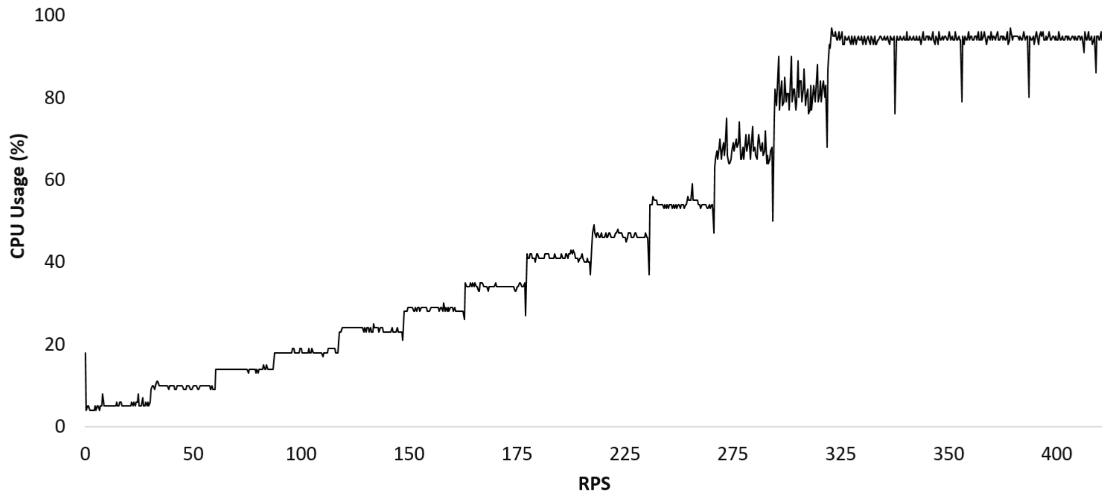

Figure A.36: CPU Usage for Fast Computation Slow Network data point

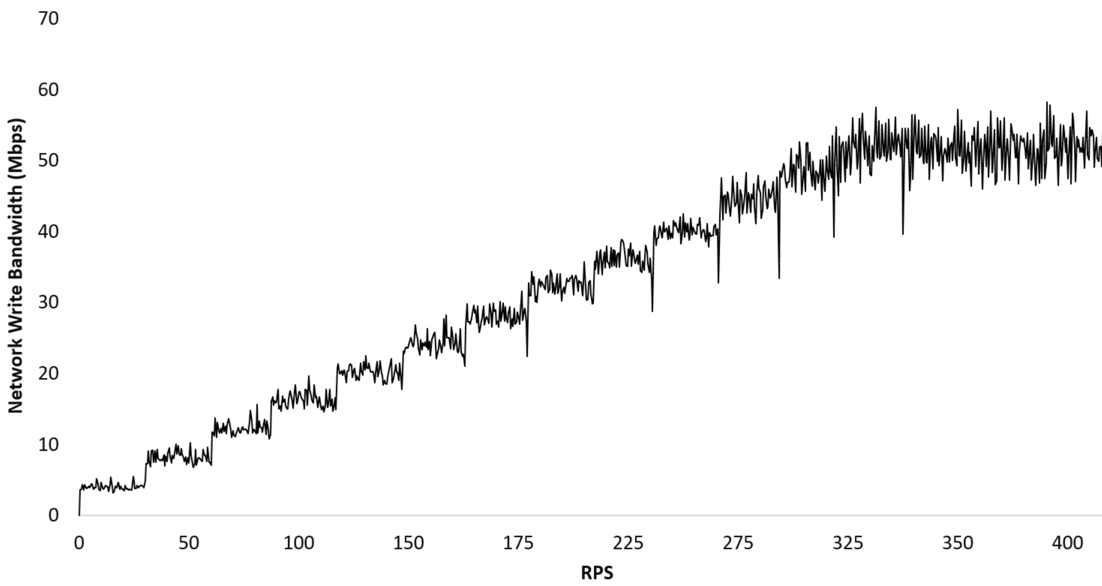

Figure A.37: Network Write Bandwidth Usage for Fast Computation Slow Network data point

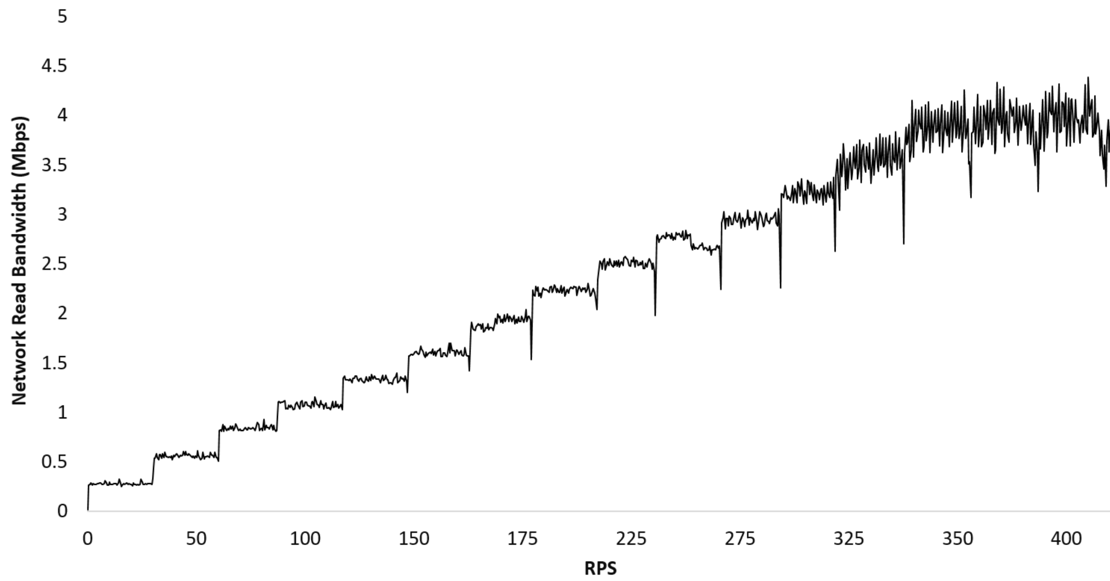

Figure A.38: Network Read Bandwidth Usage for Fast Computation Slow Network data point

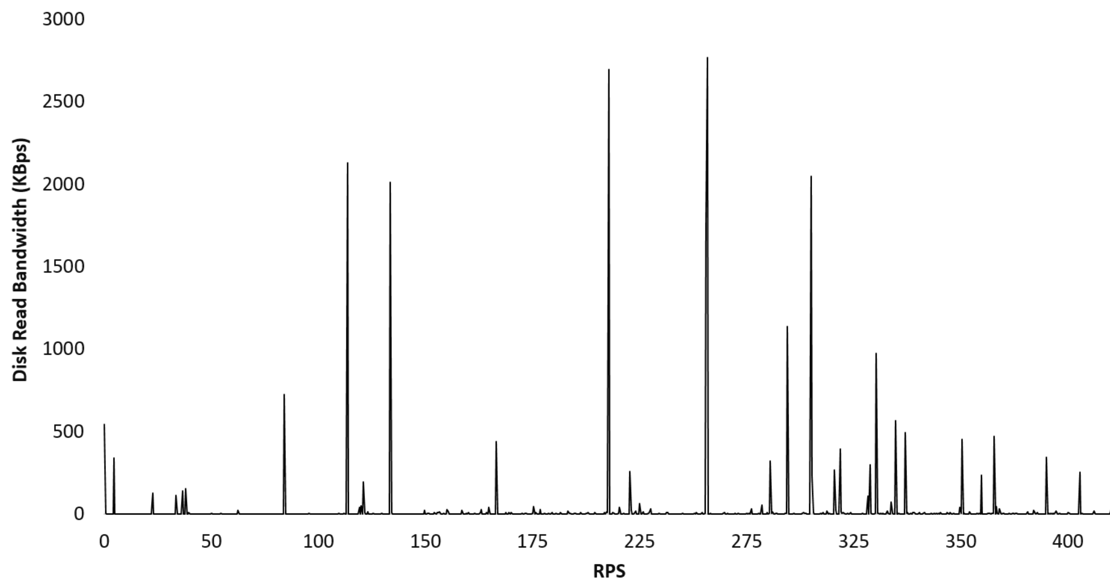

Figure A.39: Disk Read Bandwidth Usage for Fast Computation Slow Network data point

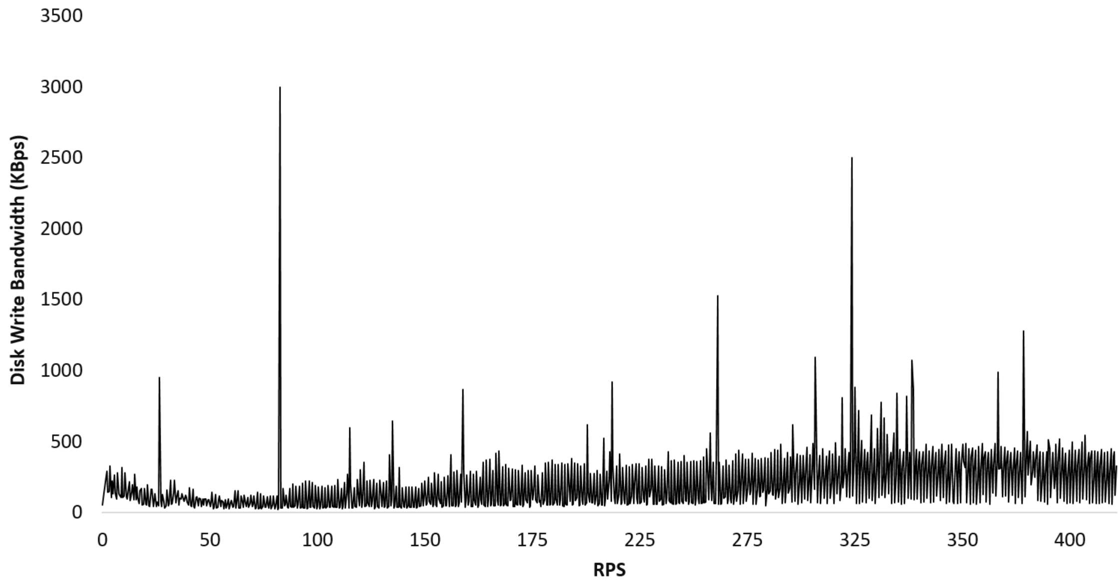

Figure A.40: Disk Write Bandwidth Usage for Fast Computation Slow Network data point

A.2.3 Slow Computation Fast Network

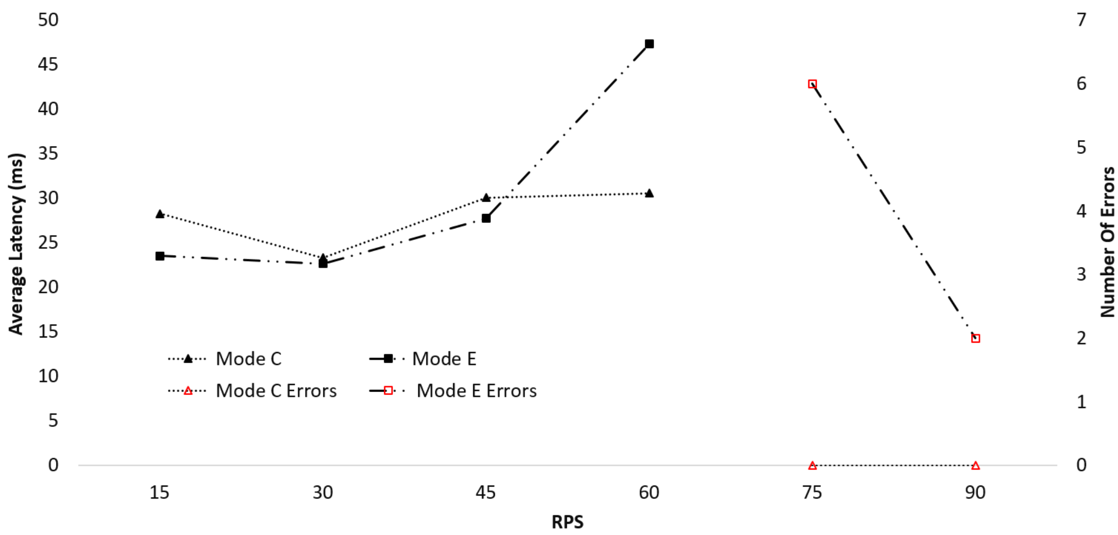

Figure A.41: Average Latency and Errors versus RPS for Slow Computation Fast Network data point

