Quantifying Resource Sharing, Resource Isolation and Agility for Web Applications with Virtual Machines[1]

M.S. Thesis

Elliot Miller

Advisor
Craig Wills

Reader
Mark Claypool

Computer Science Department
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609

August, 2007

Abstract

Resource sharing between applications can significantly improve the resources required for all, which can reduce cost, and improve performance. Isolating resources on the other hand can also be beneficial as the failure or significant load on one application does not affect another. There is a delicate balance between resource sharing and resource isolation. Virtual machines may be a solution to this problem with the added benefit of being able to perform more dynamic load balancing, but this solution may be at a significant cost in performance. This thesis compares three different configurations for machines running application servers. It looks at speed at which a new application server can be started up, resource sharing and resource isolation between applications in an attempt to quantify the tradeoffs for each type of configuration.

[1] This material is based upon work supported by the National Science Foundation under Grant No. 0615079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Any industry that consumes resources to provide a service will be the most profitable when it uses resources efficiently. If too many resources are allocated, then not all will be used and money will be wasted. If too few are allocated, the industry will not be able to service every customer and money will be lost as people look elsewhere. In most industries demand changes at a relatively slow pace allowing a fairly good prediction of the resource allocation needed. However, in the industry of web service providers, demand fluctuates rapidly and therefore so does resource usage. It is therefore imperative that efficient allocation of resources be a large concern for this industry. It is logical to want to share resources between web applications so that when the demand for service from one application suddenly skyrockets, the overloaded application can utilize resources that an underutilized service does not need. This approach is the idea of utility computing. Andrzejak et al. [1] found that reallocating resources dynamically can cut needed resources for a service provider in half.

These benefits would lead one to believe that resources should be shared as much as possible, however there is a problem. What happens if one of the web services becomes subject to a denial of service attack, or a memory leak, or even simply becomes too overloaded? The service will continuously utilize a larger and larger portion of the available resources until none are left, not only slowing down its own service to the web, but also all other applications running on the same machine.

For this reason, all web service providers must strike a balance between resource sharing and resource isolation. Too much of either can be problematic. Virtual machines

may provide a unique solution to this balancing act. Each virtual machine is isolated from both the physical machine and any other virtual machines also running on the physical machine. However, virtual machines do still share resources with each other through time-sharing of the physical machine. Virtual machines also allow the means to be more agile. The agility of a network is the ability to react quickly to changes in the network. Virtual machines provide the ability to start up quickly, and can also be easily moved between physical machines. The benefits are clearly large, but what is the cost in performance? When using multiple virtual machines one is running multiple operating systems on top of the same physical machine, which could be a significant performance problem. The goal of this thesis is to quantify the benefits and disadvantages of virtual machines to allow an informed decision to be made of whether to use them or not.

In order to quantify these differences we have looked at three different configurations for application servers in a distributed network architecture:

1. Different applications running on different application servers that are running on the same physical machine.
2. Different applications running on different applications servers each running on its own virtual machine that are running on the same physical machine.
3. Applications running on completely separate physical machines.

For each configuration we evaluated it in three categories: performance, resource isolation and agility. By doing so we were able to better understand the advantages and disadvantages of virtual machines and compare them with other standard non-virtual configurations. We were able to show that not only did virtual machines significantly

improve the agility of the system and resource isolation of applications, but it also did not have a significant impact on performance.

We will start in Chapter 2 by reviewing the technologies utilized in this study, after which, in Chapter 3, there will be a discussion of what we plan to accomplish in this study and how we plan to complete these goals. Chapter 4 will then discuss in detail each layer in the network architecture. With an understanding of the network architecture, Chapter 5 will proceed to discuss the methodology that was used to accomplish the goals set forth in Chapter 3. Chapter 6 will then list our results and analyze them. Lastly we will summarize our analysis in Chapter 7, discuss the conclusions this analysis brought us to and also future work that should be explored.

2 Background

It is important to understand the many technologies utilized in this study. This chapter contains a brief overview of all of them.

2.1 *J2EE*

J2EE (Java 2 Enterprise Edition) is a Java platform developed by Sun Microsystems. It is a platform used for creating thin client web applications; applications that perform the majority of their processing on the server side. It has become a standard in web development and is highly utilized for large business web applications. J2EE enables the programmer to modularize code allowing for re-use.

It also encourages the Model View Controller (MVC) architecture; the act of breaking a web application up into layers, the model, the view and the controller. The model is the layer that interacts with the database, the view is the layout of the pages that the user will view and the controller provides the business logic to link the other two layers together. The MVC architecture allows web applications to be organized in a manner that is highly necessary in a large application as it makes the code re-usable, easy to understand, and easy to create.

J2EE relies on servlet code to craft together HTML pages which means that excluding JavaScript, all processing is performed on the server, and none on the client aside from interpreting the HTML.

2.2 *Application Servers*

In order to run a J2EE application, an application server is needed. There are different types of application servers, but the term application server has widely become identified with a container for J2EE applications. An application server is essentially a software engine that allows access to a server-side application by a client. This engine offers the programmer the building blocks for web applications including the interaction between server and client, security, database interaction, and interpretation of servlets. A servlet such as a J2EE's JavaServer Pages (JSP) allows the programmer to embed business logic and database interaction within an HTML page, essentially the Java equivalent of CGI scripts. There are many different J2EE application servers, both open source and proprietary, including: JBoss [12], WebSphere [10], WebLogic Server [23], Tomcat [3], JOnAS [13].

Some application servers such as Tomcat provide simply a servlet container, the means for interpreting and processing servlets and communicating with the client. Other application servers add capabilities such as support for distributed processing and clustering.

2.2.1 Clustering

When a large scale web application gets utilized by enough clients, a single server is not sufficient to cope with the processing of requests. Therefore a move to a distributed model is necessary. The problem that arises when in a distributed environment is maintaining session data, the information about a client's current status in the application. If one of the servers goes down then this data is lost. Requests destined for the application must also always go to the same application server that is hosting the

session data for that user. This process means that all the application servers running that application need to in some way know about each other forming a 'cluster' of application servers.

A cluster can be formed in a couple of ways, there can be a single controller that acts as a gateway, maintaining session data and/or forwarding requests to the appropriate server, or the session data can be shared across all instances of the application server. There are problems with both models however. The controller model has the potential to create a bottleneck, whereas the lack of a controller means that a lot of network traffic is generated to pass the data between servers as it is updated. This data can also be stored in a database but storing this data also creates a lot of extra requests to the database and can slow down processing by the server.

2.3 *Content Services Switch*

Another option to clustering is using a content services switch. A content services switch is a piece of hardware that performs load balancing between application servers. It acts like the controller in a cluster, however it allows one to load balance between many different services, not necessarily all application servers. A content services switch allows one to define specific load-balancing rules for various services. Rules can be defined on a per application basis so that multiple clusters are formed under the same controller, possibly overlapping with each other. Content services switches are useful in a large business where multiple large web-based applications are hosted in a distributed environment.

2.4 *Virtual Machines*

Another approach we explore is the use of virtualization. "Virtualization is an abstraction layer that decouples the physical hardware from the operating system to deliver greater IT resource utilization and flexibility. Virtualization allows multiple virtual machines, with heterogeneous operating systems to run in isolation, side-by-side on the same physical machine. Each virtual machine has its own set of virtual hardware (e.g., RAM, CPU, NIC, etc.) upon which an operating system and applications are loaded. The operating system sees a consistent, normalized set of hardware regardless of the actual physical hardware components"[1].

The benefit of virtualization is not only that each machine is isolated from one another so that the crash of one does not affect the others, but it also allows for more dynamic responses. The virtual machine can be started up and configured with all the necessary applications running and then put in a suspended state in which it uses minimal resources to maintain itself. It can be brought out of this suspended state much quicker than it would take to boot up the machine from scratch allowing new resources to become available much faster. All of the information is stored on disk in a single file, so that it makes it easy to move between machines. Depending on the size of the virtual machine and the speed of the connection this transfer could take a short while, but it is an invaluable in the unique ability to move resources and applications between physical machines.

2.5 *Network Architecture*

In an environment where one application server is not sufficient to handle the entirety of incoming requests, multiple application servers must be utilized. When multiple servers are involved there must also be a way to distribute requests in a manor that does not leave one machine overloaded while others are underutilized. The architecture of a typical network involving large-scale web applications is shown in Figure 1. A large number of clients send HTTP requests to a content services switch which is responsible for distributing requests to one of potentially many application servers sitting behind it.

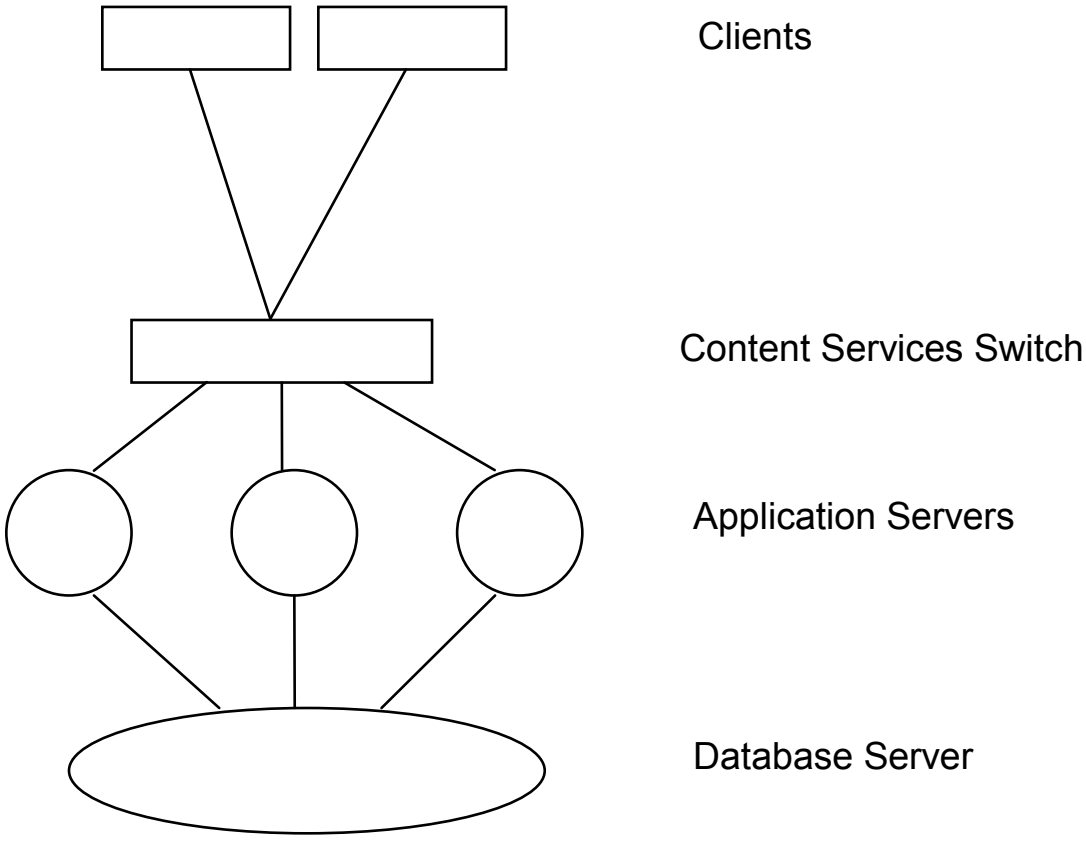

Figure 1: Typical Network Structure for Large-Scale Business Web Applications

These application servers then process the requests often exchanging information with a back-end database. In Figure 1, the application servers represented by the circles are not limited to representing a single physical machine. Depending on the configuration chosen, one may have multiple application servers running on the same physical machine, or on virtual machines residing on a single physical machine. Each circle simply represents an IP address and port number that web requests can be sent to for processing and response.

2.6 *Summary*

This chapter has discussed the technologies necessary to understand all aspects of this study. This chapter discussed J2EE and application servers, including a discussion of clustering and its advantages and disadvantages. There was also a discussion of content services switches and what they are used for. The reader also learned about what virtual machines are and why they are useful. Lastly there was a discussion of how all of these components fit together into a full network architecture that can service many requests for a large business web application.

3 Approach

Efficient allocation of web resources is a crucial component to commerce on the web. If a service provider does not allocate enough resources then customers will be lost, if too many resources are allocated then money will be wasted. Due to the fact that traffic on the web often comes in surges a single application may be idle while another is overloaded and vice versa. To avoid over-allocating resources it makes the most sense to share resources between applications which alleviates the need to allocate a large amount of resources for the potentially short period of time that the application is overloaded. This idea of utility computing shows significant benefits to service providers. Andrzejak et al. [1] found that reallocating resources dynamically can cut resources for a service provider in half.

Conversely, it is important to keep resources separate as one does not want the operation of an application to interfere with another. If an application becomes overloaded, crashes, is subject to a denial of service attack or has security flaws one does not want this problem to negatively affect another application running on the same machine. By isolating resources one can guarantee that no application will influence another.