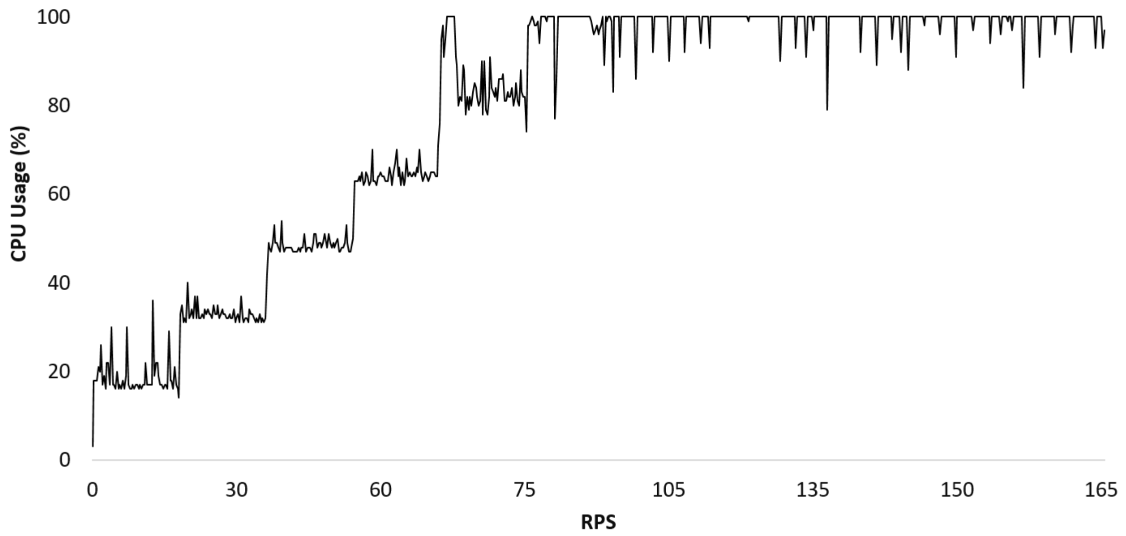

Figure A.42: CPU Usage for Slow Computation Fast Network data point

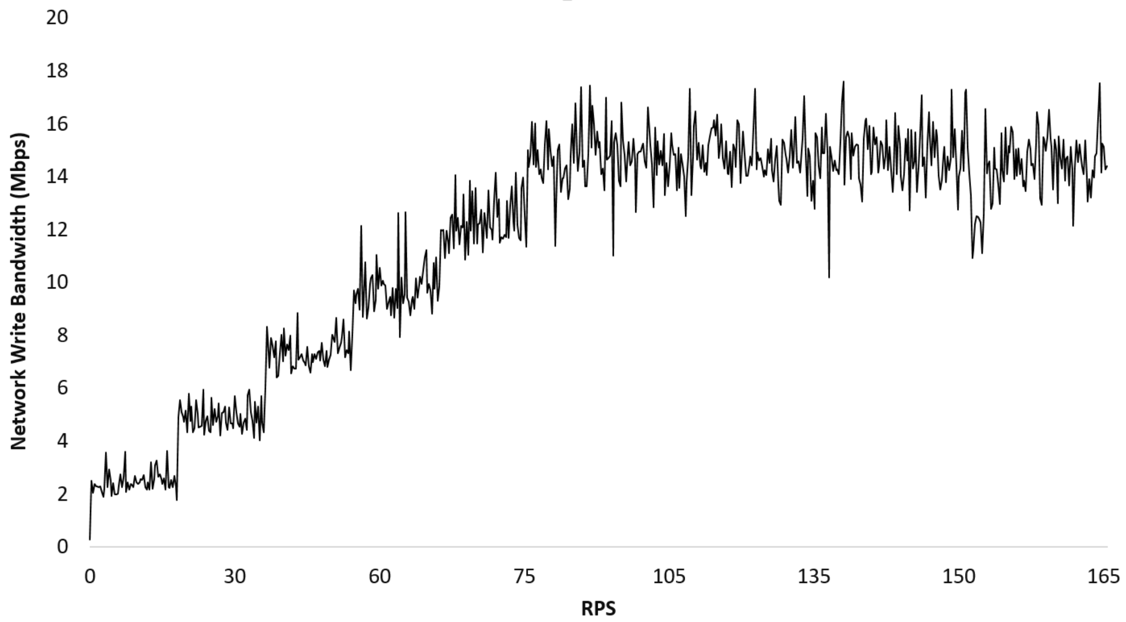

Figure A.43: Network Write Bandwidth Usage for Slow Computation Fast Network data point

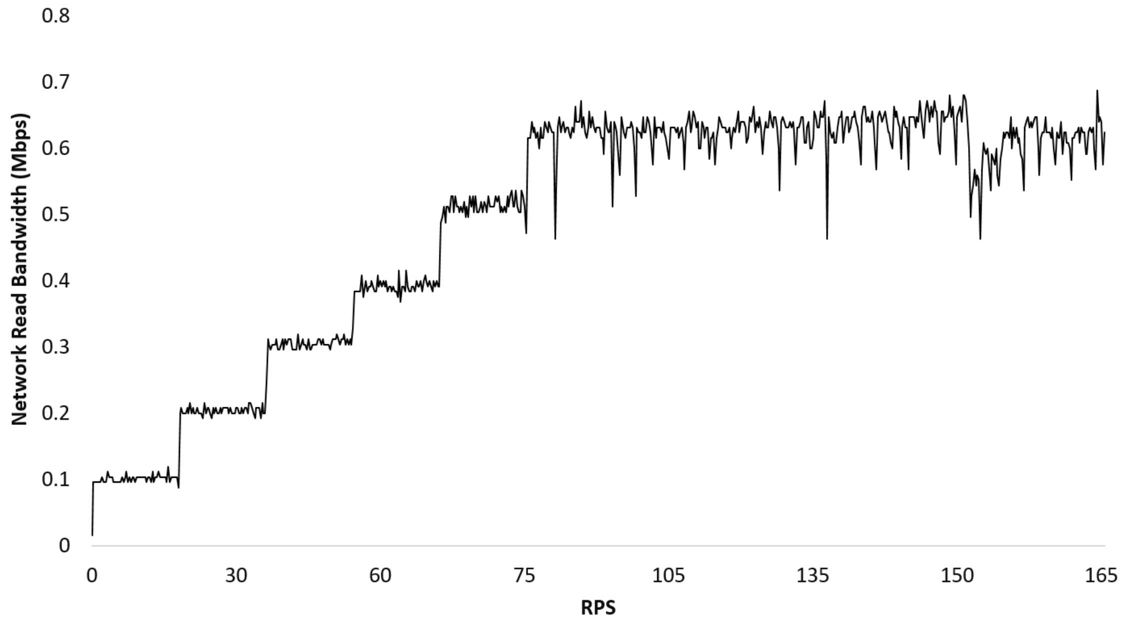

Figure A.44: Network Read Bandwidth Usage for Slow Computation Fast Network data point

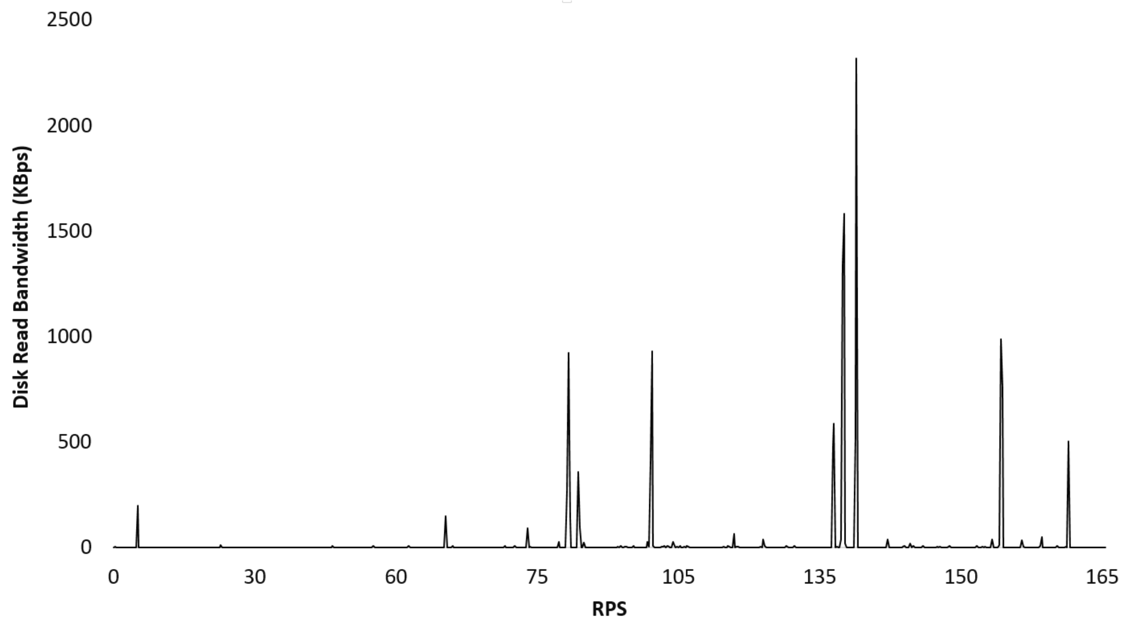

Figure A.45: Disk Read Bandwidth Usage for Slow Computation Fast Network data point

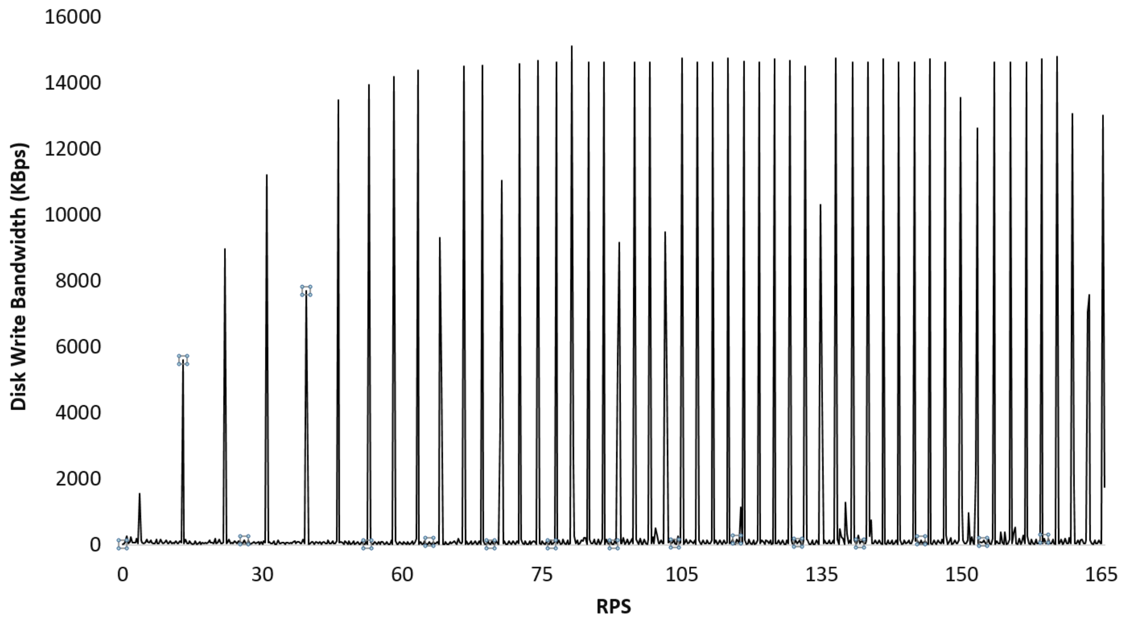

Figure A.46: Disk Write Bandwidth Usage for Slow Computation Fast Network data point

A.2.4 Slow Computation Slow Network

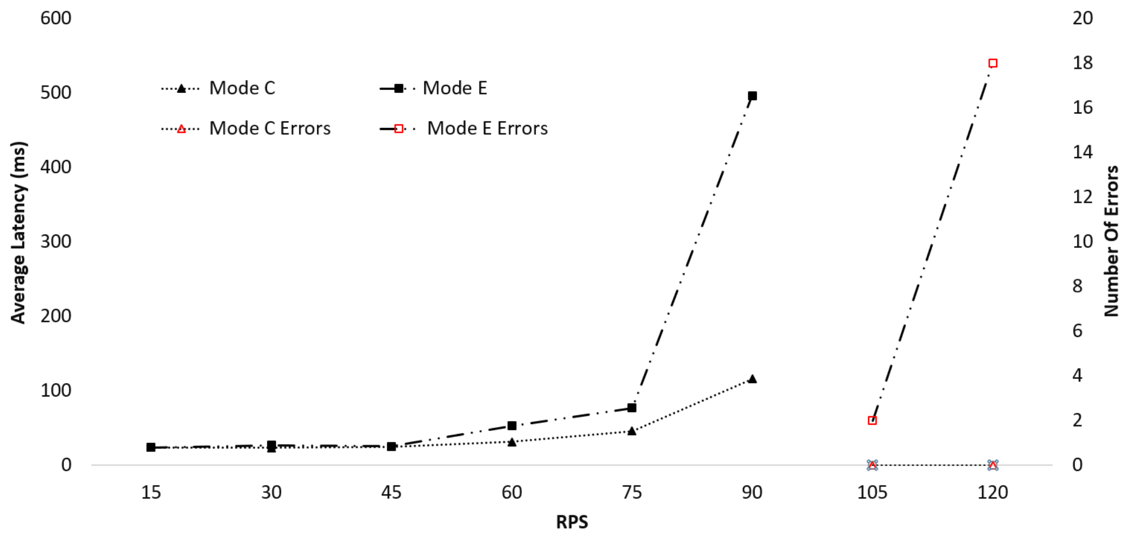

Figure A.47: Average Latency and Errors versus RPS for Slow Computation Slow Network data point

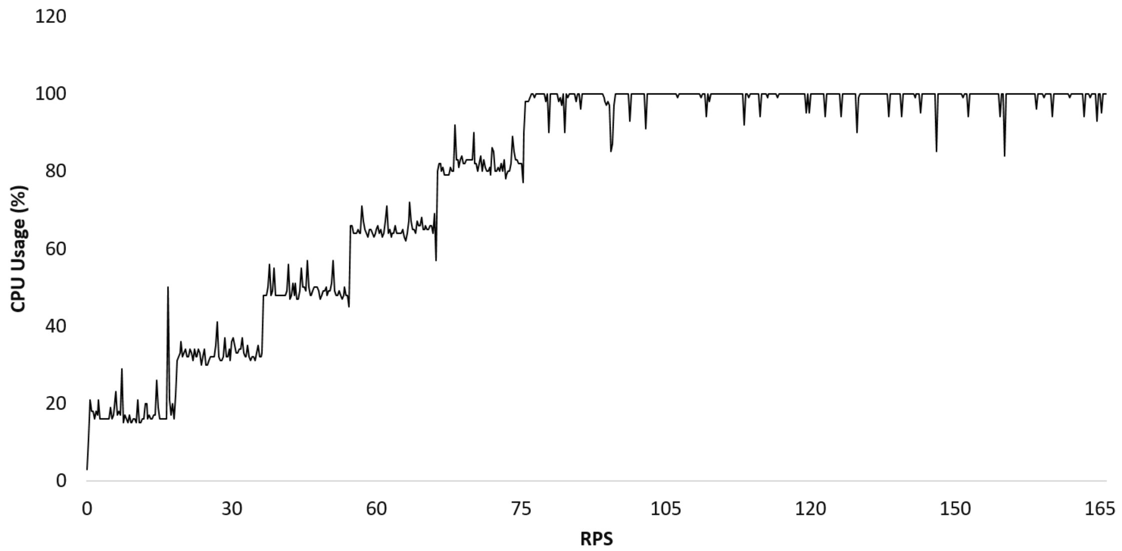

Figure A.48: CPU Usage for Slow Computation Slow Network data point

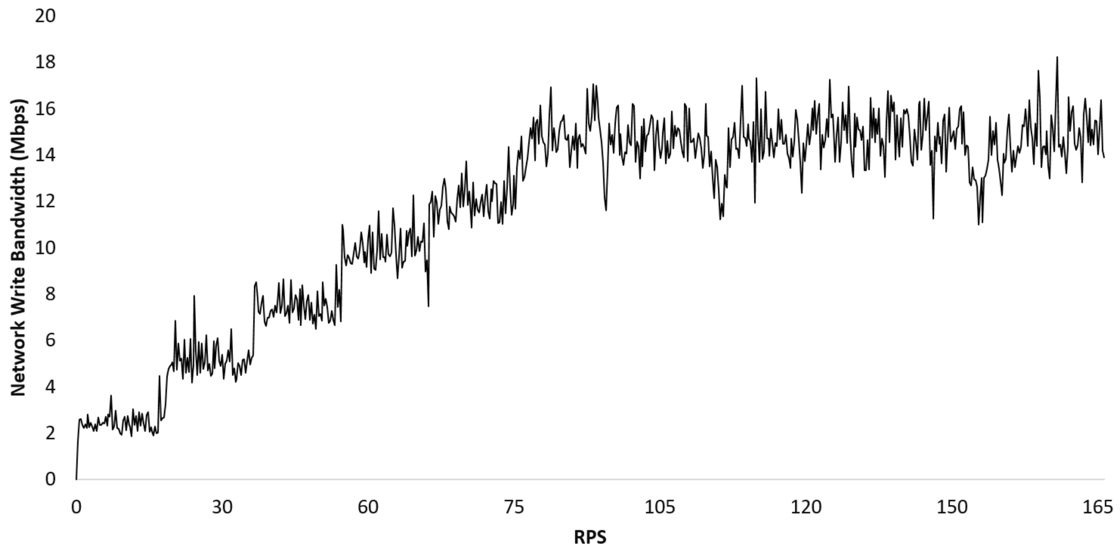

Figure A.49: Network Write Bandwidth Usage for Slow Computation Slow Network data point

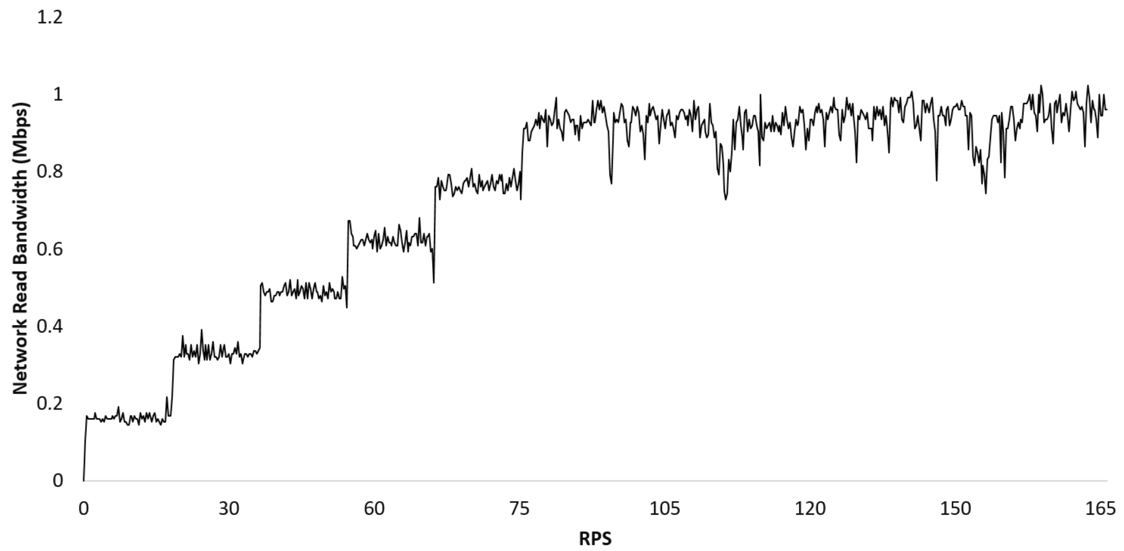

Figure A.50: Network Read Bandwidth Usage for Slow Computation Slow Network data point

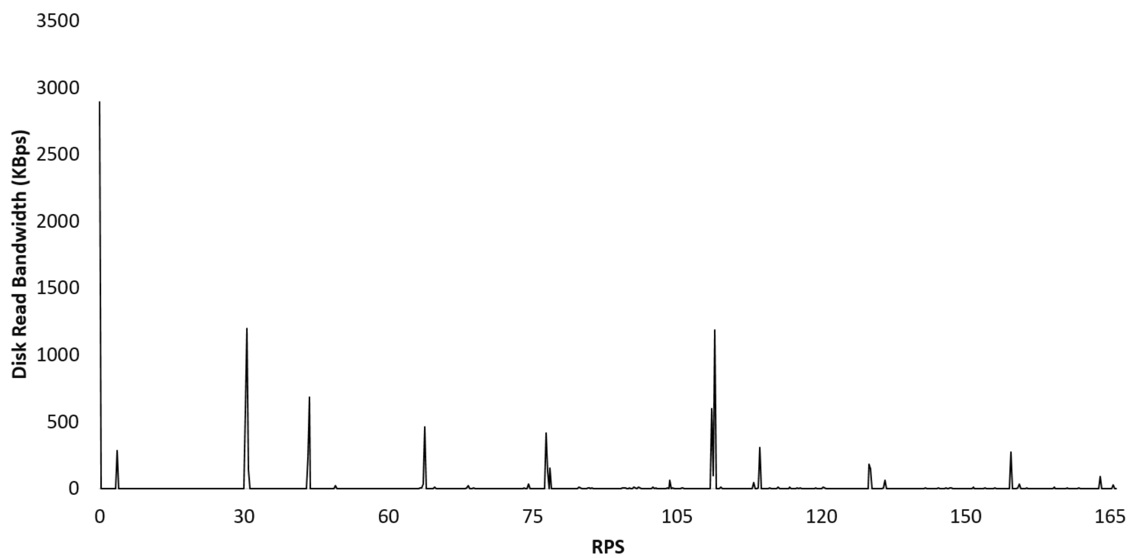

Figure A.51: Disk Read Bandwidth Usage for Slow Computation Slow Network data point

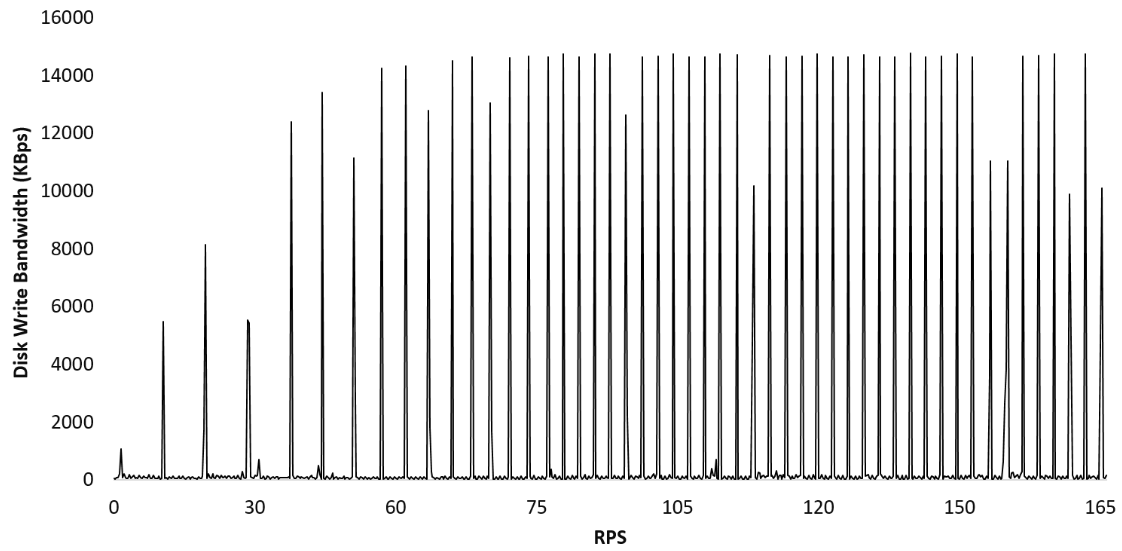

Figure A.52: Disk Write Bandwidth Usage for Slow Computation Slow Network data point

Appendix B

Yahoo Cloud Service Benchmark

Appendix B shows the code that we have developed as part of our open source contribution of the *REST module* to the YCSB.