The key is balance between resource sharing and isolation as each individually can be undesirable as described above. Virtual machines may be a prospective solution to this problem. Resources within a virtual machine are in essence isolated from the physical machine. If there are multiple VMs running on the same physical machine resources are isolated from one another, but at the same time they are sharing resources through time-sharing of the physical machine. The potential problem is that with

multiple virtual machines there are essentially multiple operating systems running at the same time off of the same physical resources, which has the potential to cause a serious slowdown in responsiveness of the application as more resources may be required than are available. Part of the goal of this project is to explore this avenue and look at the difference in required resources with the addition of virtual machines.

Virtual machines may have the added benefit of being more agile when it comes to load balancing. Using a virtual machine one is able to already have the virtual machine and application server running, but in a suspended state. This ability means that in the case that a particular application is overloaded with requests, starting up a new application server can be much quicker as it is a matter of releasing the virtual machine from the suspended state instead of taking the time to start up a new application server. Using virtual machines one can also migrate a virtual machine from one physical machine to another. This ability means that if a physical machine is idle and another has a number of application servers running on it and is overloaded, the virtual machine can be transferred to the idle physical machine.

Balancing resource isolation, resource sharing and agility is difficult and it is important to quantify the tradeoffs so that an informed decision can be made as to which is more important. A number of server configurations have been proposed that provide varied resource isolation and sharing, specifically:

1. Different applications running on different application servers that are running on the same physical machine. This configuration was demonstrated by Pacifici et al. [15] who described a load balancing system to balance between multiple servers

effectively by using a global resource manager to monitor performance of individual application servers.

2. Different applications running on different applications servers each running on its own virtual machine that are running on the same physical machine. This configuration was discussed by Whitaker et al. [24] in their paper about the Denali project which discussed using virtual machines to 'push' untrusted web services into third party hosting infrastructure, demonstrating the security benefits of virtual machines. This configuration was also proposed by Figueiredo et al. [8] in their paper that discusses using virtual machines as an abstraction layer for grid computing, thus isolating resources, providing extra security and allowing legacy support. Another group that proposed this configuration was Clark et al. [7] in their discussion of the benefits of virtual machines in the ability to migrate them between different physical machines. Lastly, Gupta et al. [9] also discussed this configuration when talking about SEDF-DC, a system that controls total CPU utilization across a number of virtual machines.
3. Applications running on different physical machines. This configuration provides the best resource isolation but at the same time does not provide efficient resource sharing. This configuration is discussed by Chase et al. [6] in their proposal for "a new cluster management architecture for mixed-use clusters", where they discuss how to share resources between grid services.

For the purposes of this paper we have named each of the configurations for easy identification, the first is heretofore called the "phys" configuration. The second configuration is called the virt configuration and the third is called the "diffphys"

configuration. For this thesis the focus is on quantifying the agility of the configuration, that is how quickly the overloaded system is able to start up a new application to relieve stress on those instances currently running.

There has been previous work done describing how the configurations are different such as in Canali et al. [5], but not much research into quantifying the differences. The goal of this project is to gain an understanding of different possible configurations that can be used to build shared utility computing platforms for J2EE web services and to put them in a common context for easy comparison.

The goal of this project is to examine the three configurations discussed in this chapter and compare them in terms of agility, performance and resource isolation. The data gleaned in the research can be plotted in a graph such as Figure 2. Ideally, one would desire high values each.

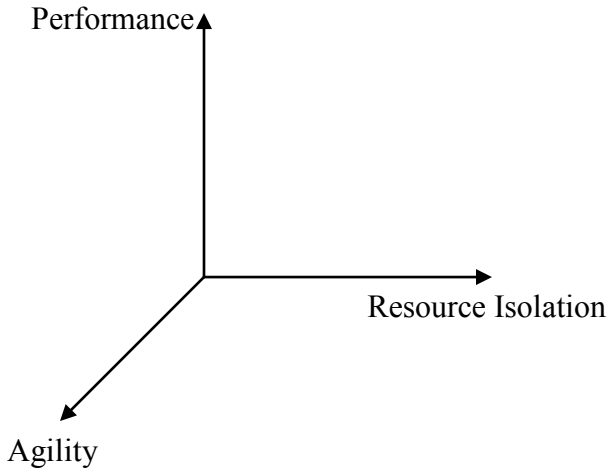

Figure 2: Dimensions of Comparison

These configurations represent base cases that other combinations can be built upon such as multiple virtual machines running on multiple physical machines, a

combination of virt and diffphys configurations. In attempting to quantify the differences between these three configurations there are some questions that need to be answered.

- What is the time difference between configurations in bringing up a new application server?
- What is the performance difference between configurations?
- What is the difference in resource isolation between configurations?

The following sections describe our approach for answering these questions.

3.1 *Agility*

The agility of a network is the ability to react quickly to changes in the network. This reaction can mean bringing a new server up in the event of a significant increase in load, it could mean taking down a server in the case that it is no longer needed, or it could mean reacting quickly when a server crashes by redirecting requests and restoring session data. For this particular study we focused specifically on the ability to react to an increase in load. Network traffic often arrives in bursts, and all of a sudden a once underutilized application server can become overloaded. The system needs to be able to respond to this sudden increase in load quickly so that performance does not suffer. Also, because of the variable amount of traffic, if a new server takes too long to bring online, by the time it is ready, the need for extra processing could be gone. In this sense the measure of how agile a system is, is the length of time it takes to start up a new application server and have it be ready to service requests. We simply measured the time it took to start a server and retrieve a response from an application hosted by it. This test only includes the time it takes to react, it does not encompass the time the system takes to decide a reaction is necessary, nor the process followed to make such a decision.

The goal of this study is ultimately to investigate virtual machines and their ability to start up faster from a suspended state than a normal application server would take. This time it takes for the virtual machine to be ready to serve requests must be compared against the other configurations discussed in order to document a baseline for the difference between them.

3.2 *Performance*

Performance of an application server can fall under two categories, the resources used by the application server on the host machine and the perceived performance of the application server by the client. Though these two categories are related, a user on the client side requires a much larger performance hit in order to notice that the application is running slowly. Ultimately it is the user's perception of performance that really matters, but server performance data is needed to back this client data up. Also, an increase in load on the server may not necessarily affect the response time of the server, in which case this load increase is not as big of a problem.

The goal of this study is to determine the performance difference, both on the client and server side of the network that virtual machines bring to bear. If the benefits of virtual machines do not outweigh the problems, then they are not worth using and a performance problem is one of the biggest concerns.

3.3 *Resource Isolation*

Measuring resource isolation is much more difficult to do. It is hard to measure how faults in an application have an impact on other application servers, but it is slightly easier to see how load has an impact on other servers. If the resources utilized by two

application servers are shared and one server starts using up all of them, the other will suffer in performance. Conversely if the resources are isolated, the non-offending server will not show a performance difference. Clearly one wants resources to be isolated so that this performance problem does not occur, but one also does not want to waste resources and if a server is under loaded the resources that could be used for processing on another application server instead sit idle.

Virtual machines present a potential middle ground between sharing and isolation of resources. The goal of this study is to attempt to quantify the amount of resource isolation that virtual machines give in relation to the other configurations studied. In order to understand the advantages of virtual machines this benefit must be understood.

3.4 *Summary*

This chapter discussed the approach to quantifying how virtual machines might impact distributed application servers. Specifically this study is looking at three aspects: performance, resource isolation and agility. There are three types of configurations this study will look at: multiple application servers on the same physical machine, multiple virtual machines running single application servers on a single virtual machine and single application servers on multiple physical machines. Agility, is only being looked at in terms of starting up a new server in the case of increased load. Agility will be measured as the time it takes to start up a new server in the given configuration. Performance will be measured from both the user's perspective and the server's perspective. Resource isolation will be measured as the impact of an overloaded server on the performance of another server that is not overloaded.

4 Testing Environment

In order to understand the specific methodology that was used to implement the approach described above, an understanding of the different layers of the network architecture is necessary. The following sections describe the technologies used for all the layers discussed in Figure 1.

4.1 *Clients*

The clients in this model are any computer making HTTP requests to the web application. This model could even include requests from an application on the same machine as the web application. The user can also be human or automated. In the network architecture described in Chapter 3 we are focusing on human users on external machines as this architecture is the majority usage of business web applications. To simulate this layer of the architecture of the network we used the client simulation ability of the benchmark application TPC-W.

4.1.1 TPC-W

There are a number of benchmark applications for J2EE application servers, including: Java Pet Store, IBM's Trade2, TPC-W and SpecJAppServer2004. All of these applications give the ability to collect information about the application server as they are run. However, only TPC-W and SpecJAppServer2004 are fully automated. That is, they have the ability to create a number of client sessions that can make HTTP requests to the application server and during this process collect data about the responsiveness of the server. As automation is important in order to accomplish the data collection desired for

this project in a timely fashion, one of these two would be ideal. TPC-W has the added benefit of being free which has led to the conclusion that TPC-W is the benchmark to use.

TPC-W is a specification developed by the Transaction Processing Performance Council [21]. It simulates an e-commerce application to test a number of aspects of an application server. Specifically:

> "The workload exercises a breadth of system components associated with such [e-commerce] environments, which are characterized by:
>
> - Multiple on-line browser sessions
> - Dynamic page generation with database access and update
> - Consistent web objects
> - The simultaneous execution of multiple transaction types that span a breadth of complexity
> - On-line transaction execution modes
> - Databases consisting of many tables with a wide variety of sizes, attributes, and relationships
> - Transaction integrity (ACID properties)
> - Contention on data access and update" [20]

One of the benefits of TPC-W is that it allows automated testing of the server. As well as being able to run TPC-W through the normal browser interface, it supplies a command line interface which can simulate multiple users and actions at the same time, automating interaction with the application. It produces a log of the interactions so that one can review statistics about the application server.

TPC-W is actually a two-piece application, the web application, and a command line java utility that is used for client simulation.

4.1.2 TPC-W Emulated Browser Utility

This utility creates a large number of emulated browsers, specified by command line arguments, each of which emulates the requests made by a user session. Each

emulated browser is a separate thread, spawned off from the initial utility. When each thread completes the utility spawns a new thread in its place to keep the total number of emulated browsers at the specified amount. The utility goes through three phases during the course of its runtime. When initially started, it goes through a ramp-up period, defaulted at 600 seconds, where it spawns off all of the emulated browsers and runs for a short while so the throughput of outgoing HTTP requests can stabilize. During this time all data recorded by the emulated browsers is dropped and not recorded. Once the ramp-up period is over, the measurement interval begins in which the data returned from the emulated browsers is recorded into a log file. The measurement is defaulted to 1800 seconds. After the measurement interval ends, the ramp-down period begins, defaulted to 300 seconds, in which the utility maintains a steady state before gracefully shutting down the threads.

4.1.3 Emulated Browsers

Each emulated browser makes a request, receives a response, logs data and then waits for a small amount of time before making another request. This idle time or "think time" is intended to emulate a human user who needs time to process the web page returned to their browser and decide on the next course of action. TPC-W uses the formula: $T = - \ln(r) * \mu$ where r is a random number, with at least 31 bits of precision, from a uniform distribution such that $(0 < r \leq 1)$ and μ is between 7 and 8 seconds inclusive [21]. TPC-W also sets a maximum on the think time that is 10 times μ. Using this model TPC-W more accurately reflects a typical user session where requests from a client come intermittently.

Also of note is that the TPC-W client utility can create three types of sessions, browsing, shopping and ordering. This parameter controls the type of session for *all* of the emulated browser generated by the utility. It determines whether the user is going to the web application to browse for items, order items or do a combination of both. For the purposes of this study the browsing session type was chosen to use as it makes large calls to the database and requires little session state management, which was not a factor in this study.

4.1.4 Recorded Data

The TPC-W emulated browser utility records data returned by each thread after each HTTP response. The particular data utilized for this study was the web interaction response time (WIRT). The WIRT is a measurement of the time from the first byte of data of the request being sent until the last byte of the response is received [59]. This piece of data was considered the most relevant that is returned by the emulated browser as it is a measurement of the time it takes for the application to respond and therefore is an indication of the user's perception of load on the server.

4.2 *Content Services Switch*

In order to load balance between multiple application servers we utilized a Content Services Switch (CSS) manufactured by Cisco Systems. This switch was responsible for doing level 3, 4 or 5 switching to the application servers. Requests coming into the switch were forwarded to the application server based on load. Figure 3 is a basic diagram of the setup of the Cisco CSS. Incoming requests are directed to content rules within the CSS. Each content rule has zero or more services registered with

it, represent application servers to which requests can be directed. The content rule decides which service to forward the request to, based on the load-balancing rules configured for it. As this study was not focused on different load-balancing methods a basic round-robin method of load-balancing was utilized.

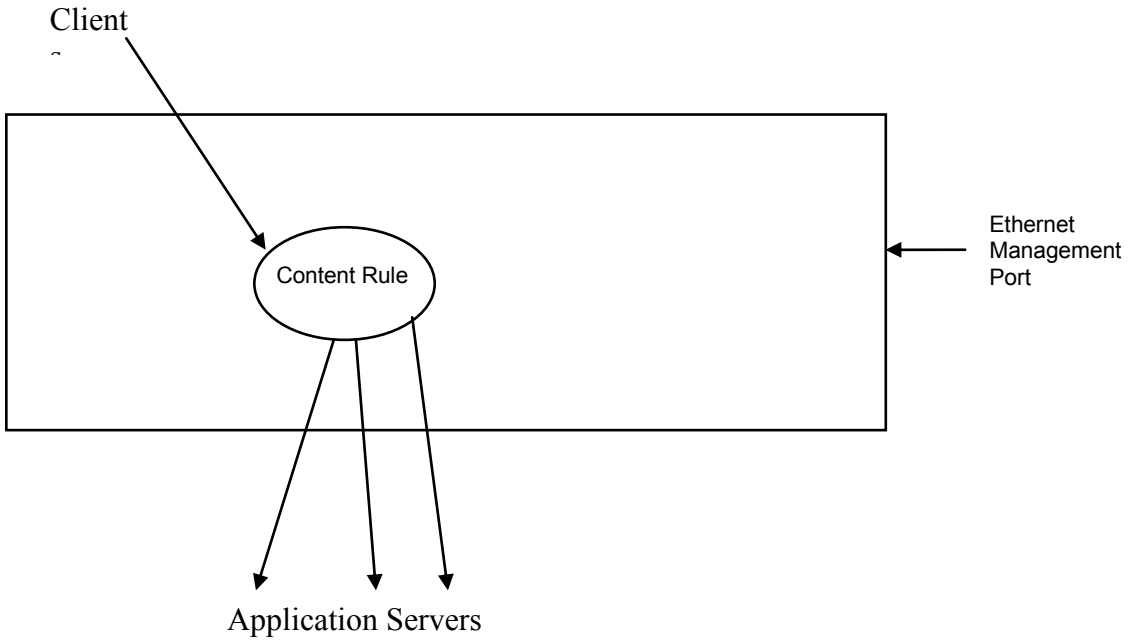

Figure 3: Cisco Content Services Switch

Each content rule has its own virtual IP address (VIP) that is used to indicate which content rule the request is destined for. When a request arrives at the switch the IP address that the request is destined for is analyzed. If the address matches the VIP of one of a content rule on the CSS, then the request is forwarded to which ever application server the content rule chooses. If the address in the request is one of the application servers behind the CSS, that request gets forwarded directly on to the server it was destined for. A client making a request to the application being load balanced by a particular content rule sends the request to the VIP of the content rule, not the switch

itself. Each content rule resides on a virtual local area network (VLAN), which is comprised of the physical Ethernet ports on the switch. This setup allows the CSS to be configured to divide the physical Ethernet ports up into multiple VLANs.

4.2.1 Problems Encountered

Each VIP is virtual in nature, allowing the CSS to contain as many content rules as desired, however this required setup posed a problem for us on our network. On the WPI academic network we were utilizing it is required to register a MAC address for each IP distributed, which is a common phenomena employed at many businesses and institutions to ensure strangers are not plugging their own laptops into the network and utilizing network resources unauthorized. The VIP takes the MAC address of the VLAN that it is a part of. Because of this setup multiple IPs must be configured for the same MAC address. We were able to solve this problem by talking with our network administrators and being able to register multiple IPs with the same MAC address.

4.2.2 Accessing the CSS

There are two methods for configuring the CSS. One can use the IP of the VLAN to log in and configure the switch manually. The preferred method of access however is the Ethernet management port. This port is a separate physical port on the CSS that is not tied into any VLAN. In fact it is required to be on a different subnet. The reason behind this port is that if the network that the switch is on goes down or is tied up for some reason there is a way to access the switch. This setup also presented some problems as we did not have a second subnet on which to put this port.

Aside from direct access to the CSS via SSH or Telnet, there is another way to configure it. One can send HTTP Put requests to the IP address of the management port containing XML. This XML is then parsed by the CSS and used to configure services, content rules, etc... This process is a useful way to access the switch as it allows one to write scripts to change the configuration of the CSS.

4.2.3 Content Rules

The benefit of being able to have multiple content rules was that we were able to configure different content rules for each of the configurations studied. Each of these content rules has its own set of services registered with it, which in some cases did overlap with other content rules. This configuration meant that to test a different configuration it is simply a matter of suspending one rule and activating another. The VIP address on each content rule is the same, meaning two content rules could not be active at once, but this approach did have a benefit; the testing program did not have to change the URL at all between tests of different configurations.

4.3 *Application Servers*

Each application server can reside on the same physical or virtual machine as another application server depending on which of the configurations used that this study is looking at. The specific application server we used for this study was JBoss, which is created and maintained by a division of Red Hat [17]. This application server is built off of a Tomcat servlet container and adds support for Enterprise JavaBeans and other new J2EE technologies. JBoss uses the autodeploy feature of Tomcat to automatically deploy applications placed within the server directory of JBoss saving the user from manually

deploying each application. Multiple copies of JBoss can be run on the same physical machine and only one installation of JBoss is needed. Each server deployed on the same physical machine has a set of XML files that determine the configuration of the server. JBoss comes preconfigured for up to three servers per physical machine and this setup is what we used for the study; however it would be a relatively easy task to configure a new server if desired. Each JBoss server on the same machine is simply listening for requests on different ports. JBoss also has capabilities for clustering, both between physical machines and even on the same physical machine. Clustering was not investigated in depth for this study as the CSS took care of all the reasons that we would need clustering for this study. The CSS took care of session state management and load-balancing, so it was not necessary to add clustering between the application servers. However, the CSS does not provide a means to handle the loss of an application server, which is one of the main reasons to use clustering. As this problem was not an aspect investigated for this study, clustering was not needed.

4.3.1 Physical Machines

There were three physical machines that were used to run the instances of JBoss for this study. Each machine was identical and ran Suse Linux. The machines were all mirrored copies of each other in terms of hardware and software installed. Each machine was directly hooked up to the CSS and was configured to run up to three instances of JBoss at a time. Each machine had 2 GB of RAM and a single 1.86 GH Intel dual core processor.

4.3.2 Virtual Machines

Each physical machine also could run a number of virtual machines that had the same software installed as the physical machine. Each virtual machine ran Suse Linux with JBoss on top of it. Each virtual machine also ran on top of Suse Linux. The specific virtual machine we used was VMWare. The reason we chose VMWare over competing virtualization software such as Xen was because VMWare is a popularly used piece of software and also because the licenses had already been procured so it was much easier to set up.

4.3.3 Configurations

The three configurations discussed earlier were the three types we implemented in this system. Since JBoss was pre-configured to have up to three instances running at one time, we used this configuration as a limiting factor to keep load distribution fair. This setup meant that there were at most only three services registered with our TPC-W content rule on the CSS at one time. This setup means that for any given test there was always a maximum of three utilized application servers, even if that meant one per physical machine, in this way the configurations could be compared more fairly.

4.4 *Database*

The database is the final layer in the network architecture described in Section 2.5. It is used as long-term data storage and data retrieval for the web applications running on the application servers. For this study Oracle 10g was used. TPC-W came ready to use with MySQL, however it was decided that Oracle would be more robust and is also a commonly used database for businesses and institutions. TPC-W did require some work

to convert the SQL statements to Oracle, however if it is desired to change the database again, we have pulled all the statements out of the TPC-W code and put them in properties files, so it should be much easier to change in the future.

4.4.1 Modified Database Architecture

It is at this point that there is a slight variation of the architecture described thus far. Figure 1 shows the database directly connected to the application servers, which is good to understand conceptually what components talk to which others; however, often the database is hooked directly to the CSS.

There are a couple of good reasons for setting the network up this way. First of all depending on how many application servers one has it may not be feasible to connect every single one to the database server. The other reason is that the application deployed on the application servers could well have to change slightly to modify which port on the database server its communication is going to if it is hooked up directly, meaning that one would have to recompile the application for each physical machine one would want to deploy it on. The concern would again be that there would be a bottleneck at this point, however, almost all of the processing is done either on the application server or on the database server and it uses few resources on the CSS to forward communication from the application server to the database. The CSS knows all the IPs for the machines that are connected to it. So that communication bound for a machine behind the CSS is not processed at all but immediately forwarded to the desired machine. It is this architecture that we used for our study.

4.4.2 Database Population

TPC-W is also bundled with a database generation tool. This tool populates the database with a specified number of items and information for a specified number of emulated browsers. The information for each emulated browser is related to the information stored for a particular user that would be visiting the site. This configuration means that at any given time, no matter how many instances of the TPC-W client utility one has running, the maximum number of emulated browsers cannot exceed the maximum stored in the database, as this would indicate that a user was logged in on multiple machines and is not allowed.

4.5 *Summary*

This chapter discussed the testing environment in which we conducted the tests for this study. It looked at all the aspects of the typical network architecture discussed in Chapter 2. The client layer uses the client application of TPC-W, which generates a number of emulate browsers that each run on a separate thread, who then make requests to the application swithch. The application switch is a Cisco Content Services Switch. When a request comes into the switch it is then matched to a content rule and routed to the appropriate application server based on the load-balancing rules defined in the content rule. The application servers were JBoss application servers, these servers each host a copy of the TPC-W server application. Requests are served by JBoss by generating HTML through servlet technology while communicating with an Oracle database. Each JBoss application server may be running either on a physical machine on top of Linux, or on Linux in a virtual machine, which is running on top of Linux.

5 Methodology

This chapter describes the methodology that was utilized to answer the questions posed in Chapter 3 using the testing environment described in Chapter 4.

5.1 *Agility*

In order to measure agility we need to see how quickly the system can respond to increasing load. The best way to measure the time of the response is to measure the time it takes to start up a new server. To this end we wrote three scripts: one that starts up an application server on an already running physical machine (Appendix A.3), one that boots a virtual machine from scratch and then starts an application server (Appendix A.1) and one that brings a virtual machine with an application server that is already running on a suspended virtual machine, out of its suspended state (Appendix A.2). The first script tests the phys configuration, the second tests the diffphys configuration and the third tests the virt configuration. Each of these scripts, after starting their application server, sent XML to the switch to configure the service and add it to a content rule. An example of this XML used for registering a new application server on a physical machine is shown in Appendix A.6. Finally each of the three scripts calls another script (Appendices A.4 and A.5) that polls the application with HTTP get requests until it is able to get a response from the application, thus indicating it is online and ready. Each of the three scripts records the start and end time of its runtime. This is used to find the time it takes to have an application server ready to go and is averaged over 10 tests.

5.2 *Performance*

Performance was measured on two fronts, the performance perceived by the user and the actual performance of the application server. These two can be closely linked, but the user may not always notice as much of a difference as would be seen from the application server's view. Also if it much easier to understand the change in response time if there is server data to back it up.

As stated earlier, TPC-W collects data about the round trip response time of the application from the client utility. This data gives a good indication of how responsive the user feels the application is and is the benchmark used to analyze performance on the client end.

In order to measure the performance of the application server we look at two statistics; free memory and CPU utilization. As load on the application servers increases and hence load on the physical machine increases, the CPU utilization should go up and the amount of free memory should go down. This change provides an indication of how much of the resources is being used by the application server. All the tests for the configurations were run on the same machine for consistency, except of the diffphys configuration which of course needed all of the servers to test. In order to be able to compare the results for the server performance with the other two configurations, the diffphys configuration measured the server statistics on the same machine that the other two configurations measured. This could have skewed results slightly if the load balancing was not evenly distributed between the three servers, however since we were using a round-robin load-balancing method, this should not have been a significant problem. To measure server performance we used the vmstat utility.

5.2.1 vmstat

Vmstat is a command line utility that comes as part of Linux that measures system statistics and displays them. It allows one to see the current CPU, disk, memory and other important statistics about the current status of one's computer. This data can be printed once or an ongoing display can be used to continuously update the data at a current rate. Each time the data is printed, it is printed as an average since the last vmstat call that was made. Therefore, the first time one calls vmstat, the average statistics since one turned the computer on are displayed. This utility is useful to gather data about the machine one is running it on.

We measured the current conditions of the server every 5 seconds of the 45 minute test run. When recording output every 5 seconds, vmstat takes the average of the statistics over the last 5 seconds and displays that average. We piped this output to a file and analyze the data from there. By logging every 5 seconds, there are two advantages, first, it eliminates outliers and shows the general trend of the statistics over the course of the run, and second it keeps the data log to a reasonable length.

5.3 *Resource Isolation*

As stated before, resource isolation is difficult to measure. In order to determine how load affects coexisting application servers, a utility that would load up one of the application servers was needed and an application called Httperf was chosen.

5.3.1 Httperf

Httperf is a tool used to generate HTTP requests and can be used for stress testing of a server. It can perform single HTTP get requests but it is utilized more for creating a

large load on a server and determining how well the server can handle it. Httperf records basic statistics such as (# requests) / (# responses).

To measure resource isolation we ran two separate application servers. The first server was used to handle the normal TPC-W client requests. For the second we used Httperf to bombard the server with requests for the TPC-W homepage. We ran this test twice per configuration. The first with Httperf running and the second time without it running. These tests gave a performance difference between the cases where another server was overloaded and when it was not, resulting in a demonstration of which configuration is least affected by performance hits from another server.