B.1 REST Workload

Appendix B.1 shows the *RestWorkload.java* in the core module that is responsible for generating the YCSB workload.

```
1  package com.yahoo.ycsb.workloads;
2
3  import com.yahoo.ycsb.ByteIterator;
4  import com.yahoo.ycsb.DB;
5  import com.yahoo.ycsb.RandomByteIterator;
6  import com.yahoo.ycsb.WorkloadException;
7  import com.yahoo.ycsb.generator.*;
8
9  import java.io.BufferedReader;
10 import java.io.FileReader;
11 import java.io.IOException;
12 import java.util.HashMap;
13 import java.util.Map;
14 import java.util.Properties;
15
16 /**
17  * Typical RESTFul services benchmarking scenario. Represents a set of client
18  * calling REST operations like HTTP DELETE, GET, POST, PUT on a web service.
19  * This scenario is completely different from CoreWorkload which is mainly
20  * designed for databases benchmarking. However due to some reusable
21  * functionality this class extends {@link CoreWorkload} and overrides
      necessary
```

```
22  * methods like init, doTransaction etc.
23  */
24 public class RestWorkload extends CoreWorkload {
25
26   /**
27    * The name of the property for the proportion of transactions that are
28    * delete.
29    */
30   public static final String DELETE_PROPORTION_PROPERTY = "deleteproportion";
31
32   /**
33    * The default proportion of transactions that are delete.
34    */
35   public static final String DELETE_PROPORTION_PROPERTY_DEFAULT = "0.00";
36
37   /**
38    * The name of the property for the file that holds the field length size
        for insert operations.
39    */
40   public static final String FIELD_LENGTH_DISTRIBUTION_FILE_PROPERTY = "
       fieldlengthdistfile";
41
42   /**
43    * The default file name that holds the field length size for insert
        operations.
44    */
45   public static final String FIELD_LENGTH_DISTRIBUTION_FILE_PROPERTY_DEFAULT =
        "fieldLengthDistFile.txt";
46
47   /**
48    * In web services even though the CRUD operations follow the same request
49    * distribution, they have different traces and distribution parameter
50    * values. Hence configuring the parameters of these operations separately
51    * makes the benchmark more flexible and capable of generating better
52    * realistic workloads.
53    */
54   // Read related properties.
55   private static final String READ_TRACE_FILE = "url.trace.read";
56   private static final String READ_TRACE_FILE_DEFAULT = "readtrace.txt";
57   private static final String READ_ZIPFIAN_CONSTANT = "readzipfconstant";
58   private static final String READ_ZIPFIAN_CONSTANT_DEAFULT = "0.99";
59   private static final String READ_RECORD_COUNT_PROPERTY = "readrecordcount";
60   // Insert related properties.
61   private static final String INSERT_TRACE_FILE = "url.trace.insert";
62   private static final String INSERT_TRACE_FILE_DEFAULT = "inserttrace.txt";
63   private static final String INSERT_ZIPFIAN_CONSTANT = "insertzipfconstant";
64   private static final String INSERT_ZIPFIAN_CONSTANT_DEAFULT = "0.99";
65   private static final String INSERT_SIZE_ZIPFIAN_CONSTANT = "
       insertsizezipfconstant";
66   private static final String INSERT_SIZE_ZIPFIAN_CONSTANT_DEAFULT = "0.99";
```

```
67     private static final String INSERT_RECORD_COUNT_PROPERTY = "
           insertrecordcount";
68     // Delete related properties.
69     private static final String DELETE_TRACE_FILE = "url.trace.delete";
70     private static final String DELETE_TRACE_FILE_DEFAULT = "deletetrace.txt";
71     private static final String DELETE_ZIPFIAN_CONSTANT = "deletezipfconstant";
72     private static final String DELETE_ZIPFIAN_CONSTANT_DEAFULT = "0.99";
73     private static final String DELETE_RECORD_COUNT_PROPERTY = "
           deleterecordcount";
74     // Delete related properties.
75     private static final String UPDATE_TRACE_FILE = "url.trace.update";
76     private static final String UPDATE_TRACE_FILE_DEFAULT = "updatetrace.txt";
77     private static final String UPDATE_ZIPFIAN_CONSTANT = "updatezipfconstant";
78     private static final String UPDATE_ZIPFIAN_CONSTANT_DEAFULT = "0.99";
79     private static final String UPDATE_RECORD_COUNT_PROPERTY = "
           updaterecordcount";
80 
81     private Map<Integer, String> readUrlMap;
82     private Map<Integer, String> insertUrlMap;
83     private Map<Integer, String> deleteUrlMap;
84     private Map<Integer, String> updateUrlMap;
85     private int readRecordCount;
86     private int insertRecordCount;
87     private int deleteRecordCount;
88     private int updateRecordCount;
89     private NumberGenerator readKeyChooser;
90     private NumberGenerator insertKeyChooser;
91     private NumberGenerator deleteKeyChooser;
92     private NumberGenerator updateKeyChooser;
93     private NumberGenerator fieldlengthgenerator;
94     private DiscreteGenerator operationchooser;
95 
96     @Override
97     public void init(Properties p) throws WorkloadException {
98 
99       readRecordCount = Integer.parseInt(p.getProperty(
             READ_RECORD_COUNT_PROPERTY, String.valueOf(Integer.MAX_VALUE)));
100      insertRecordCount = Integer
101        .parseInt(p.getProperty(INSERT_RECORD_COUNT_PROPERTY, String.valueOf(
               Integer.MAX_VALUE)));
102      deleteRecordCount = Integer
103        .parseInt(p.getProperty(DELETE_RECORD_COUNT_PROPERTY, String.valueOf(
               Integer.MAX_VALUE)));
104      updateRecordCount = Integer
105        .parseInt(p.getProperty(UPDATE_RECORD_COUNT_PROPERTY, String.valueOf(
               Integer.MAX_VALUE)));
106 
107      readUrlMap = getTrace(p.getProperty(READ_TRACE_FILE,
             READ_TRACE_FILE_DEFAULT), readRecordCount);
108      insertUrlMap = getTrace(p.getProperty(INSERT_TRACE_FILE,
             INSERT_TRACE_FILE_DEFAULT), insertRecordCount);
```

```
109       deleteUrlMap = getTrace(p.getProperty(DELETE_TRACE_FILE,
              DELETE_TRACE_FILE_DEFAULT), deleteRecordCount);
110       updateUrlMap = getTrace(p.getProperty(UPDATE_TRACE_FILE,
              UPDATE_TRACE_FILE_DEFAULT), updateRecordCount);
111
112       operationchooser = createOperationGenerator(p);
113
114       // Common distribution for all operations.
115       String requestDistrib = p.getProperty(REQUEST_DISTRIBUTION_PROPERTY,
              REQUEST_DISTRIBUTION_PROPERTY_DEFAULT);
116
117       double readZipfconstant = Double.parseDouble(p.getProperty(
              READ_ZIPFIAN_CONSTANT, READ_ZIPFIAN_CONSTANT_DEAFULT));
118       readKeyChooser = getKeyChooser(requestDistrib, readUrlMap.size(),
              readZipfconstant, p);
119       double updateZipfconstant = Double
120           .parseDouble(p.getProperty(UPDATE_ZIPFIAN_CONSTANT,
                  UPDATE_ZIPFIAN_CONSTANT_DEAFULT));
121       updateKeyChooser = getKeyChooser(requestDistrib, updateUrlMap.size(),
              updateZipfconstant, p);
122       double insertZipfconstant = Double
123           .parseDouble(p.getProperty(INSERT_ZIPFIAN_CONSTANT,
                  INSERT_ZIPFIAN_CONSTANT_DEAFULT));
124       insertKeyChooser = getKeyChooser(requestDistrib, insertUrlMap.size(),
              insertZipfconstant, p);
125       double deleteZipfconstant = Double
126           .parseDouble(p.getProperty(DELETE_ZIPFIAN_CONSTANT,
                  DELETE_ZIPFIAN_CONSTANT_DEAFULT));
127       deleteKeyChooser = getKeyChooser(requestDistrib, deleteUrlMap.size(),
              deleteZipfconstant, p);
128
129       fieldlengthgenerator = getFieldLengthGenerator(p);
130     }
131
132     public static DiscreteGenerator createOperationGenerator(final Properties p)
            {
133       // Re-using CoreWorkload method.
134       final DiscreteGenerator operationChooser = CoreWorkload.
              createOperationGenerator(p);
135       // Needs special handling for delete operations not supported in
              CoreWorkload.
136       double deleteproportion = Double
137           .parseDouble(p.getProperty(DELETE_PROPORTION_PROPERTY,
                  DELETE_PROPORTION_PROPERTY_DEFAULT));
138       if (deleteproportion > 0) {
139         operationChooser.addValue(deleteproportion, "DELETE");
140       }
141       return operationChooser;
142     }
143
```

```
144   private static NumberGenerator getKeyChooser(String requestDistrib, int
          recordCount, double zipfContant,
145                                                   Properties p) throws
                                                         WorkloadException {
146     NumberGenerator keychooser;
147
148     switch (requestDistrib) {
149     case "exponential":
150       double percentile = Double.parseDouble(p.getProperty(
              ExponentialGenerator.EXPONENTIAL_PERCENTILE_PROPERTY,
151           ExponentialGenerator.EXPONENTIAL_PERCENTILE_DEFAULT));
152       double frac = Double.parseDouble(p.getProperty(ExponentialGenerator.
              EXPONENTIAL_FRAC_PROPERTY,
153           ExponentialGenerator.EXPONENTIAL_FRAC_DEFAULT));
154       keychooser = new ExponentialGenerator(percentile, recordCount * frac);
155       break;
156     case "uniform":
157       keychooser = new UniformIntegerGenerator(0, recordCount - 1);
158       break;
159     case "zipfian":
160       keychooser = new ZipfianGenerator(recordCount, zipfContant);
161       break;
162     case "latest":
163       throw new WorkloadException("Latest request distribution is not
              supported for RestWorkload.");
164     case "hotspot":
165       double hotsetfraction = Double.parseDouble(p.getProperty(
              HOTSPOT_DATA_FRACTION, HOTSPOT_DATA_FRACTION_DEFAULT));
166       double hotopnfraction = Double.parseDouble(p.getProperty(
              HOTSPOT_OPN_FRACTION, HOTSPOT_OPN_FRACTION_DEFAULT));
167       keychooser = new HotspotIntegerGenerator(0, recordCount - 1,
              hotsetfraction, hotopnfraction);
168       break;
169     default:
170       throw new WorkloadException("Unknown request distribution \"" +
              requestDistrib + "\"");
171     }
172     return keychooser;
173   }
174
175   protected static NumberGenerator getFieldLengthGenerator(Properties p)
          throws WorkloadException {
176     // Re-using CoreWorkload method.
177     NumberGenerator fieldLengthGenerator = CoreWorkload.
            getFieldLengthGenerator(p);
178     String fieldlengthdistribution = p.getProperty(
            FIELD_LENGTH_DISTRIBUTION_PROPERTY,
179         FIELD_LENGTH_DISTRIBUTION_PROPERTY_DEFAULT);
180     // Needs special handling for Zipfian distribution for variable Zipf
            Constant.
181     if (fieldlengthdistribution.compareTo("zipfian") == 0) {
```

```
182          int fieldlength = Integer.parseInt(p.getProperty(FIELD_LENGTH_PROPERTY,
                FIELD_LENGTH_PROPERTY_DEFAULT));
183          double insertsizezipfconstant = Double
184              .parseDouble(p.getProperty(INSERT_SIZE_ZIPFIAN_CONSTANT,
                    INSERT_SIZE_ZIPFIAN_CONSTANT_DEAFULT));
185          fieldLengthGenerator = new ZipfianGenerator(1, fieldlength,
                insertsizezipfconstant);
186        }
187        return fieldLengthGenerator;
188      }
189
190      /**
191       * Reads the trace file and returns a URL map.
192       */
193      private static Map<Integer, String> getTrace(String filePath, int
            recordCount)
194        throws WorkloadException {
195        Map<Integer, String> urlMap = new HashMap<Integer, String>();
196        int count = 0;
197        String line;
198        try {
199          FileReader inputFile = new FileReader(filePath);
200          BufferedReader bufferReader = new BufferedReader(inputFile);
201          while ((line = bufferReader.readLine()) != null) {
202            urlMap.put(count++, line.trim());
203            if (count >= recordCount) {
204              break;
205            }
206          }
207          bufferReader.close();
208        } catch (IOException e) {
209          throw new WorkloadException(
210            "Error while reading the trace. Please make sure the trace file path
                is correct. "
211              + e.getLocalizedMessage());
212        }
213        return urlMap;
214      }
215
216      /**
217       * Not required for Rest Clients as data population is service specific.
218       */
219      @Override
220      public boolean doInsert(DB db, Object threadstate) {
221        return false;
222      }
223
224      @Override
225      public boolean doTransaction(DB db, Object threadstate) {
226        String operation = operationchooser.nextString();
227        if (operation == null) {
```

```
228         return false;
229       }
230 
231       switch (operation) {
232       case "UPDATE":
233         doTransactionUpdate(db);
234         break;
235       case "INSERT":
236         doTransactionInsert(db);
237         break;
238       case "DELETE":
239         doTransactionDelete(db);
240         break;
241       default:
242         doTransactionRead(db);
243       }
244       return true;
245     }
246 
247     /**
248      * Returns next URL to be called.
249      */
250     private String getNextURL(int opType) {
251       if (opType == 1) {
252         return readUrlMap.get(readKeyChooser.nextValue().intValue());
253       } else if (opType == 2) {
254         return insertUrlMap.get(insertKeyChooser.nextValue().intValue());
255       } else if (opType == 3) {
256         return deleteUrlMap.get(deleteKeyChooser.nextValue().intValue());
257       } else {
258         return updateUrlMap.get(updateKeyChooser.nextValue().intValue());
259       }
260     }
261 
262     @Override
263     public void doTransactionRead(DB db) {
264       HashMap<String, ByteIterator> result = new HashMap<String, ByteIterator>()
            ;
265       db.read(null, getNextURL(1), null, result);
266     }
267 
268     @Override
269     public void doTransactionInsert(DB db) {
270       HashMap<String, ByteIterator> value = new HashMap<String, ByteIterator>();
271       // Create random bytes of insert data with a specific size.
272       value.put("data", new RandomByteIterator(fieldlengthgenerator.nextValue().
            longValue()));
273       db.insert(null, getNextURL(2), value);
274     }
275 
276     public void doTransactionDelete(DB db) {
```

```
277       db.delete(null, getNextURL(3));
278     }
279 
280     @Override
281     public void doTransactionUpdate(DB db) {
282       HashMap<String, ByteIterator> value = new HashMap<String, ByteIterator>();
283       // Create random bytes of update data with a specific size.
284       value.put("data", new RandomByteIterator(fieldlengthgenerator.nextValue().
            longValue()));
285       db.update(null, getNextURL(4), value);
286     }
287 
288 }
```