5.3.2 Resource Isolation of Suspended Machines

There was one other test that was looking at resource isolation but in a slightly different case. This case was looking at the performance difference when there were a number of virtual machines in a suspended state in the background. This case is important because it needs to be certain that holding virtual machines suspended and ready to go does not create an overshadowing performance hit to the responsiveness one would get by using such a setup.

5.4 *Summary*

This chapter discussed the methodology that was used to quantify the three aspects of agility, performance and resource isolation. To measure agility, scripts were used to start an application server and receive a response back, recording the time between the two events. Performance was measured on both the server and client side, using vmstat to measure server performance and TPC-W client statistics to measure user

perceived performance. We used CPU and amount of free memory as statistics that measure the performance of the server and the response time of the server as a measure of the user's perception of performance. In order to measure resource isolation we used Httperf to measure the difference between two servers, in the first case one being overloaded by Httperf and in the other case, neither was overloaded by Httperf. This difference gave us an understanding of how the resource usage of one application server impacted the others for each configuration.

6 Analysis

This chapter describes the analysis of the data collected on each axis of Figure 2: performance, agility and resource isolation. Once the data was gathered we analyzed it and broke it up into graphs. For user perception of responsiveness we took the output of TPC-W and ran it through a script to summarize the results (Appendix B.1), after which we converted it into a CDF using another script (Appendix B.2) giving us a graph such as Figure 4. A CDF graph is much easier to view this sort of data; the amount of data is so large and varies so much that it would be difficult to understand the results if they are not sorted as such. In order to graph the performance statistics of the server, we took the output from vmstat and ran two scripts on it (Appendices B.5 and B.6) to split the data into two files, one for CPU utilization and one for free memory. Free memory was graphed over time as there was little fluctuation, allowing one to easily see the results of this metric as in Figure 6. CPU utilization however fluctuated a large amount, so it was necessary to also convert this data to CDF format after which we got a graph such as Figure 5. It is using these analysis methods that we were able to graph the performance and resource isolation of the different configurations.

6.1 *Performance*

Figure 4 shows the user perception of the performance of TPC-W on each of the three configurations. As one can see there is almost no difference between any of the three configurations. This lack of difference seemed odd to us as there is clearly a difference between the amount of resources consumed between configurations as shown in Figure 5 and Figure 6. We noticed that the servers were not making full use of their

capabilities to serve requests to the application, indicating that the servers are not under a significant amount of load and hence will show no performance differences to the user as all are serving requests at the fastest speed possible.

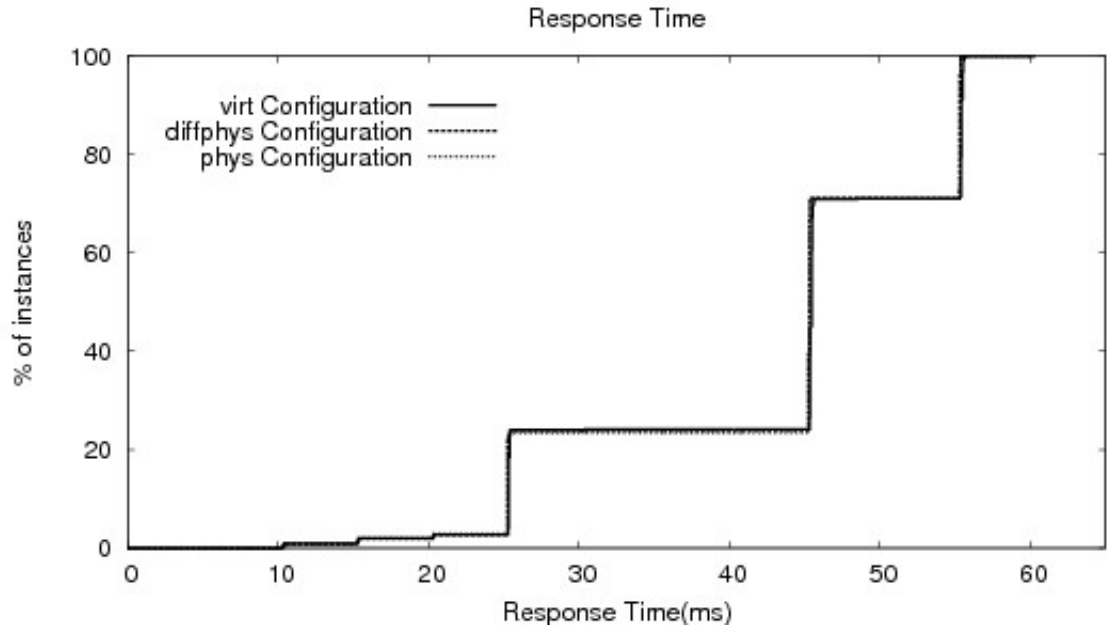

Figure 4: Response time for three configuration comparison

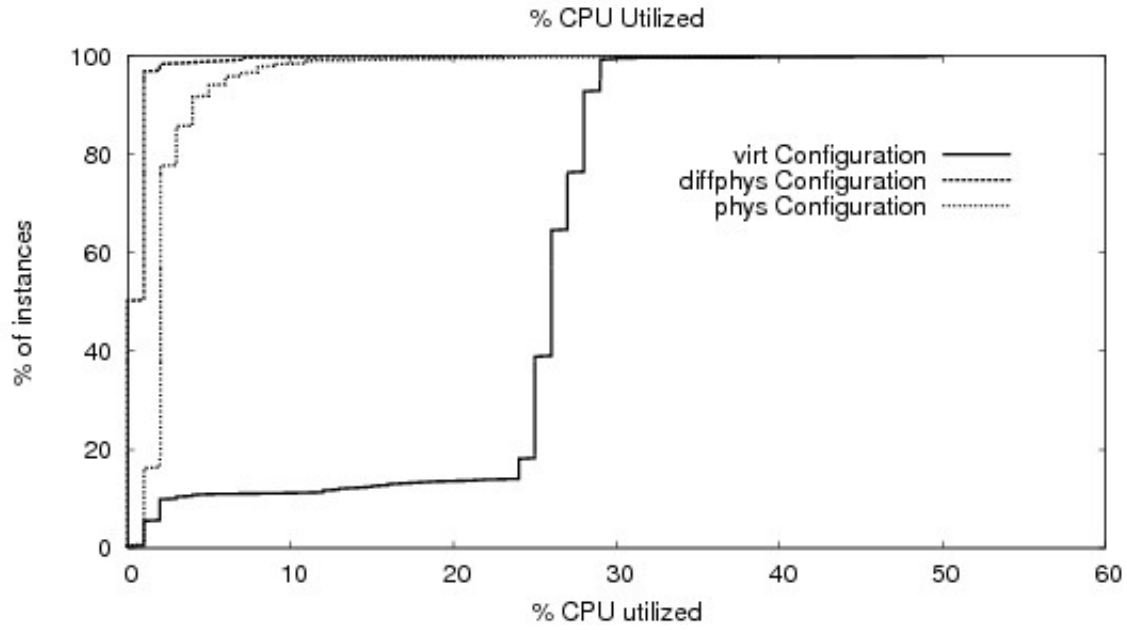

Figure 5: CPU utilization for three configuration comparison

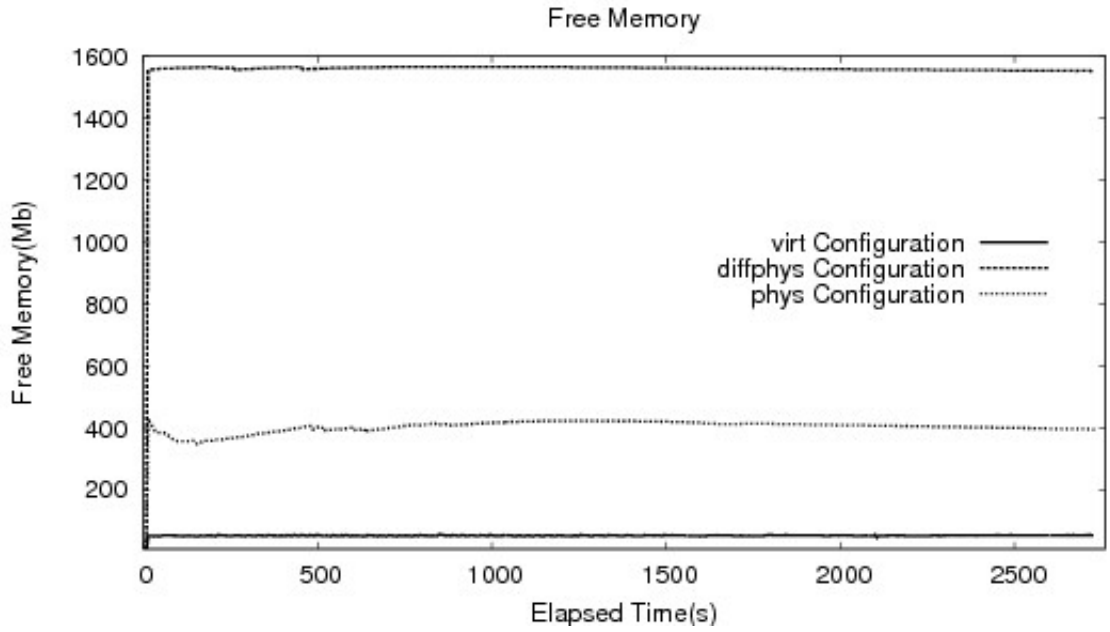

Figure 6: Free memory for three configuration comparison

6.1.1 Increasing Load

Clearly one of the ways to increase load on the server is to have more clients making requests to the application. The most logical conclusion then is that we need to run more TPC-W emulated browsers. The different client sessions should be run from separate machines so that a bottleneck does not form, however this method caused problems for us. The database as mentioned earlier was populated for 500 emulated browsers. Each TPC-W client application running uses the id of one of these browsers to do various session specific actions, such as logging in and shopping. The expectation was that we could run multiple instances of the TPC-W client so that instead of 500 emulated browsers we would have 1000 or 1500 browsers. When we attempted this approach however, TPC-W would have errors when an emulated browser logged in through one client application would also try to log in from another client application. The use of the same user id in two different emulated browsers would cause TPC-W to

have an error as the same person was trying to log in from two locations. An alternative to this option is to populate the database for a larger amount of emulated browsers and run a single client application with more browsers. This solution was not considered a desirable one however as having so many threads running on one physical machine could cause a bottleneck and skew the results that we received. A third option is to combine the first two possibilities described by having multiple TPC-W client sessions on different physical machines with a larger emulated browser database where each client session uses a range of browser ids. This option also was not viable for us as it requires significant modification of the TPC-W code, which we did not view as a good option.

Instead of these three options we decided to utilize the Httperf utility that we were already using to measure resource isolation. Unlike TPC-W, Httperf does not have delay between requests built in, meaning that many more requests can be generated. Also, no analysis of the response is done, meaning that there is less processing time lost on the client side and hence more time spent generating the next request. By putting Httperf sessions on multiple physical machines, we were able to send requests to the application servers at a far increased rate. We sent requests from the Httperf sessions to home page of the TPC-W application. This use of Httperf meant that there was database access and that the page was still generated each time. We were then able to again run the TPC-W client application and be making requests against a server that was under a great deal more load. This approach caused us to be able to see a more realistic difference in performance between configurations.

6.1.2 Httperf vs No Httperf

It is important to see the difference between making requests to a server that is under load by Httperf and one that is not so that we can justify this methodology and show that it does appropriately load up the servers. We tested each configuration with 0, 1, 2, 3 and 4 instances of Httperf running against the application server. Shown in Figure 7, Figure 8 and Figure 9 is a comparison of these 5 tests for the diffphys configuration.

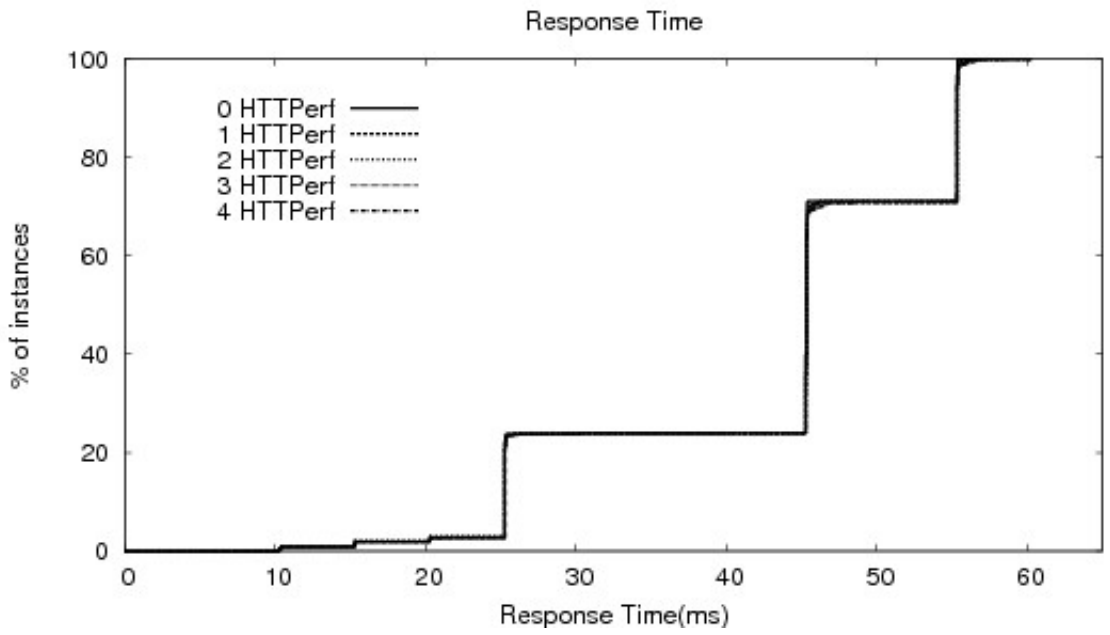

Figure 7: Response Time Httperf Comparison for diffphys Configuration

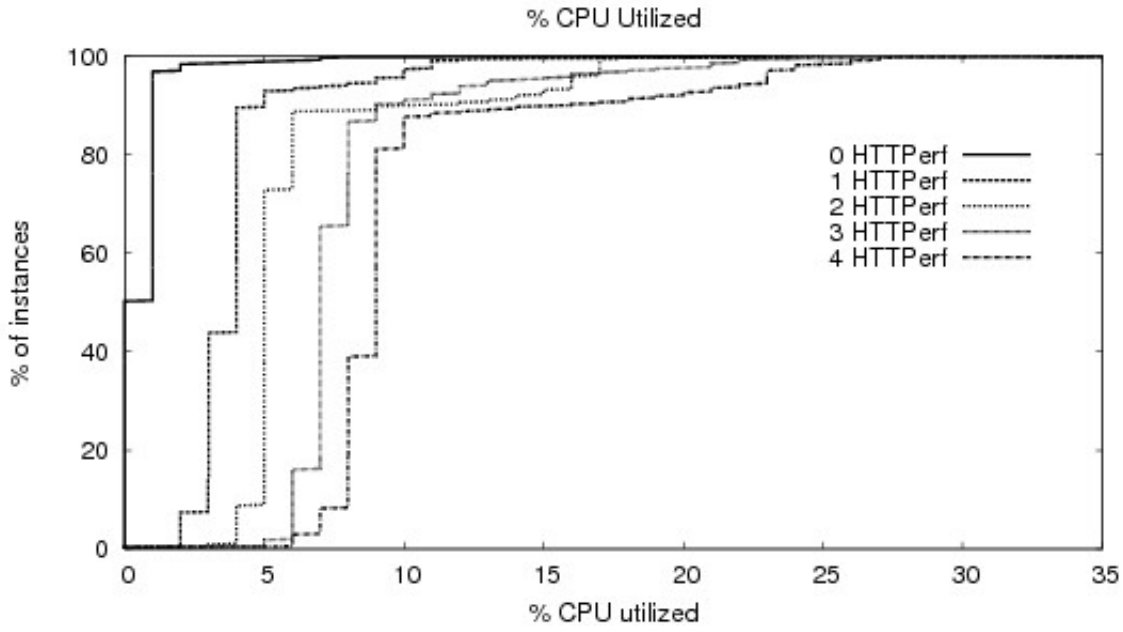

Figure 8: CPU Utilization Httperf Comparison for diffphys Configuration

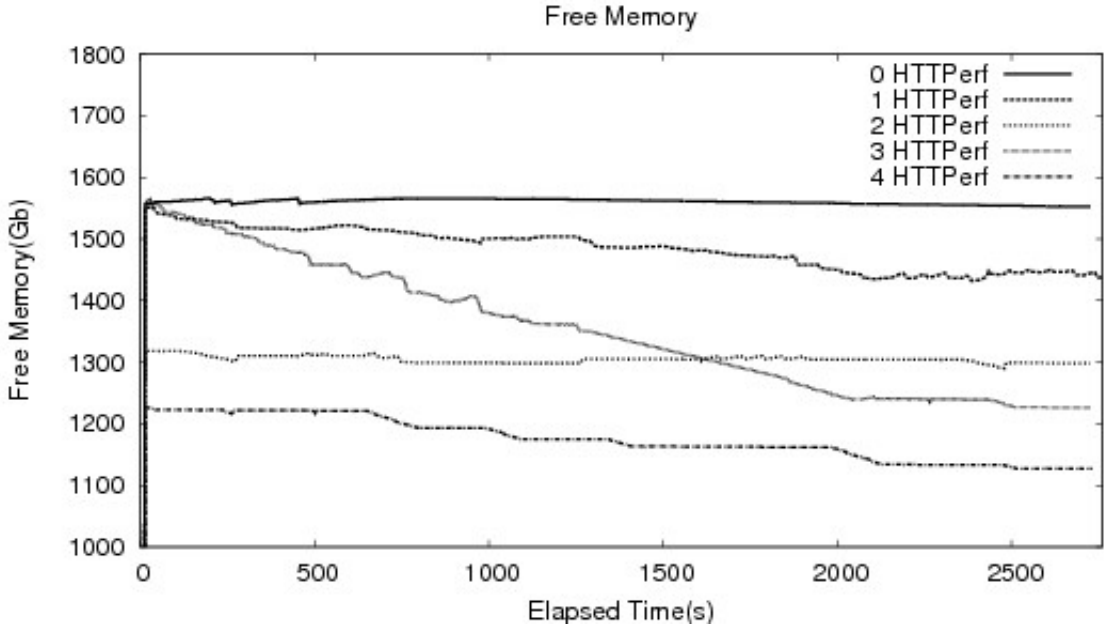

Figure 9: Free Memory Httperf Comparison for diffphys Configuration

Clearly for every instance of Httperf there is an increase in the load on the server. The amount of free memory steadily decreases and the CPU utilization increases. However we were still not able to see a significant difference in what the user perceives as the performance of the server.

6.1.3 Configurations Under Load

Figure 10, Figure 11 and Figure 12 show the results from running TPC-W with 3 instances of Httperf also running against the same server. There was not a significant change in the user perception of the performance of the server, however we did see an increase in the CPU utilization and a decrease in free memory which is what was expected.

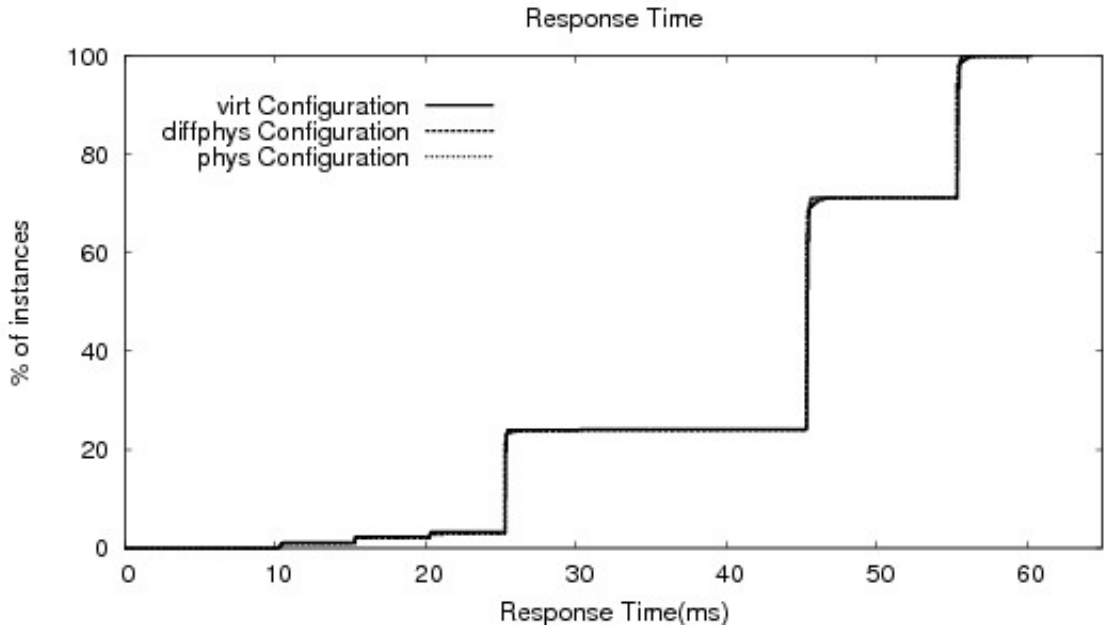

Figure 10: Response Time for Three Configuration Comparison Using Httperf

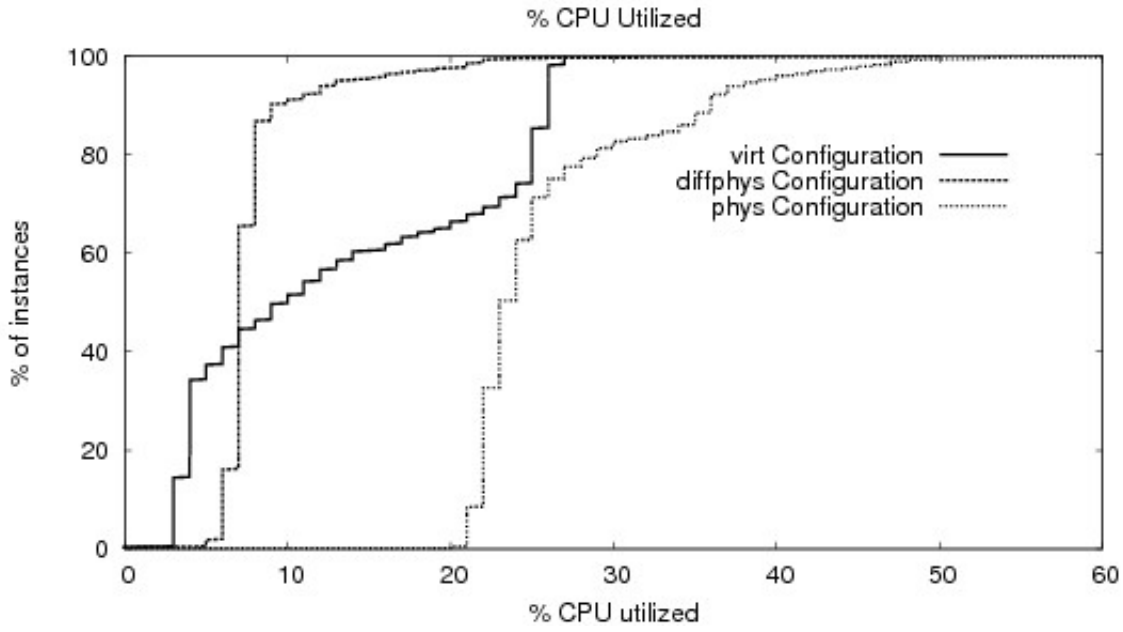

Figure 11: CPU Utilization for Three Configuration Comparison Using Httperf

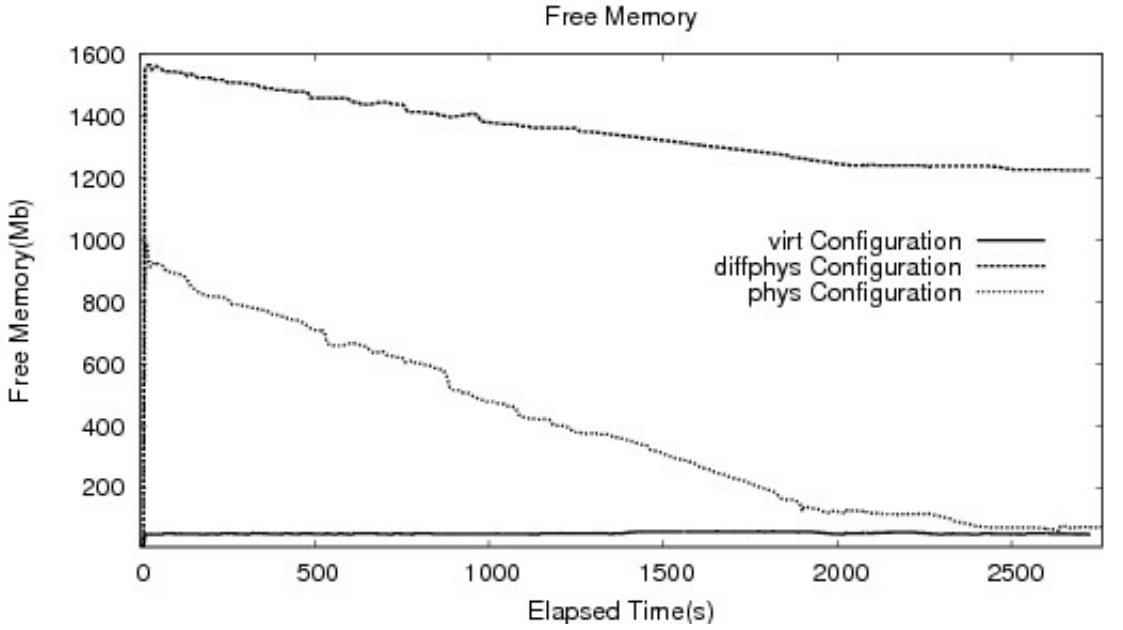

Figure 12: Free Memory for Three Configuration Comparison Using Httperf

There are two oddities in these results. The amount of free memory for the virt configuration did not seem to change. This lack of change is because the amount of memory for a virtual machine is bounded, in our case at 256 Mb, meaning that unlike an

application server running on the physical machine which can keep grabbing more and more memory as it is needed, a virtual machine can only grab as much memory as it wants up to 256 Mb. So unlike the other configurations shown in Figure 12, the virt configuration stays constant in its memory usage because it cannot take any more. The second oddity is that in Figure 11, the virt configuration uses less CPU cycles than the phys configuration, when it should actually be the opposite.