B.2 REST Client

Appendix B.2 shows the *RestClient.java* in the core module that is responsible for generating HTTP REST requests.

```
1  package com.yahoo.ycsb.webservice.rest;
2 
3  import java.io.BufferedReader;
4  import java.io.ByteArrayInputStream;
5  import java.io.IOException;
6  import java.io.InputStream;
7  import java.io.InputStreamReader;
8  import java.util.HashMap;
9  import java.util.Properties;
10 import java.util.Set;
11 import java.util.Vector;
12 import java.util.zip.GZIPInputStream;
13 
14 import javax.ws.rs.HttpMethod;
15 
16 import org.apache.http.HttpEntity;
17 import org.apache.http.client.ClientProtocolException;
18 import org.apache.http.client.config.RequestConfig;
19 import org.apache.http.client.methods.CloseableHttpResponse;
20 import org.apache.http.client.methods.HttpDelete;
21 import org.apache.http.client.methods.HttpEntityEnclosingRequestBase;
22 import org.apache.http.client.methods.HttpGet;
23 import org.apache.http.client.methods.HttpPost;
24 import org.apache.http.client.methods.HttpPut;
25 import org.apache.http.entity.ContentType;
26 import org.apache.http.entity.InputStreamEntity;
27 import org.apache.http.impl.client.CloseableHttpClient;
28 import org.apache.http.impl.client.HttpClientBuilder;
```

```
29 import org.apache.http.util.EntityUtils;
30
31 import com.yahoo.ycsb.ByteIterator;
32 import com.yahoo.ycsb.DB;
33 import com.yahoo.ycsb.DBException;
34 import com.yahoo.ycsb.Status;
35 import com.yahoo.ycsb.StringByteIterator;
36
37 /**
38  * Class responsible for making web service requests for benchmarking purpose.
39  * Using Apache HttpClient over standard Java HTTP API as this is more
       flexible
40  * and provides better functionality. For example HttpClient can automatically
41  * handle redirects and proxy authentication which the standard Java API can't
       .
42  */
43 public class RestClient extends DB {
44
45   private static final String URL_PREFIX = "url.prefix";
46   private static final String CON_TIMEOUT = "timeout.con";
47   private static final String READ_TIMEOUT = "timeout.read";
48   private static final String EXEC_TIMEOUT = "timeout.exec";
49   private static final String LOG_ENABLED = "log.enable";
50   private static final String HEADERS = "headers";
51   private static final String COMPRESSED_RESPONSE = "response.compression";
52   private boolean compressedResponse;
53   private boolean logEnabled;
54   private String urlPrefix;
55   private Properties props;
56   private String[] headers;
57   private CloseableHttpClient client;
58   private int conTimeout = 10000;
59   private int readTimeout = 10000;
60   private int execTimeout = 10000;
61   private volatile Criteria requestTimedout = new Criteria(false);
62
63   @Override
64   public void init() throws DBException {
65     props = getProperties();
66     urlPrefix = props.getProperty(URL_PREFIX, "http://127.0.0.1:8080");
67     conTimeout = Integer.valueOf(props.getProperty(CON_TIMEOUT, "10")) * 1000;
68     readTimeout = Integer.valueOf(props.getProperty(READ_TIMEOUT, "10")) *
         1000;
69     execTimeout = Integer.valueOf(props.getProperty(EXEC_TIMEOUT, "10")) *
         1000;
70     logEnabled = Boolean.valueOf(props.getProperty(LOG_ENABLED, "false").trim
         ());
71     compressedResponse = Boolean.valueOf(props.getProperty(COMPRESSED_RESPONSE
         , "false").trim());
72     headers = props.getProperty(HEADERS, "Accept */* Content-Type application/
         xml user-agent Mozilla/5.0 ").trim()
```

```
73              .split(" ");
74        setupClient();
75      }
76
77      private void setupClient() {
78        RequestConfig.Builder requestBuilder = RequestConfig.custom();
79        requestBuilder = requestBuilder.setConnectTimeout(conTimeout);
80        requestBuilder = requestBuilder.setConnectionRequestTimeout(readTimeout);
81        requestBuilder = requestBuilder.setSocketTimeout(readTimeout);
82        HttpClientBuilder clientBuilder = HttpClientBuilder.create().
            setDefaultRequestConfig(requestBuilder.build());
83        this.client = clientBuilder.setConnectionManagerShared(true).build();
84      }
85
86      @Override
87      public Status read(String table, String endpoint, Set<String> fields,
          HashMap<String, ByteIterator> result) {
88        int responseCode;
89        try {
90          responseCode = httpGet(urlPrefix + endpoint, result);
91        } catch (Exception e) {
92          responseCode = handleExceptions(e, urlPrefix + endpoint, HttpMethod.GET)
              ;
93        }
94        if (logEnabled) {
95          System.err.println(new StringBuilder("GET Request: ").append(urlPrefix).
              append(endpoint)
96              .append(" | Response Code: ").append(responseCode).toString());
97        }
98        return getStatus(responseCode);
99      }
100
101     @Override
102     public Status insert(String table, String endpoint, HashMap<String,
          ByteIterator> values) {
103       int responseCode;
104       try {
105         responseCode = httpExecute(new HttpPost(urlPrefix + endpoint), values.
              get("data").toString());
106       } catch (Exception e) {
107         responseCode = handleExceptions(e, urlPrefix + endpoint, HttpMethod.POST
              );
108       }
109       if (logEnabled) {
110         System.err.println(new StringBuilder("POST Request: ").append(urlPrefix)
              .append(endpoint)
111             .append(" | Response Code: ").append(responseCode).toString());
112       }
113       return getStatus(responseCode);
114     }
115
```

```
116   @Override
117   public Status delete(String table, String endpoint) {
118     int responseCode;
119     try {
120       responseCode = httpDelete(urlPrefix + endpoint);
121     } catch (Exception e) {
122       responseCode = handleExceptions(e, urlPrefix + endpoint, HttpMethod.
          DELETE);
123     }
124     if (logEnabled) {
125       System.err.println(new StringBuilder("DELETE Request: ").append(
          urlPrefix).append(endpoint)
126           .append(" | Response Code: ").append(responseCode).toString());
127     }
128     return getStatus(responseCode);
129   }
130
131   @Override
132   public Status update(String table, String endpoint, HashMap<String,
        ByteIterator> values) {
133     int responseCode;
134     try {
135       responseCode = httpExecute(new HttpPut(urlPrefix + endpoint), values.get
          ("data").toString());
136     } catch (Exception e) {
137       responseCode = handleExceptions(e, urlPrefix + endpoint, HttpMethod.PUT)
          ;
138     }
139     if (logEnabled) {
140       System.err.println(new StringBuilder("PUT Request: ").append(urlPrefix).
          append(endpoint)
141           .append(" | Response Code: ").append(responseCode).toString());
142     }
143     return getStatus(responseCode);
144   }
145
146   @Override
147   public Status scan(String table, String startkey, int recordcount, Set<
        String> fields,
148       Vector<HashMap<String, ByteIterator>> result) {
149     return Status.NOT_IMPLEMENTED;
150   }
151
152   // Maps HTTP status codes to YCSB status codes.
153   private Status getStatus(int responseCode) {
154     int rc = responseCode / 100;
155     if (responseCode == 400) {
156       return Status.BAD_REQUEST;
157     } else if (responseCode == 403) {
158       return Status.FORBIDDEN;
159     } else if (responseCode == 404) {
```

```
160         return Status.NOT_FOUND;
161       } else if (responseCode == 501) {
162         return Status.NOT_IMPLEMENTED;
163       } else if (responseCode == 503) {
164         return Status.SERVICE_UNAVAILABLE;
165       } else if (rc == 5) {
166         return Status.ERROR;
167       }
168       return Status.OK;
169     }
170 
171     private int handleExceptions(Exception e, String url, String method) {
172       if (logEnabled) {
173         System.err.println(new StringBuilder(method).append(" Request: ").append
                (url).append(" | ")
174             .append(e.getClass().getName()).append(" occured | Error message: ")
175             .append(e.getMessage()).toString());
176       }
177 
178       if (e instanceof ClientProtocolException) {
179         return 400;
180       }
181       return 500;
182     }
183 
184     // Connection is automatically released back in case of an exception.
185     private int httpGet(String endpoint, HashMap<String, ByteIterator> result)
          throws IOException {
186       requestTimedout.setIsSatisfied(false);
187       Thread timer = new Thread(new Timer(execTimeout, requestTimedout));
188       timer.start();
189       int responseCode = 200;
190       HttpGet request = new HttpGet(endpoint);
191       for (int i = 0; i < headers.length; i = i + 2) {
192         request.setHeader(headers[i], headers[i + 1]);
193       }
194       CloseableHttpResponse response = client.execute(request);
195       responseCode = response.getStatusLine().getStatusCode();
196       HttpEntity responseEntity = response.getEntity();
197       // If null entity don't bother about connection release.
198       if (responseEntity != null) {
199         InputStream stream = responseEntity.getContent();
200         /*
201          * TODO: Gzip Compression must be supported in the future. Header[]
202          * header = response.getAllHeaders();
203          * if(response.getHeaders("Content-Encoding")[0].getValue().contains
204          * ("gzip")) stream = new GZIPInputStream(stream);
205          */
206         BufferedReader reader = new BufferedReader(new InputStreamReader(stream,
                "UTF-8"));
207         StringBuffer responseContent = new StringBuffer();
```

```
208         String line = "";
209         while ((line = reader.readLine()) != null) {
210           if (requestTimedout.isSatisfied()) {
211             // Must avoid memory leak.
212             reader.close();
213             stream.close();
214             EntityUtils.consumeQuietly(responseEntity);
215             response.close();
216             client.close();
217             throw new TimeoutException();
218           }
219           responseContent.append(line);
220         }
221         timer.interrupt();
222         result.put("response", new StringByteIterator(responseContent.toString()
              ));
223         // Closing the input stream will trigger connection release.
224         stream.close();
225       }
226       EntityUtils.consumeQuietly(responseEntity);
227       response.close();
228       client.close();
229       return responseCode;
230     }
231
232     private int httpExecute(HttpEntityEnclosingRequestBase request, String data)
            throws IOException {
233       requestTimedout.setIsSatisfied(false);
234       Thread timer = new Thread(new Timer(execTimeout, requestTimedout));
235       timer.start();
236       int responseCode = 200;
237       for (int i = 0; i < headers.length; i = i + 2) {
238         request.setHeader(headers[i], headers[i + 1]);
239       }
240       InputStreamEntity reqEntity = new InputStreamEntity(new
              ByteArrayInputStream(data.getBytes()),
241             ContentType.APPLICATION_FORM_URLENCODED);
242       reqEntity.setChunked(true);
243       request.setEntity(reqEntity);
244       CloseableHttpResponse response = client.execute(request);
245       responseCode = response.getStatusLine().getStatusCode();
246       HttpEntity responseEntity = response.getEntity();
247       // If null entity don't bother about connection release.
248       if (responseEntity != null) {
249         InputStream stream = responseEntity.getContent();
250         if (compressedResponse) {
251           stream = new GZIPInputStream(stream);
252         }
253         BufferedReader reader = new BufferedReader(new InputStreamReader(stream,
              "UTF-8"));
254         StringBuffer responseContent = new StringBuffer();
```

```
255         String line = "";
256         while ((line = reader.readLine()) != null) {
257           if (requestTimedout.isSatisfied()) {
258             // Must avoid memory leak.
259             reader.close();
260             stream.close();
261             EntityUtils.consumeQuietly(responseEntity);
262             response.close();
263             client.close();
264             throw new TimeoutException();
265           }
266           responseContent.append(line);
267         }
268         timer.interrupt();
269         // Closing the input stream will trigger connection release.
270         stream.close();
271       }
272       EntityUtils.consumeQuietly(responseEntity);
273       response.close();
274       client.close();
275       return responseCode;
276     }
277 
278     private int httpDelete(String endpoint) throws IOException {
279       requestTimedout.setIsSatisfied(false);
280       Thread timer = new Thread(new Timer(execTimeout, requestTimedout));
281       timer.start();
282       int responseCode = 200;
283       HttpDelete request = new HttpDelete(endpoint);
284       for (int i = 0; i < headers.length; i = i + 2) {
285         request.setHeader(headers[i], headers[i + 1]);
286       }
287       CloseableHttpResponse response = client.execute(request);
288       responseCode = response.getStatusLine().getStatusCode();
289       response.close();
290       client.close();
291       return responseCode;
292     }
293 
294     /**
295      * Marks the input {@link Criteria} as satisfied when the input time has
           elapsed.
296      */
297     class Timer implements Runnable {
298 
299       private long timeout;
300       private Criteria timedout;
301 
302       public Timer(long timeout, Criteria timedout) {
303         this.timedout = timedout;
304         this.timeout = timeout;
```

```
305       }
306
307       @Override
308       public void run() {
309         try {
310           Thread.sleep(timeout);
311           this.timedout.setIsSatisfied(true);
312         } catch (InterruptedException e) {
313           // Do nothing.
314         }
315       }
316
317     }
318
319     /**
320      * Sets the flag when a criteria is fulfilled.
321      */
322     class Criteria {
323
324       private boolean isSatisfied;
325
326       public Criteria(boolean isSatisfied) {
327         this.isSatisfied = isSatisfied;
328       }
329
330       public boolean isSatisfied() {
331         return isSatisfied;
332       }
333
334       public void setIsSatisfied(boolean satisfied) {
335         this.isSatisfied = satisfied;
336       }
337
338     }
339
340     /**
341      * Private exception class for execution timeout.
342      */
343     class TimeoutException extends RuntimeException {
344
345       private static final long serialVersionUID = 1L;
346
347       public TimeoutException() {
348         super("HTTP Request exceeded execution time limit.");
349       }
350
351     }
352
353   }
```