6.2 *Agility*

The agility test changed slightly when it was actually run. It proved difficult to write a script that booted up a machine entirely from scratch. It was however much easier to start a virtual machine from scratch. This process should be comparable for the following reasons:

- The machine being tested had no other applications or servers running on it, so there was no load at all to slow down the virtual machine.
- The virtual machine is a mirror image of the physical machine in terms of software and operating system, thus it goes through the same process to start up.

Starting an application server on a physical machine is an indication of the agility of the phys configuration, starting a virtual machine from a suspended state is an indication of the agility of the virt configuration and booting a virtual machine from scratch is an indication of the diffphys configuration. The results we got for the tests were on the order of what we expected (Table 1). We were not able to get an exact number on booting the machine from scratch test as in the script we had to wait until the machine was fully booted before we could tell it to start the application server, otherwise the SSH request would fail. We got it as close as possible and believe to have a margin

of error no more than a couple of seconds. Even with that margin, the difference in startup time from the other two is so great that it is clearly not favorable.

Table 1: Agility Results

Test Configuration	Time (Seconds)
Start Application Server	11.8
Start Virtual Machine (suspended)	3.1
Start Virtual Machine (boot)	~ 45

The time to come out of a suspended state and be ready to receive requests was impressive; it took virtually no time at all and was a clear winner amongst the three. As long as the virtual machines in a suspended state do not pose a problem in terms of performance, having a couple suspended and ready to go is clearly an optimal choice. This result may be skewed as the virtual machine was suspended and restarted without much time in between and it may have not been given enough time to wipe the memory associated with it. This was also the case for the shutdown and startup of the application server, however, so even though the results may be reduced to shorter times than they otherwise might be, the difference between the configurations should still be very similar.

6.3 *Resource Isolation*

To quantify the last of the three axes we ran tests on each configuration to determine how large of an impact resource isolation had on the servers within the configuration. For each configuration tested, we ran two application servers, one handled TPC-W requests from a single TPC-W client, and the other handled Httperf requests from

a single Httperf client. Each of these tests was split into two, one test that used Httperf and the other that did not. This split gave us two test results which allowed us to compare the difference between the times when the application server that was servicing Httperf was overloaded and when it was not to determine how it impacted the application server that was servicing TPC-W emulated browsers.

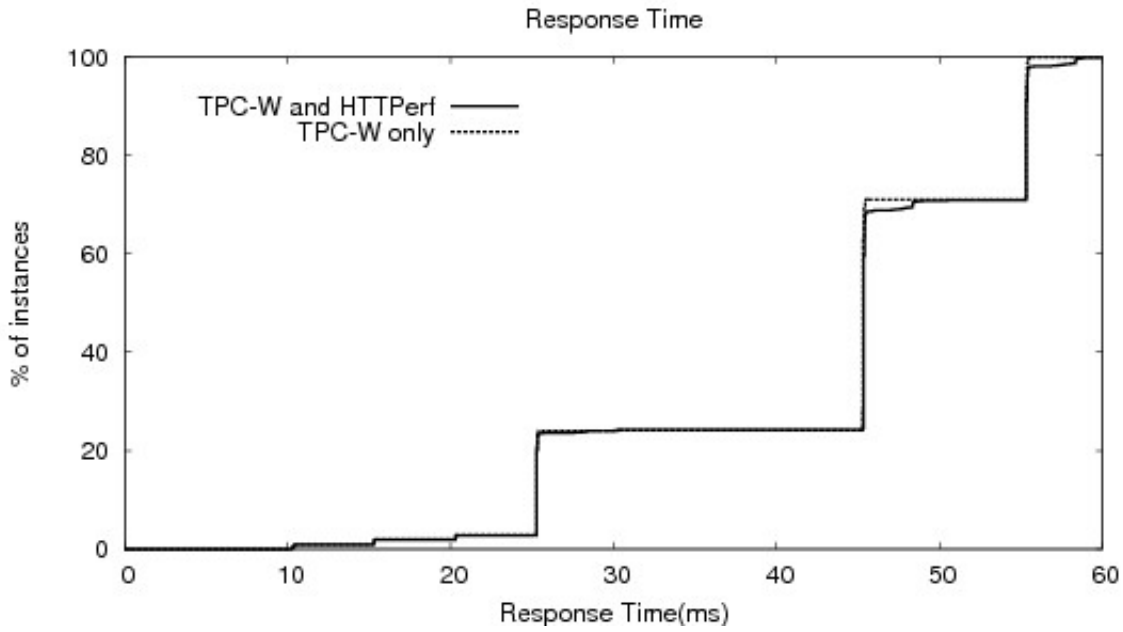

Figure 13: phys Configuration Resource Isolation Comparison (Response Time)

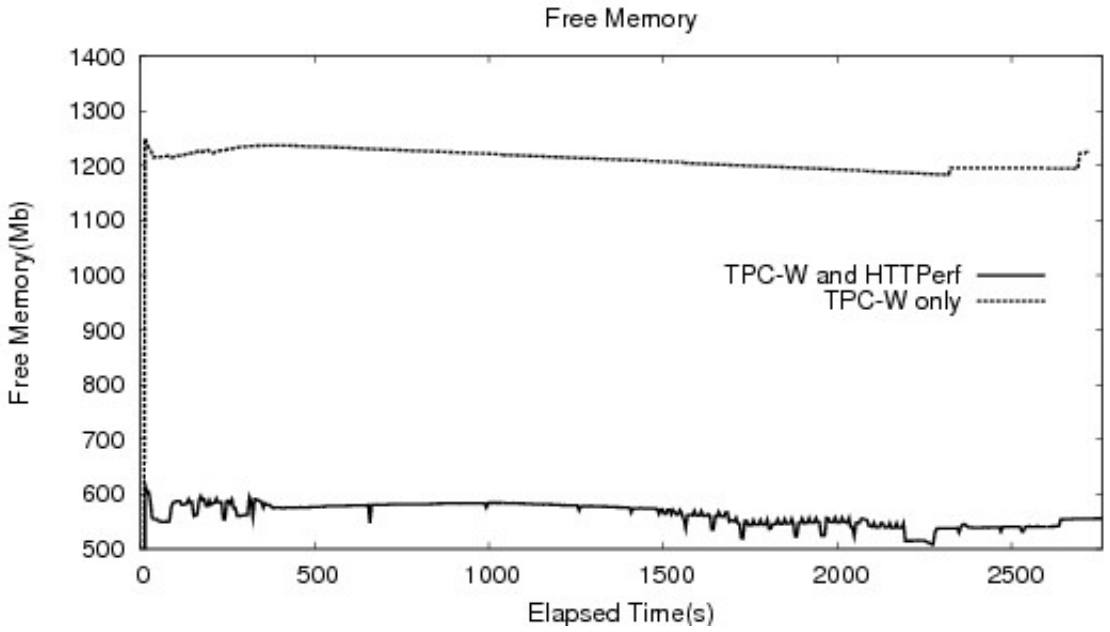

Figure 14: phys Configuration Resource Isolation Comparison (Memory)

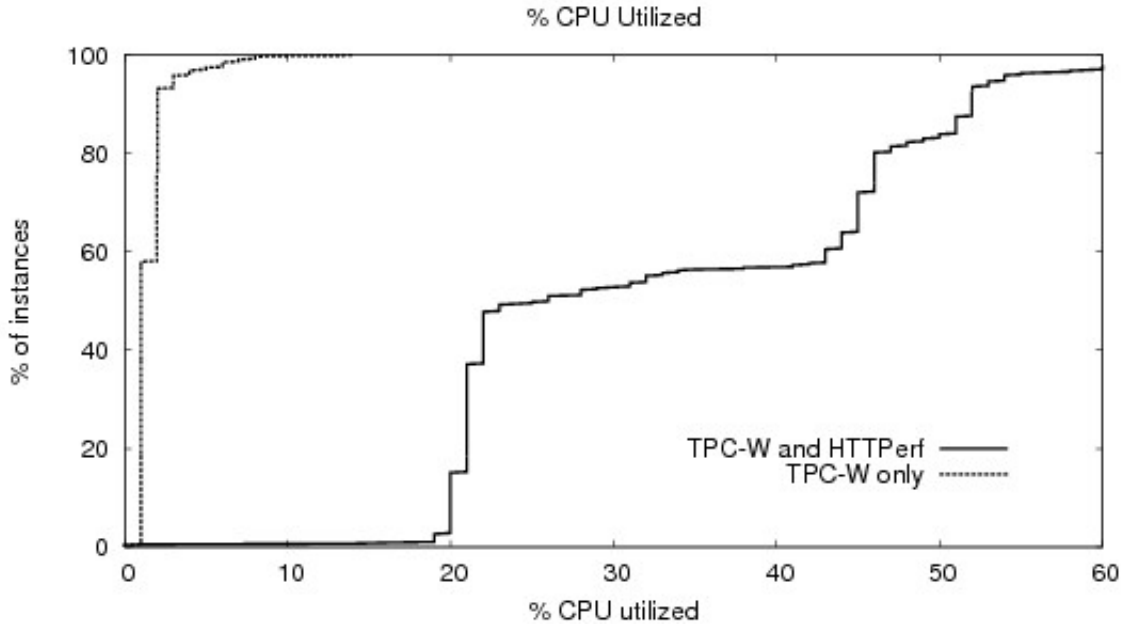

Figure 15: phys Configuration Resource Isolation Comparison (CPU)

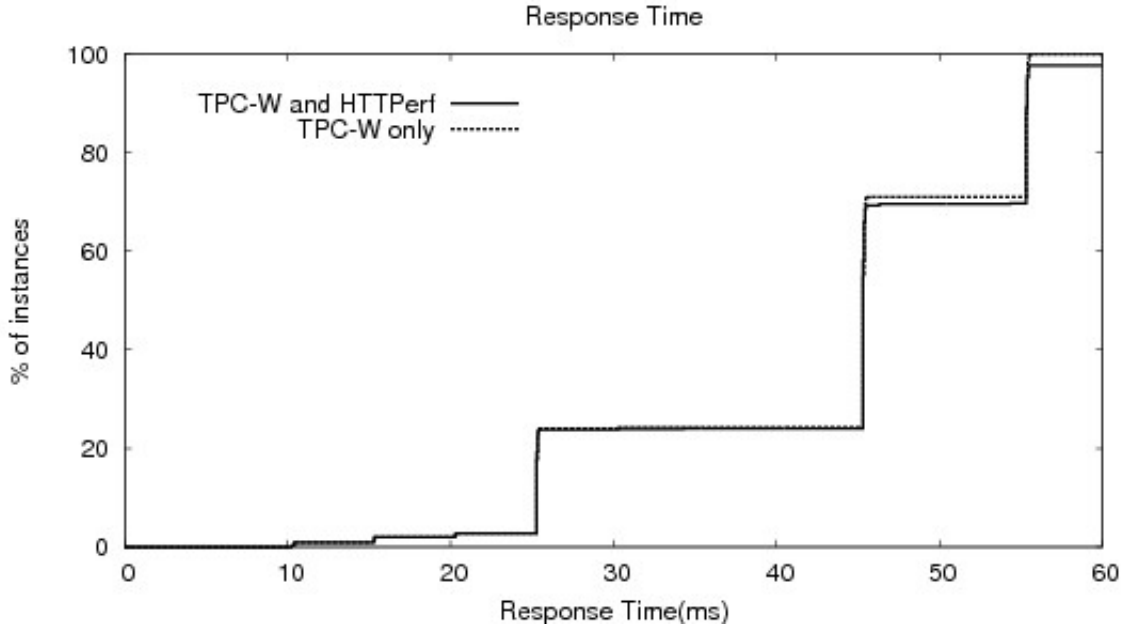

Figure 16: virt Configuration Resource Isolation Comparison (Response Time)