B.3 Integration Test

Appendix B.3 shows the *IntegrationTest.java* in the REST module that is responsible for verifying the functionality of the REST module.

```
1  package com.yahoo.ycsb.webservice.rest;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.io.File;
6  import java.io.FileNotFoundException;
7  import java.io.IOException;
8  import java.util.List;
9
10 import javax.servlet.ServletException;
11
12 import org.apache.catalina.Context;
13 import org.apache.catalina.LifecycleException;
14 import org.apache.catalina.startup.Tomcat;
15 import org.glassfish.jersey.server.ResourceConfig;
16 import org.glassfish.jersey.servlet.ServletContainer;
17 import org.junit.AfterClass;
18 import org.junit.BeforeClass;
19 import org.junit.FixMethodOrder;
20 import org.junit.Rule;
21 import org.junit.Test;
22 import org.junit.contrib.java.lang.system.Assertion;
23 import org.junit.contrib.java.lang.system.ExpectedSystemExit;
24 import org.junit.runners.MethodSorters;
25
26 import com.yahoo.ycsb.Client;
27 import com.yahoo.ycsb.DBException;
28 import com.yahoo.ycsb.webservice.rest.Utils;
29
30 /**
31  * Integration test cases to verify the end to end working of the rest-binding
32  * module. It performs these steps in order. 1. Runs an embedded Tomcat
33  * server with a mock RESTFul web service. 2. Invokes the {@link Client}
34  * class with the required parameters to start benchmarking the mock REST
35  * service. 3. Compares the response stored in the output file by {@link
      Client}
36  * class with the response expected. 4. Stops the embedded Tomcat server.
37  * Cases for verifying the handling of different HTTP status like 2xx & 5xx
      have
38  * been included in success and failure test cases.
39  */
40 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
41 public class IntegrationTest {
42
```

```
43    @Rule
44    public final ExpectedSystemExit exit = ExpectedSystemExit.none();
45
46    private static int port = 8080;
47    private static Tomcat tomcat;
48    private static final String WORKLOAD_FILEPATH =  IntegrationTest.class.
        getClassLoader().getResource("workload_rest").getPath();
49    private static final String TRACE_FILEPATH = IntegrationTest.class.
        getClassLoader().getResource("trace.txt").getPath();
50    private static final String ERROR_TRACE_FILEPATH = IntegrationTest.class.
        getClassLoader().getResource("error_trace.txt").getPath();
51    private static final String RESULTS_FILEPATH = IntegrationTest.class.
        getClassLoader().getResource(".").getPath() + "results.txt";
52
53    @BeforeClass
54    public static void init() throws ServletException, LifecycleException,
        FileNotFoundException, IOException,
55         DBException, InterruptedException {
56      String webappDirLocation =  IntegrationTest.class.getClassLoader().
          getResource("WebContent").getPath();
57      while (!Utils.available(port)) {
58        port++;
59      }
60      tomcat = new Tomcat();
61      tomcat.setPort(Integer.valueOf(port));
62      Context context = tomcat.addWebapp("/webService", new File(
          webappDirLocation).getAbsolutePath());
63      Tomcat.addServlet(context, "jersey-container-servlet", resourceConfig());
64      context.addServletMapping("/rest/*", "jersey-container-servlet");
65      tomcat.start();
66      // Allow time for proper startup.
67      Thread.sleep(1000);
68    }
69
70    @AfterClass
71    public static void cleanUp() throws LifecycleException {
72      tomcat.stop();
73    }
74
75    // All read operations during benchmark are executed successfully with an
        HTTP OK status.
76    @Test
77    public void testReadOpsBenchmarkSuccess() throws InterruptedException {
78      exit.expectSystemExit();
79      exit.checkAssertionAfterwards(new Assertion() {
80        @Override
81        public void checkAssertion() throws Exception {
82          List<String> results = Utils.read(RESULTS_FILEPATH);
83          assertEquals(true, results.contains("[READ], Return=OK, 1"));
84          Utils.delete(RESULTS_FILEPATH);
85        }
```

```
86       });
87       Client.main(getArgs(TRACE_FILEPATH, 1, 0, 0, 0));
88     }
89
90     //All read operations during benchmark are executed with an HTTP 500 error.
91     @Test
92     public void testReadOpsBenchmarkFailure() throws InterruptedException {
93       exit.expectSystemExit();
94       exit.checkAssertionAfterwards(new Assertion() {
95         @Override
96         public void checkAssertion() throws Exception {
97           List<String> results = Utils.read(RESULTS_FILEPATH);
98           assertEquals(true, results.contains("[READ], Return=ERROR, 1"));
99           Utils.delete(RESULTS_FILEPATH);
100        }
101      });
102      Client.main(getArgs(ERROR_TRACE_FILEPATH, 1, 0, 0, 0));
103    }
104
105    //All insert operations during benchmark are executed successfully with an
         HTTP OK status.
106    @Test
107    public void testInsertOpsBenchmarkSuccess() throws InterruptedException {
108      exit.expectSystemExit();
109      exit.checkAssertionAfterwards(new Assertion() {
110        @Override
111        public void checkAssertion() throws Exception {
112          List<String> results = Utils.read(RESULTS_FILEPATH);
113          assertEquals(true, results.contains("[INSERT], Return=OK, 1"));
114          Utils.delete(RESULTS_FILEPATH);
115        }
116      });
117      Client.main(getArgs(TRACE_FILEPATH, 0, 1, 0, 0));
118    }
119
120    //All read operations during benchmark are executed with an HTTP 500 error.
121    @Test
122    public void testInsertOpsBenchmarkFailure() throws InterruptedException {
123      exit.expectSystemExit();
124      exit.checkAssertionAfterwards(new Assertion() {
125        @Override
126        public void checkAssertion() throws Exception {
127          List<String> results = Utils.read(RESULTS_FILEPATH);
128          assertEquals(true, results.contains("[INSERT], Return=ERROR, 1"));
129          Utils.delete(RESULTS_FILEPATH);
130        }
131      });
132      Client.main(getArgs(ERROR_TRACE_FILEPATH, 0, 1, 0, 0));
133    }
134
```

```
135     //All update operations during benchmark are executed successfully with an
          HTTP OK status.
136     @Test
137     public void testUpdateOpsBenchmarkSuccess() throws InterruptedException {
138       exit.expectSystemExit();
139       exit.checkAssertionAfterwards(new Assertion() {
140         @Override
141         public void checkAssertion() throws Exception {
142           List<String> results = Utils.read(RESULTS_FILEPATH);
143           assertEquals(true, results.contains("[UPDATE], Return=OK, 1"));
144           Utils.delete(RESULTS_FILEPATH);
145         }
146       });
147       Client.main(getArgs(TRACE_FILEPATH, 0, 0, 1, 0));
148     }
149
150     //All read operations during benchmark are executed with an HTTP 500 error.
151     @Test
152     public void testUpdateOpsBenchmarkFailure() throws InterruptedException {
153       exit.expectSystemExit();
154       exit.checkAssertionAfterwards(new Assertion() {
155         @Override
156         public void checkAssertion() throws Exception {
157           List<String> results = Utils.read(RESULTS_FILEPATH);
158           assertEquals(true, results.contains("[UPDATE], Return=ERROR, 1"));
159           Utils.delete(RESULTS_FILEPATH);
160         }
161       });
162       Client.main(getArgs(ERROR_TRACE_FILEPATH, 0, 0, 1, 0));
163     }
164
165     //All delete operations during benchmark are executed successfully with an
          HTTP OK status.
166     @Test
167     public void testDeleteOpsBenchmarkSuccess() throws InterruptedException {
168       exit.expectSystemExit();
169       exit.checkAssertionAfterwards(new Assertion() {
170         @Override
171         public void checkAssertion() throws Exception {
172           List<String> results = Utils.read(RESULTS_FILEPATH);
173           assertEquals(true, results.contains("[DELETE], Return=OK, 1"));
174           Utils.delete(RESULTS_FILEPATH);
175         }
176       });
177       Client.main(getArgs(TRACE_FILEPATH, 0, 0, 0, 1));
178     }
179
180     //All read operations during benchmark are executed with an HTTP 500 error.
181     @Test
182     public void testDeleteOpsBenchmarkFailure() throws InterruptedException {
183       exit.expectSystemExit();
```

```
184       exit.checkAssertionAfterwards(new Assertion() {
185         @Override
186         public void checkAssertion() throws Exception {
187           List<String> results = Utils.read(RESULTS_FILEPATH);
188           assertEquals(true, results.contains("[DELETE], Return=ERROR, 1"));
189           Utils.delete(RESULTS_FILEPATH);
190         }
191       });
192       Client.main(getArgs(ERROR_TRACE_FILEPATH, 0, 0, 0, 1));
193     }
194 
195     private String[] getArgs(String traceFilePath, float rp, float ip, float up,
          float dp) {
196       String[] args = new String[25];
197       args[0] = "-target";
198       args[1] = "1";
199       args[2] = "-t";
200       args[3] = "-P";
201       args[4] = WORKLOAD_FILEPATH;
202       args[5] = "-p";
203       args[6] = "url.prefix=http://127.0.0.1:"+port+"/webService/rest/resource/"
            ;
204       args[7] = "-p";
205       args[8] = "url.trace.read=" + traceFilePath;
206       args[9] = "-p";
207       args[10] = "url.trace.insert=" + traceFilePath;
208       args[11] = "-p";
209       args[12] = "url.trace.update=" + traceFilePath;
210       args[13] = "-p";
211       args[14] = "url.trace.delete=" + traceFilePath;
212       args[15] = "-p";
213       args[16] = "exportfile=" + RESULTS_FILEPATH;
214       args[17] = "-p";
215       args[18] = "readproportion=" + rp;
216       args[19] = "-p";
217       args[20] = "updateproportion=" + up;
218       args[21] = "-p";
219       args[22] = "deleteproportion=" + dp;
220       args[23] = "-p";
221       args[24] = "insertproportion=" + ip;
222       return args;
223     }
224 
225     private static ServletContainer resourceConfig() {
226       return new ServletContainer(new ResourceConfig(new ResourceLoader().
            getClasses()));
227     }
228 
229 }
```

B.4 REST Client Test

Appendix B.4 shows the *RestClientTest.java* in the core module that is responsible for verifying the functionality of the REST client.

```
1  package com.yahoo.ycsb.webservice.rest;
2
3  import static org.junit.Assert.assertEquals;
4
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.util.HashMap;
9  import java.util.Properties;
10
11 import javax.servlet.ServletException;
12
13 import org.apache.catalina.Context;
14 import org.apache.catalina.LifecycleException;
15 import org.apache.catalina.startup.Tomcat;
16 import org.glassfish.jersey.server.ResourceConfig;
17 import org.glassfish.jersey.servlet.ServletContainer;
18 import org.junit.AfterClass;
19 import org.junit.BeforeClass;
20 import org.junit.Test;
21
22 import com.yahoo.ycsb.ByteIterator;
23 import com.yahoo.ycsb.DBException;
24 import com.yahoo.ycsb.Status;
25 import com.yahoo.ycsb.StringByteIterator;
26
27 /**
28  * Test cases to verify the {@link RestClient} of the rest-binding
29  * module. It performs these steps in order. 1. Runs an embedded Tomcat
30  * server with a mock RESTFul web service. 2. Invokes the {@link RestClient}
31  * class for all the various methods which make HTTP calls to the mock REST
32  * service. 3. Compares the response from such calls to the mock REST
33  * service with the response expected. 4. Stops the embedded Tomcat server.
34  * Cases for verifying the handling of different HTTP status like 2xx, 4xx &
35  * 5xx have been included in success and failure test cases.
36  */
37 public class RestClientTest {
38
39   private static Integer port = 8080;
40   private static Tomcat tomcat;
41   private static RestClient rc = new RestClient();
42   private static final String RESPONSE_TAG = "response";
43   private static final String DATA_TAG = "data";
44   private static final String VALID_RESOURCE = "resource_valid";
```

```
45    private static final String INVALID_RESOURCE = "resource_invalid";
46    private static final String ABSENT_RESOURCE = "resource_absent";
47    private static final String UNAUTHORIZED_RESOURCE = "resource_unauthorized";
48    private static final String INPUT_DATA = "<field1>one</field1><field2>two</
        field2>";
49
50    @BeforeClass
51    public static void init() throws IOException, DBException, ServletException,
         LifecycleException, InterruptedException {
52      String webappDirLocation =  IntegrationTest.class.getClassLoader().
          getResource("WebContent").getPath();
53      while (!Utils.available(port)) {
54        port++;
55      }
56      tomcat = new Tomcat();
57      tomcat.setPort(Integer.valueOf(port));
58      Context context = tomcat.addWebapp("/webService", new File(
          webappDirLocation).getAbsolutePath());
59      Tomcat.addServlet(context, "jersey-container-servlet", resourceConfig());
60      context.addServletMapping("/rest/*", "jersey-container-servlet");
61      tomcat.start();
62      // Allow time for proper startup.
63      Thread.sleep(1000);
64      Properties props = new Properties();
65      props.load(new FileReader(RestClientTest.class.getClassLoader().
          getResource("workload_rest").getPath()));
66      // Update the port value in the url.prefix property.
67      props.setProperty("url.prefix", props.getProperty("url.prefix").replaceAll
          ("PORT", port.toString()));
68      rc.setProperties(props);
69      rc.init();
70    }
71
72    @AfterClass
73    public static void cleanUp() throws DBException {
74      rc.cleanup();
75    }
76
77    // Read success.
78    @Test
79    public void read_200() {
80      HashMap<String, ByteIterator> result = new HashMap<String, ByteIterator>()
          ;
81      Status status = rc.read(null, VALID_RESOURCE, null, result);
82      assertEquals(Status.OK, status);
83      assertEquals(result.get(RESPONSE_TAG).toString(), "HTTP GET response to: "
          + VALID_RESOURCE);
84    }
85
86    // Unauthorized request error.
87    @Test
```

```
88    public void read_403() {
89      HashMap<String, ByteIterator> result = new HashMap<String, ByteIterator>()
          ;
90      Status status = rc.read(null, UNAUTHORIZED_RESOURCE, null, result);
91      assertEquals(Status.FORBIDDEN, status);
92    }
93
94    //Not found error.
95    @Test
96    public void read_404() {
97      HashMap<String, ByteIterator> result = new HashMap<String, ByteIterator>()
          ;
98      Status status = rc.read(null, ABSENT_RESOURCE, null, result);
99      assertEquals(Status.NOT_FOUND, status);
100   }
101
102   // Server error.
103   @Test
104   public void read_500() {
105     HashMap<String, ByteIterator> result = new HashMap<String, ByteIterator>()
          ;
106     Status status = rc.read(null, INVALID_RESOURCE, null, result);
107     assertEquals(Status.ERROR, status);
108   }
109
110   // Insert success.
111   @Test
112   public void insert_200() {
113     HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
114     data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
115     Status status = rc.insert(null, VALID_RESOURCE, data);
116     assertEquals(Status.OK, status);
117   }
118
119   @Test
120   public void insert_403() {
121     HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
122     data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
123     Status status = rc.insert(null, UNAUTHORIZED_RESOURCE, data);
124     assertEquals(Status.FORBIDDEN, status);
125   }
126
127   @Test
128   public void insert_404() {
129     HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
130     data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
131     Status status = rc.insert(null, ABSENT_RESOURCE, data);
132     assertEquals(Status.NOT_FOUND, status);
133   }
134
135   @Test
```

```
136     public void insert_500() {
137       HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
138       data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
139       Status status = rc.insert(null, INVALID_RESOURCE, data);
140       assertEquals(Status.ERROR, status);
141     }
142
143     // Delete success.
144     @Test
145     public void delete_200() {
146       Status status = rc.delete(null, VALID_RESOURCE);
147       assertEquals(Status.OK, status);
148     }
149
150     @Test
151     public void delete_403() {
152       Status status = rc.delete(null, UNAUTHORIZED_RESOURCE);
153       assertEquals(Status.FORBIDDEN, status);
154     }
155
156     @Test
157     public void delete_404() {
158       Status status = rc.delete(null, ABSENT_RESOURCE);
159       assertEquals(Status.NOT_FOUND, status);
160     }
161
162     @Test
163     public void delete_500() {
164       Status status = rc.delete(null, INVALID_RESOURCE);
165       assertEquals(Status.ERROR, status);
166     }
167
168     @Test
169     public void update_200() {
170       HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
171       data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
172       Status status = rc.update(null, VALID_RESOURCE, data);
173       assertEquals(Status.OK, status);
174     }
175
176     @Test
177     public void update_403() {
178       HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
179       data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
180       Status status = rc.update(null, UNAUTHORIZED_RESOURCE, data);
181       assertEquals(Status.FORBIDDEN, status);
182     }
183
184     @Test
185     public void update_404() {
186       HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
```

```
187       data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
188       Status status = rc.update(null, ABSENT_RESOURCE, data);
189       assertEquals(Status.NOT_FOUND, status);
190     }
191 
192     @Test
193     public void update_500() {
194       HashMap<String, ByteIterator> data = new HashMap<String, ByteIterator>();
195       data.put(DATA_TAG, new StringByteIterator(INPUT_DATA));
196       Status status = rc.update(null, INVALID_RESOURCE, data);
197       assertEquals(Status.ERROR, status);
198     }
199 
200     @Test
201     public void scan() {
202       assertEquals(Status.NOT_IMPLEMENTED, rc.scan(null, null, 0, null, null));
203     }
204 
205     private static ServletContainer resourceConfig() {
206       return new ServletContainer(new ResourceConfig(new ResourceLoader().
            getClasses()));
207     }
208 
209 }
```

B.5 REST Test Resource

Appendix B.5 shows the *RestTestResource.java* in the core module that is responsible for creating dummy REST endpoints for testing and verification of the module.

```
 1 package com.yahoo.ycsb.webservice.rest;
 2 
 3 import javax.ws.rs.DELETE;
 4 import javax.ws.rs.GET;
 5 import javax.ws.rs.HttpMethod;
 6 import javax.ws.rs.POST;
 7 import javax.ws.rs.PUT;
 8 import javax.ws.rs.Path;
 9 import javax.ws.rs.PathParam;
10 import javax.ws.rs.Produces;
11 import javax.ws.rs.core.MediaType;
12 import javax.ws.rs.core.Response;
13 
14 /**
15  * Class that implements a mock RESTFul web service to be used for integration
16  * testing.
17  */
```

```
18 @Path("/resource/{id}")
19 public class RestTestResource {
20
21   @GET
22   @Produces(MediaType.TEXT_PLAIN)
23   public Response respondToGET(@PathParam("id") String id) {
24     return processRequests(id, HttpMethod.GET);
25   }
26
27   @POST
28   @Produces(MediaType.TEXT_PLAIN)
29   public Response respondToPOST(@PathParam("id") String id) {
30     return processRequests(id, HttpMethod.POST);
31   }
32
33   @DELETE
34   @Produces(MediaType.TEXT_PLAIN)
35   public Response respondToDELETE(@PathParam("id") String id) {
36     return processRequests(id, HttpMethod.DELETE);
37   }
38
39   @PUT
40   @Produces(MediaType.TEXT_PLAIN)
41   public Response respondToPUT(@PathParam("id") String id) {
42     return processRequests(id, HttpMethod.PUT);
43   }
44
45   private static Response processRequests(String id, String method) {
46     if (id.equals("resource_invalid"))
47       return Response.serverError().build();
48     else if (id.equals("resource_absent"))
49       return Response.status(Response.Status.NOT_FOUND).build();
50     else if (id.equals("resource_unauthorized"))
51       return Response.status(Response.Status.FORBIDDEN).build();
52     return Response.ok("HTTP " + method + " response to: " + id).build();
53   }
54 }
```

B.6 Resource Loader

Appendix B.6 shows the *ResourceLoader.java* in the core module that is responsible for loading *RestTestResource*.

```
1 package com.yahoo.ycsb.webservice.rest;
2
3 import java.util.HashSet;
4 import java.util.Set;
```

```
5
6 import javax.ws.rs.core.Application;
7
8 /**
9  * Class responsible for loading mock rest resource class like
10 * {@link RestTestResource}.
11 */
12 public class ResourceLoader extends Application {
13
14   @Override
15   public Set<Class<?>> getClasses() {
16     final Set<Class<?>> classes = new HashSet<Class<?>>();
17     classes.add(RestTestResource.class);
18     return classes;
19   }
20
21 }
```

B.7 Utils

Appendix B.7 shows the *Utils.java* in the core module that is responsible for holding common utility methods.

```
1 package com.yahoo.ycsb.webservice.rest;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileReader;
6 import java.io.IOException;
7 import java.net.DatagramSocket;
8 import java.net.ServerSocket;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 /**
13  * Holds the common utility methods.
14  */
15 public class Utils {
16
17   /**
18    * Returns true if the port is available.
19    *
20    * @param port
21    * @return isAvailable
22    */
23   public static boolean available(int port) {
24     ServerSocket ss = null;
```

```
25      DatagramSocket ds = null;
26      try {
27        ss = new ServerSocket(port);
28        ss.setReuseAddress(true);
29        ds = new DatagramSocket(port);
30        ds.setReuseAddress(true);
31        return true;
32      } catch (IOException e) {
33      } finally {
34        if (ds != null) {
35          ds.close();
36        }
37        if (ss != null) {
38          try {
39            ss.close();
40          } catch (IOException e) {
41            /* should not be thrown */
42          }
43        }
44      }
45      return false;
46    }
47
48    public static List<String> read(String filepath) {
49      List<String> list = new ArrayList<String>();
50      try {
51        BufferedReader file = new BufferedReader(new FileReader(filepath));
52        String line = null;
53        while ((line = file.readLine()) != null) {
54          list.add(line.trim());
55        }
56        file.close();
57      } catch (IOException e) {
58        e.printStackTrace();
59      }
60      return list;
61    }
62
63    public static void delete(String filepath) {
64      try {
65        new File(filepath).delete();
66      } catch (Exception e) {
67        e.printStackTrace();
68      }
69    }
70
71  }
```

Appendix C

Local Archiving and Monitoring

Appendix C shows the local archiving service that we developed and the real time monitoring system that we configured as a part of this research.

C.1 Local Archiving Service Using Persistent Files

Appendix C.1 shows the *LocalArchivingService.php* that is responsible for accepting a Web archiving request from the SiteStory module and persists it locally on the file system.

```
1  <?php
2
3     /**
4     * LocalArchivingService.php
5     *
6     * Cache service that reads HTTP PUT request data and
7     * persists it locally on file-system.
8     *
9     * @author: shivam.maharshi
10    */
11
12    include '/var/www/html/apache-log4php-2.3.0/src/main/php/Logger.php';
13
14    Logger::configure('config.xml');
15    $logger = Logger::getLogger("myLogger");
16    error_reporting(E_ALL);
17
18    $file = '/var/www/html/archive/'.md5("http://".explode('http://', $_SERVER['
        REQUEST_URI'])[1])."txt";
19
20    if(!file_exists($file)){
21        $handle = fopen('php://input', 'rb');
```

```
22          $data = '';
23
24          if($handle) {
25              while(($buffer = fgets($handle, 4096)) !== false) {
26                   $data .= $buffer;
27              }
28              if(!feof($handle)) {
29                  $logger->error("Error: unexpected fgets() fail.");
30              }
31          }
32
33          file_put_contents($file, $data);
34          fclose($handle);
35      }
36  ?>
```

C.2 Local Archiving Service Using RAM as Store

Appendix C.2 shows the *LocalArchivingServiceUsingRAM.php* that is responsible for accepting a Web archiving request from the SiteStory module and persists it locally on the file system.

```
1   <?php
2
3     /**
4     * LocalArchivingServiceUsingRAM.php
5     *
6     * Cache service that reads HTTP PUT request data and
7     * persists it locally on file-system.
8     *
9     * @author: shivam.maharshi
10    */
11
12    include '/var/www/html/apache-log4php 2.3.0/src/main/php/Logger.php';
13    require "/var/www/html/predis/autoload.php";
14    Predis\Autoloader::register();
15
16    error_reporting(E_ALL);
17
18    Logger::configure('config.xml');
19    $log = Logger::getLogger("myLog");
20
21    $key = md5(explode('http://', $_SERVER['REQUEST_URI'])[1]);
22    $redis = new Predis\Client(array("schema" => "tcp", "host" => "127.0.0.1", "
        port" => "6379"));
23    $data = '';
24    $handle = fopen('php://input', 'rb');
```

```
25
26    if($handle) {
27        while(($buffer = fgets($handle, 4096)) !== false) {
28            $data .= $buffer;
29        }
30        if(!feof($handle)) {
31            $logger->error("Error: unexpected fgets() fail.");
32        }
33    }
34
35    $redis->set($key, $data);
36    $redis->disconnect();
37
38 ?>
```