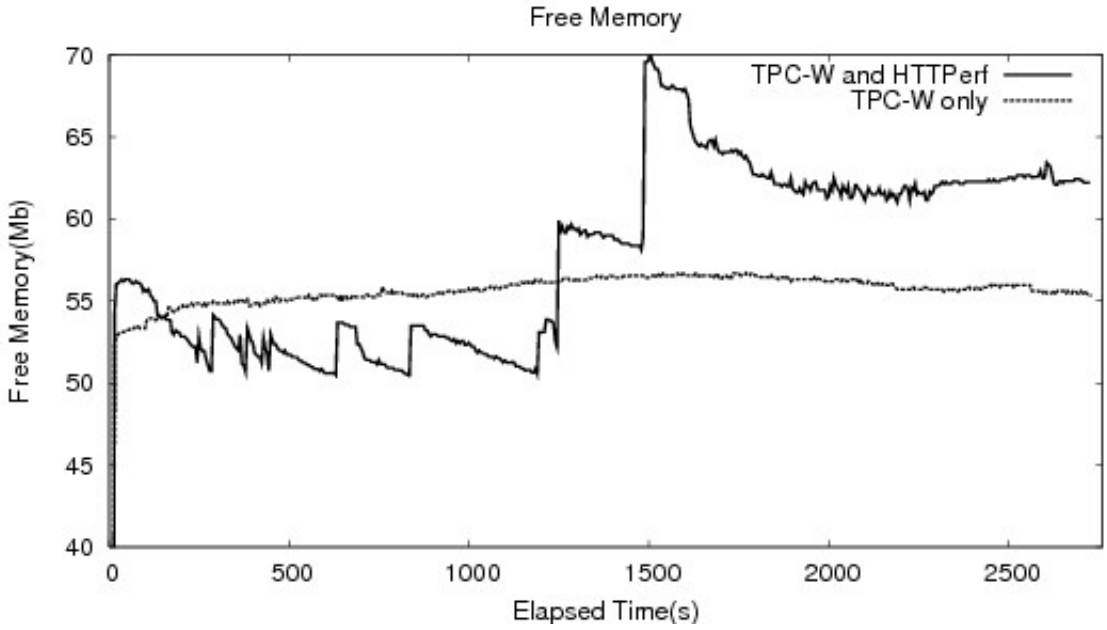

Figure 17: virt Configuration Resource Isolation Comparison (Memory)

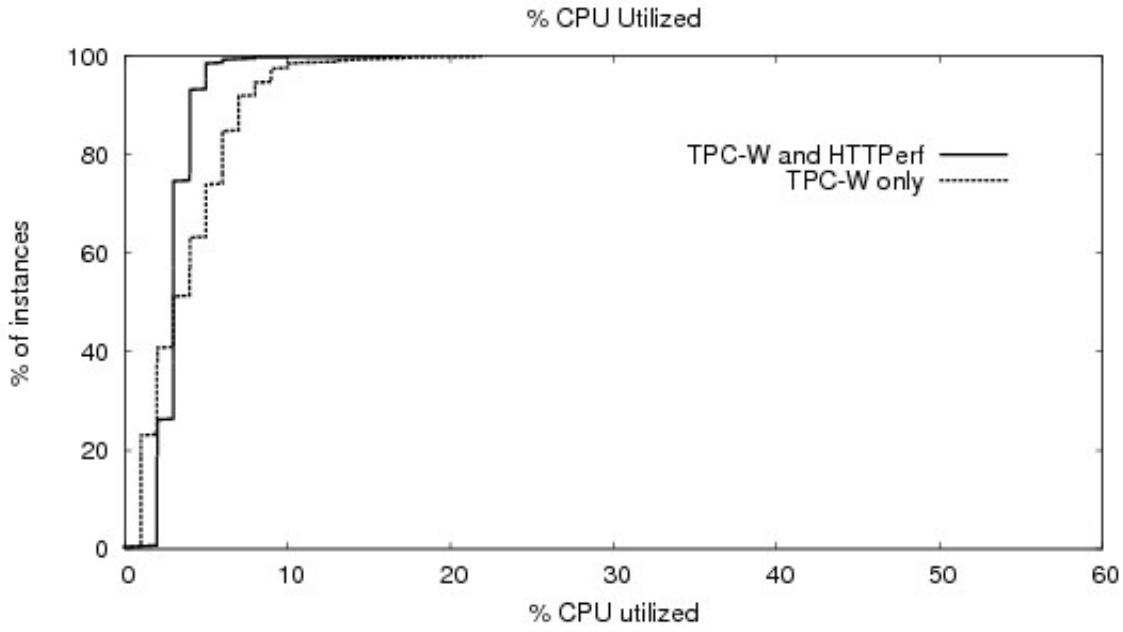

Figure 18: virt Configuration Resource Isolation Comparison (CPU)

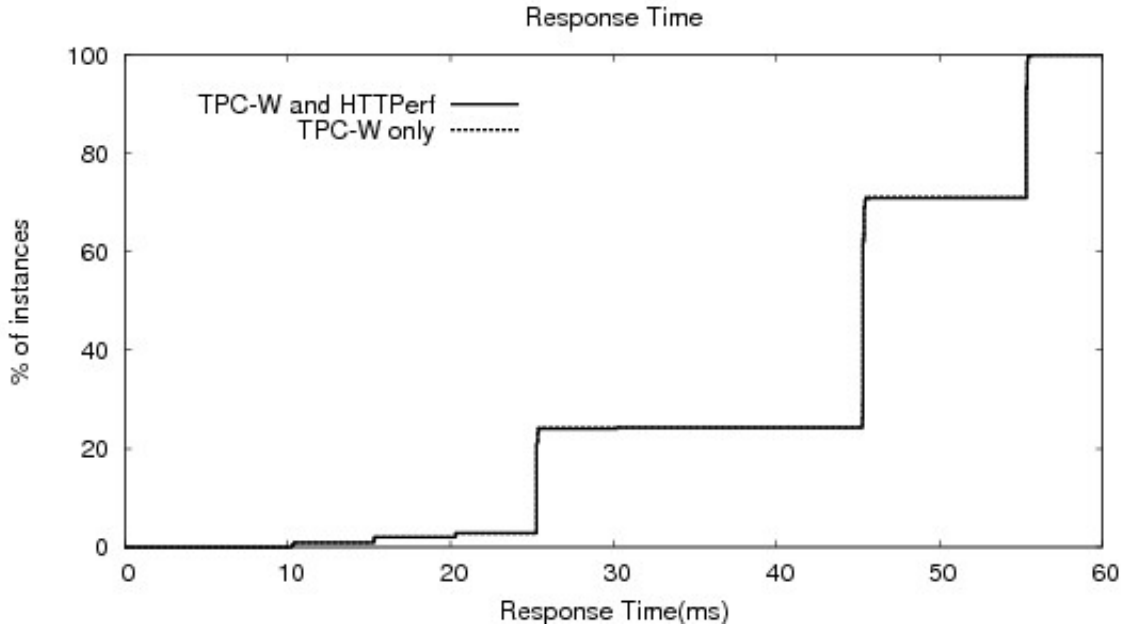

Figure 19: diffphys Configuration Resource Isolation Comparison (Response Time)

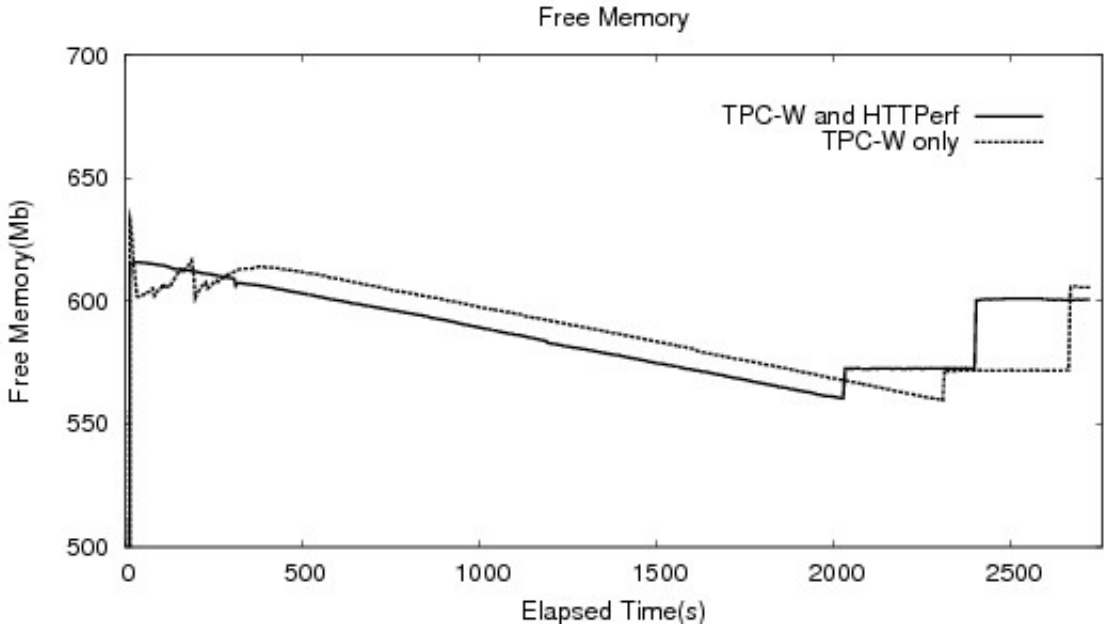

Figure 20: diffphys Configuration Resource Isolation Comparison (Memory)

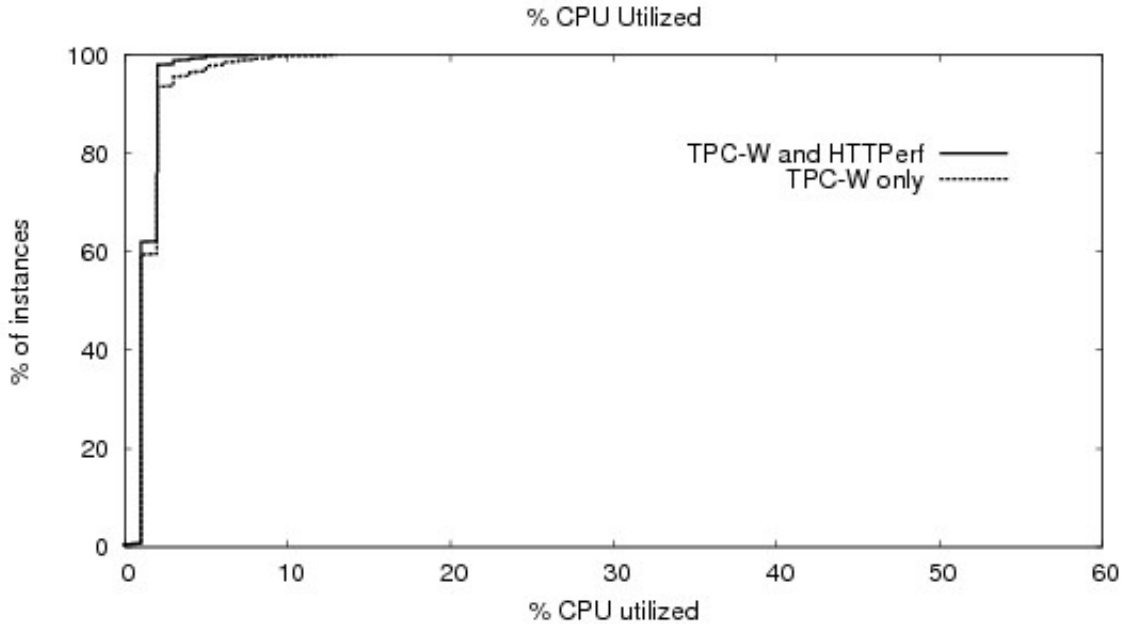

Figure 21: diffphys Configuration Resource Isolation Comparison (CPU)

As one can see increased load on an application server in the phys configuration caused a significant increase in load on the server side (Figure 13 and Figure 14) and a slightly noticeable increase in user perceived load (Figure 15). Memory especially was

heavily utilized. For the virt configuration there was a smaller difference between perceived load of the two tests (Figure 16), and the server load was also less (Figure 17 and Figure 18). For configuration diffphys (single application server per physical machine) there was a negligible difference between the two tests (Figure 19, Figure 20 and Figure 21). This difference was as expected as the resources are completely isolated from each other. In Table 2 we have displayed the mean and median response times for each of the configurations using data from the resource isolation tests for each configuration. This table allows us to confirm what we see on the graphs, that there is no difference in response time for the diffphys configuration, slightly more difference for the phys configuration and even more for the virt configuration, though all differences are miniscule. We believe this to be because we were not able to load the server up enough to get a large difference in response time, even though there was a difference in resource usage on the server side.

Table 2: Mean/Median Response Time Comparison

Configuration	Median (Httperf/No Httperf) (milliseconds)	Mean (Httperf/No Httperf) (milliseconds)
Virt	43.5/45.3	42.9/42.9
Phys	45.4/45.3	43.0/42.8
Diffphys	45.4/45.4	42.8/42.8

Lastly we also looked at resource isolation a little differently in attempting to determine how much of an impact many suspended machines in the background have on the application server. The results were all encouraging. The response time was hardly impacted at all (Figure 22), and the amount of free memory did not change dramatically (Figure 23). What was a little surprising was that the amount of CPU utilized when there

was a large number of suspended machines went up more than expected. The amount of CPU utilized was still under 50% however, and the fact that it took so many suspended machines to increase the load so much is promising. In a situation where one has suspended machines ready to go in the case of a sudden increase of load, it would only be necessary to have a couple of machines suspended at a time. Having 7 suspended is overkill and the fact that having that many suspended increased the load so little means that they do not use up large amounts of resources to do so. Though the results are promising, they are still not what is expected, a suspended virtual machine should have be utilizing no CPU cycles. The fact that our results show such a significant increase in CPU utilization might be an indication that the virtual machines were not being suspended correctly.