C.3 Monitor Resources Script

Appendix C.3 shows the *monitor-resources.sh* shell script that is responsible for capturing the system resource usage and appending it to a resource.txt file periodically every ten seconds.

```
1  collectl -P -scmdn -oT -oD --interval=10 > /home/shivam/monitor/resource.txt
```

C.4 Logstash Startup Script

Appendix C.4 shows the *logstash.sh* shell script that is responsible for starting a Logstash service that reads the recently appended data from the resource.txt file, performs data manipulation on it, and sends it over to a remote Elasticsearch server for indexing.

```
1  /opt/logstash/bin/logstash -e 'input { file { path => [ "/home/shivam/monitor
       /*.txt" ] type => "collectl" } } filter { if [type] == "collectl" {
2      csv {
3        separator => " "
4      }
5      mutate {
6        remove_field => [column3, column4, column6, column7, column8, column9,
             column10, column12, column13, column14, column15, column16, column17
             , column18, column19, column20, column21, column22, column23,
             column24, column26, column27, column28, column29, column30, column31
             , column32, column33, column34, column35, column36, column37,
             column38, column39, column40, column41, column42, column43,
             column44, column45, column46, column47, column48, column49, column50
```

```
, column51, column52, column53, column56, column57, column58,
   column59, column60, column61, column62, column63, column66, column67
   , column68, column69]
7      rename => ["column1", "DATE", "column2", "TIME", "column11", "CPU", "
          column25", "RAM", "column54", "NET_RX_KB", "column55", "NET_TX_KB",
          "column64", "DSK_RD_KB", "column65", "DSK_WT_KB"]
8      convert => {
9                  "CPU" => "integer"
10     "RAM" => "integer"
11                 "NET_RX_KB" => "integer"
12                 "NET_TX_KB" => "integer"
13                 "DSK_RD_KB" => "integer"
14                 "DSK_WT_KB" => "integer"
15             }
16   } } } output { elasticsearch { action => "index" index => "logstash-%{+
        YYYY.MM.dd}" hosts => "192.168.1.52" } }'
```

C.5 Logstash Visualization Configuration

Appendix C.5 shows the *logstash-viz.json* configuration file that creates six histograms to visualize CPU, RAM, Network Read Bandwidth, Network Write Bandwidth, Disk Read Bandwidth, and Disk Write Bandwidth.

```
1  [
2    {
3      "_id": "CPU",
4      "_type": "visualization",
5      "_source": {
6        "title": "CPU",
7        "visState": "{\"type\":\"line\",\"params\":{\"shareYAxis\":true,\"
             addTooltip\":true,\"addLegend\":true,\"showCircles\":true,\"
             smoothLines\":false,\"interpolate\":\"linear\",\"scale\":\"linear\"
             ,\"drawLinesBetweenPoints\":true,\"radiusRatio\":9,\"times\":[],\"
             addTimeMarker\":false,\"defaultYExtents\":true,\"setYExtents\":false
             ,\"yAxis\":{}},\"aggs\":[{\"id\":\"1\",\"type\":\"avg\",\"schema\":\
             "metric\",\"params\":{\"field\":\"CPU\"}},{\"id\":\"2\",\"type\":\"
             date_histogram\",\"schema\":\"segment\",\"params\":{\"field\":\"
             @timestamp\",\"interval\":\"auto\",\"customInterval\":\"2h\",\"
             min_doc_count\":1,\"extended_bounds\":{}}}],\"listeners\":{}}",
8        "uiStateJSON": "{}",
9        "description": "",
10       "version": 1,
11       "kibanaSavedObjectMeta": {
12         "searchSourceJSON": "{\"index\":\"logstash-*\",\"query\":{\"
              query_string\":{\"query\":\"*\",\"analyze_wildcard\":true}},\"
              filter\":[]}"
13       }
```

```
14        }
15      },
16      {
17        "_id": "NTW_READ",
18        "_type": "visualization",
19        "_source": {
20          "title": "NTW_READ",
21          "visState": "{\"type\":\"line\",\"params\":{\"shareYAxis\":true,\"
              addTooltip\":true,\"addLegend\":true,\"showCircles\":true,\"
              smoothLines\":false,\"interpolate\":\"linear\",\"scale\":\"linear\"
              ,\"drawLinesBetweenPoints\":true,\"radiusRatio\":9,\"times\":[],\"
              addTimeMarker\":false,\"defaultYExtents\":true,\"setYExtents\":false
              ,\"yAxis\":{}},\"aggs\":[{\"id\":\"1\",\"type\":\"avg\",\"schema\":\
              "metric\",\"params\":{\"field\":\"NET_RX_KB\"}},{\"id\":\"2\",\"type
              \":\"date_histogram\",\"schema\":\"segment\",\"params\":{\"field\":\
              "@timestamp\",\"interval\":\"auto\",\"customInterval\":\"2h\",\"
              min_doc_count\":1,\"extended_bounds\":{}}}],\"listeners\":{}}",
22          "uiStateJSON": "{}",
23          "description": "",
24          "version": 1,
25          "kibanaSavedObjectMeta": {
26            "searchSourceJSON": "{\"index\":\"logstash-*\",\"query\":{\"
                query_string\":{\"query\":\"*\",\"analyze_wildcard\":true}},\"
                filter\":[]}"
27          }
28        }
29      },
30      {
31        "_id": "DISK_READ",
32        "_type": "visualization",
33        "_source": {
34          "title": "DISK_READ",
35          "visState": "{\"type\":\"line\",\"params\":{\"shareYAxis\":true,\"
              addTooltip\":true,\"addLegend\":true,\"showCircles\":true,\"
              smoothLines\":false,\"interpolate\":\"linear\",\"scale\":\"linear\"
              ,\"drawLinesBetweenPoints\":true,\"radiusRatio\":9,\"times\":[],\"
              addTimeMarker\":false,\"defaultYExtents\":true,\"setYExtents\":false
              ,\"yAxis\":{}},\"aggs\":[{\"id\":\"1\",\"type\":\"avg\",\"schema\":\
              "metric\",\"params\":{\"field\":\"DSK_RD_KB\"}},{\"id\":\"2\",\"type
              \":\"date_histogram\",\"schema\":\"segment\",\"params\":{\"field\":\
              "@timestamp\",\"interval\":\"auto\",\"customInterval\":\"2h\",\"
              min_doc_count\":1,\"extended_bounds\":{}}}],\"listeners\":{}}",
36          "uiStateJSON": "{}",
37          "description": "",
38          "version": 1,
39          "kibanaSavedObjectMeta": {
40            "searchSourceJSON": "{\"index\":\"logstash-*\",\"query\":{\"
                query_string\":{\"query\":\"*\",\"analyze_wildcard\":true}},\"
                filter\":[]}"
41          }
42        }
```

```
43    },
44    {
45      "_id": "NTW_TX",
46      "_type": "visualization",
47      "_source": {
48        "title": "NTW_TX",
49        "visState": "{\"type\":\"line\",\"params\":{\"shareYAxis\":true,\"
             addTooltip\":true,\"addLegend\":true,\"showCircles\":true,\"
             smoothLines\":false,\"interpolate\":\"linear\",\"scale\":\"linear\"
             ,\"drawLinesBetweenPoints\":true,\"radiusRatio\":9,\"times\":[],\"
             addTimeMarker\":false,\"defaultYExtents\":true,\"setYExtents\":false
             ,\"yAxis\":{}},\"aggs\":[{\"id\":\"1\",\"type\":\"avg\",\"schema\":\
             "metric\",\"params\":{\"field\":\"NET_TX_KB\"}},{\"id\":\"2\",\"type
             \":\"date_histogram\",\"schema\":\"segment\",\"params\":{\"field\":\
             "@timestamp\",\"interval\":\"auto\",\"customInterval\":\"2h\",\"
             min_doc_count\":1,\"extended_bounds\":{}}}],\"listeners\":{}}",
50        "uiStateJSON": "{}",
51        "description": "",
52        "version": 1,
53        "kibanaSavedObjectMeta": {
54          "searchSourceJSON": "{\"index\":\"logstash-*\",\"query\":{\"
               query_string\":{\"query\":\"*\",\"analyze_wildcard\":true}},\"
               filter\":[]}"
55        }
56      }
57    },
58    {
59      "_id": "DISK_WRITE",
60      "_type": "visualization",
61      "_source": {
62        "title": "DISK_WRITE",
63        "visState": "{\"type\":\"line\",\"params\":{\"shareYAxis\":true,\"
             addTooltip\":true,\"addLegend\":true,\"showCircles\":true,\"
             smoothLines\":false,\"interpolate\":\"linear\",\"scale\":\"linear\"
             ,\"drawLinesBetweenPoints\":true,\"radiusRatio\":9,\"times\":[],\"
             addTimeMarker\":false,\"defaultYExtents\":true,\"setYExtents\":false
             ,\"yAxis\":{}},\"aggs\":[{\"id\":\"1\",\"type\":\"avg\",\"schema\":\
             "metric\",\"params\":{\"field\":\"DSK_WT_KB\"}},{\"id\":\"2\",\"type
             \":\"date_histogram\",\"schema\":\"segment\",\"params\":{\"field\":\
             "@timestamp\",\"interval\":\"auto\",\"customInterval\":\"2h\",\"
             min_doc_count\":1,\"extended_bounds\":{}}}],\"listeners\":{}}",
64        "uiStateJSON": "{}",
65        "description": "",
66        "version": 1,
67        "kibanaSavedObjectMeta": {
68          "searchSourceJSON": "{\"index\":\"logstash-*\",\"query\":{\"
               query_string\":{\"query\":\"*\",\"analyze_wildcard\":true}},\"
               filter\":[]}"
69        }
70      }
71    },
```

```
72    {
73      "_id": "RAM",
74      "_type": "visualization",
75      "_source": {
76        "title": "RAM",
77        "visState": "{\"type\":\"line\",\"params\":{\"addLegend\":true,\"
             addTimeMarker\":false,\"addTooltip\":true,\"defaultYExtents\":false
             ,\"drawLinesBetweenPoints\":true,\"interpolate\":\"linear\",\"
             radiusRatio\":9,\"scale\":\"linear\",\"setYExtents\":false,\"
             shareYAxis\":true,\"showCircles\":true,\"smoothLines\":false,\"times
             \":[],\"yAxis\":{}},\"aggs\":[{\"id\":\"1\",\"type\":\"avg\",\"
             schema\":\"metric\",\"params\":{\"field\":\"RAM\"}},{\"id\":\"2\",\"
             type\":\"date_histogram\",\"schema\":\"segment\",\"params\":{\"field
             \":\"@timestamp\",\"interval\":\"auto\",\"customInterval\":\"2h\",\"
             min_doc_count\":1,\"extended_bounds\":{}}}],\"listeners\":{}}",
78        "uiStateJSON": "{}",
79        "description": "",
80        "version": 1,
81        "kibanaSavedObjectMeta": {
82          "searchSourceJSON": "{\"index\":\"logstash-*\",\"query\":{\"
               query_string\":{\"analyze_wildcard\":true,\"query\":\"*\"}},\"
               filter\":[]}"
83        }
84      }
85    }
86  ]
```

C.6 Logstash Dashboard Configuration

Appendix C.6 shows the *logstash-dash.json* configuration file that creates a dashboard with the six visualizations mentioned in Appendix C.5.

```
1  [
2    {
3      "_id": "wiki",
4      "_type": "dashboard",
5      "_source": {
6        "title": "wiki",
7        "hits": 0,
8        "description": "",
9        "panelsJSON": "[{\"col\":1,\"id\":\"CPU\",\"panelIndex\":1,\"row\":1,\"
             size_x\":6,\"size_y\":4,\"type\":\"visualization\"},{\"col\":7,\"id\
             ":\"DISK_READ\",\"panelIndex\":2,\"row\":5,\"size_x\":6,\"size_y\"
             :3,\"type\":\"visualization\"},{\"col\":7,\"id\":\"DISK_WRITE\",\"
             panelIndex\":3,\"row\":8,\"size_x\":6,\"size_y\":3,\"type\":\"
             visualization\"},{\"col\":1,\"id\":\"NTW_READ\",\"panelIndex\":4,\"
             row\":5,\"size_x\":6,\"size_y\":3,\"type\":\"visualization\"},{\"col
```

```
              \":1,\"id\":\"NTW_TX\",\"panelIndex\":5,\"row\":8,\"size_x\":6,\"
              size_y\":3,\"type\":\"visualization\"},{\"id\":\"RAM\",\"type\":\"
              visualization\",\"panelIndex\":6,\"size_x\":6,\"size_y\":4,\"col\"
              :7,\"row\":1}]",
10          "optionsJSON": "{\"darkTheme\":true}",
11          "uiStateJSON": "{}",
12          "version": 1,
13          "timeRestore": false,
14          "kibanaSavedObjectMeta": {
15            "searchSourceJSON": "{\"filter\":[{\"query\":{\"query_string\":{\"
                analyze_wildcard\":true,\"query\":\"*\"}}}]}"
16          }
17        }
18      }
19    ]
```