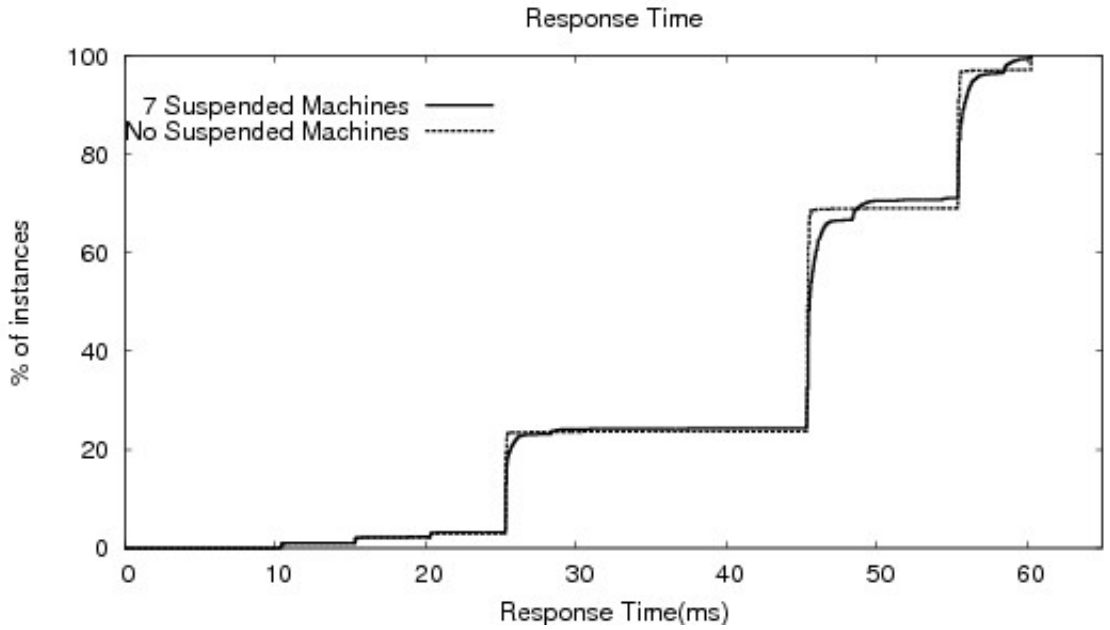

Figure 22: Suspended Vs. No Suspended Virtual Machines (Response Time)

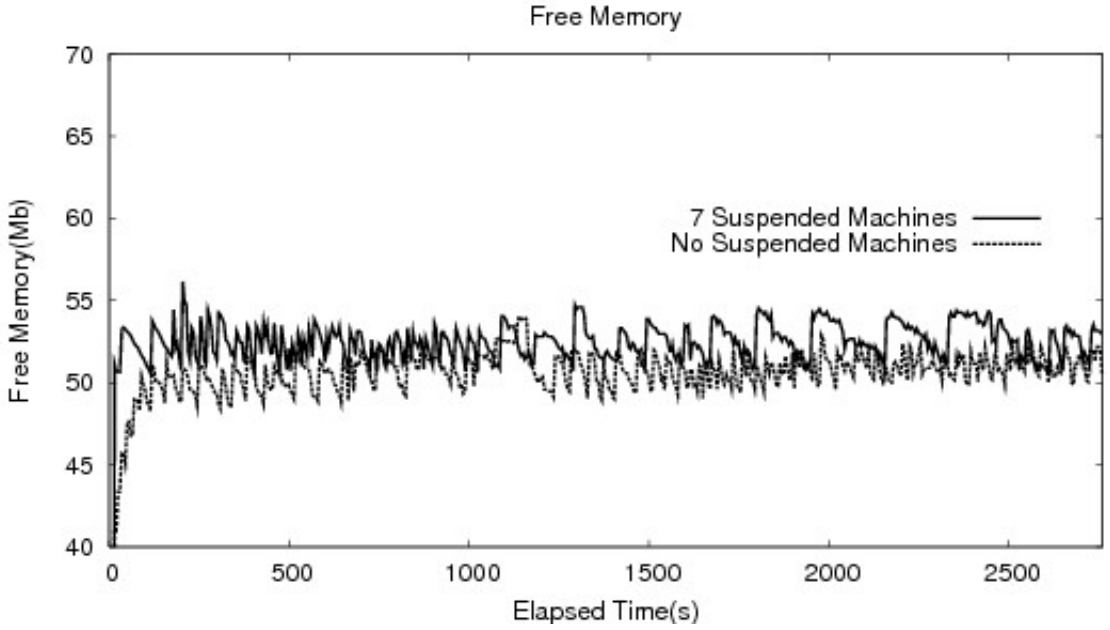

Figure 23: Suspended Vs. No Suspended Virtual Machines (Memory)

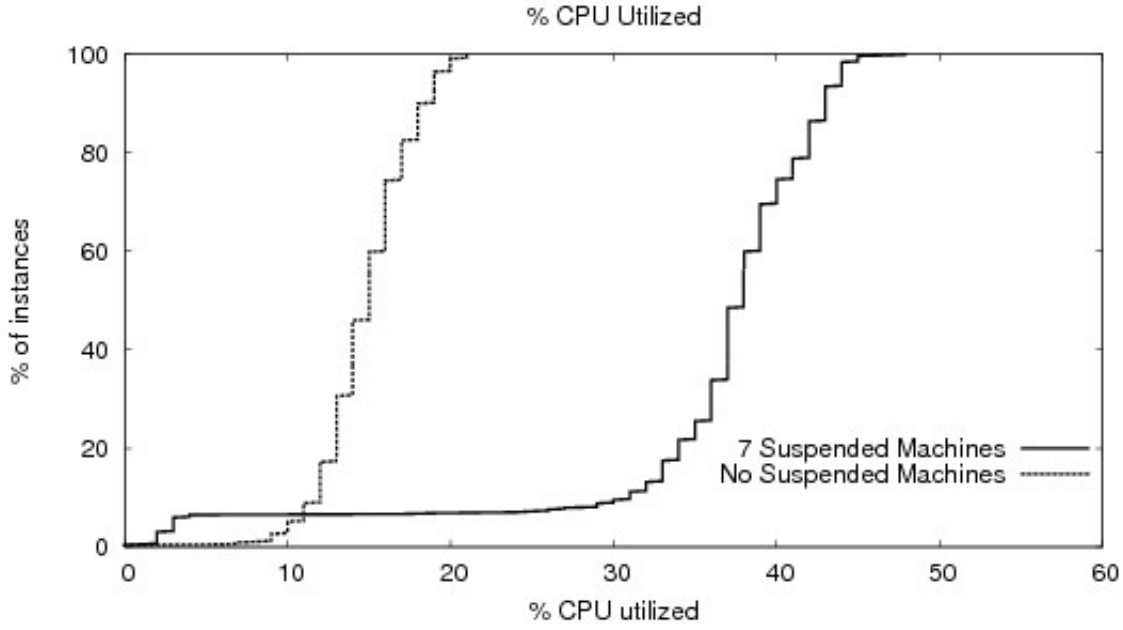

Figure 24: Suspended Vs. No Suspended Virtual Machines (CPU)

7 Conclusions

Given the results from Chapter 6, we feel it is clear to see that memory is a much more important resource than CPU is for large business web applications. We also discovered that when running virtual machines, CPU is also a necessary resource. Though we were not able to see a difference in what the user perceives as the performance of the server, we believe this lack of difference may be due to the possibility that even with 3 Httperfs running, we were not able to fully load the servers. Unfortunately we were not able to run more than three machines for Httperf until the end of the study at which point we felt for consistency's sake we needed to stick with our previous maximum of three. It would be good to try these tests with more sessions of Httperf running so that the servers are loaded up even more.

Agility needs to be looked at in terms of speed in taking an unused application server down and in speed of recovering after a crash. These were topics not looked into for this study, but that have a significant impact on the agility of a configuration. Clustering can also affect the agility of a configuration; it can recover quickly from a crash, but is far less agile in starting up and shutting down servers as well as using up resources to maintain the cluster. Clustering should be looked at and the benefits and disadvantages of this approach need to be quantified. Another path that should be explored is the ability of virtual machines to be migrated between physical machines and how that might impact the ability of network architecture to be agile.

The goal of this thesis is to quantify the benefits and disadvantages of virtual machines to allow an informed decision to be made of whether to use them or not. Virtual machines have been shown to pose a significant advantage to that of the other

architectures. There is a performance hit that is taken when using virtual machines; however the large amount of resource isolation gained using this configuration and the ability to react to an increase in load by starting up another application server quickly outweighs this downside. This benefit is especially true due to the results that show that having suspended machines ready for use, does not affect the resource usage of the server in a significant way. For these reasons virtual machines should be looked into as platforms for application servers.

8 References

[1] "About Virtualization" VMWare. http://www.vmware.com/virtualization/ (Accessed July 08, 2007)

[2] Andrzejak, M. Arlitt, and J. Rolia. Bounding the resource savings of utility computing models. Technical Report HPL-2002-339, HP Labs, December 2002.

[3] Apache Tomcat http://tomcat.apache.org/ (Accessed August 5, 2007)

[4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield. Xen and the Art of Virtualization. *Proceedings of the nineteenth ACM symposium on Operating systems principles*. Bolton Landing, NY, USA 2003

[5] C. Canali, M. Rabinovich, and Z. Xiao. Utility computing for internet applications. An invited chapter in Xueyan Tang et al. (Eds.). Web Content Delivery, Springer, 2005

[6] Jeffrey S. Chase, David E. Irwin, Laura E. Grit, Justin D. Moore, and Sara E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC '03),* 2003

[7] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpack, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Usenix Symposium on Networked Systems Design and Implementation,* 2005

[8] Renato J.O. Figueiredo, Peter A. Dinda, and José A.B. Fortes. A case for grid computing on virtual machines. In *ICDCS*, pages 550-559, 2003

[9] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, Amin Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. *ACM/IFIP/USENIX 7th International Middleware Conference*. Melbourne, Australia. November 2006.

[10] IBM WebSphere software http://www-306.ibm.com/software/websphere/ (Accessed August 5, 2007)

[11] Java TPC-W Implementation Distribution http://www.ece.wisc.edu/~pharm/tpcw.shtml (Accessed November 21, 2006)

[12] JBoss.com http://www.jboss.com/ (Accessed August 5, 2007)

[13] JOnAS OpenSource Java EE Application Server http://jonas.objectweb.org (Accessed August 5, 2007)

[14] Pradnya Karbhari, Michael Rabinovich, Zhen Xiao, and Fred Douglis. ACDN: a content delivery network for applications. In *Proceedings of ACM SIGMOD (project demo),* pages 619-619, June 2002

[15] R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. In *8th IFIP/IEEE International Symposium on Integrated Network Management,* pages 247-261, 2003.

[16] Michael Rabinovich, Zhen Xiao, and Amit Aggarwal. Computing on the edge: A platform for replicating Internet applications In *Proceedings of the Eighth International Workshop on Web Content Caching and Distribution*, September 2003.

[17] Red Hat http://www.redhat.com (Accessed August 5, 2007)

[18] S. Sivasubramanian, G. Pierre, and M. van Steen. Replicating web applications on-demand. In *Proceedings of the IEEE International Conference on Services Computing,* Shanghai, China, September 2004.

[19] Ananth I. Sundararaj and Peter A. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. USENIX 3rd Virtual Machine Research and Technology Symposium, San Jose, California, USA, May 2004.

[20] "TPC-W" http://www.tpc.org/tpcw/ (Accessed November 21, 2006)

[21] Transaction Processing Council. TPC Benchmark W (Web Commerce) Specification. Feb. 19th 2002.

[22] TPC-W (Web Commerce) Benchmark http://www.cs.nyu.edu/~totok/professional/software/tpcw/tpcw.html (Accessed November 21, 2006)

[23] WebLogic Application Server http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/weblogic&WT.ac=topnav_products_weblogic (Accessed August 5, 2007)

[24] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and performance in the denali isolation kernel. In *OSDI'02: Proceedings of the 5th symposium on Operating systems design and implementation,* pages 195-209, 2002.

Appendix A. Agility Scripts

A.1 Booting a virtual machine

bootVirtualMachine.sh

```
#!/bin/bash
date +%s%t%N
ssh cew-research2.cs.wpi.edu 'vmware-cmd /home/vmware/VM1/VM1.vmx start'
sleep 40
xterm -hold -e "ssh 130.215.29.102 'JAVA_ROOT=/usr/local/java;export
JAVA_ROOT;JAVA_HOME=/usr/local/java;export
JAVA_HOME;JDK_HOME=/usr/local/java;export
JDK_HOME;JRE_HOME=/usr/local/java/jre;export
JRE_HOME;JAVA_BINDIR=/usr/local/java/bin;export
JAVA_BINDIR;PATH=/usr/local/java/bin:"$PATH";/home/eamiller/jboss/bin/run.sh -c
node1'" &
#curl -T xmltest.xml -u eamiller:[3ll10t] http://130.215.29.126/
./getPageVirtual.cgi
date +%s%t%N
```

A.2 Starting a virtual machine

startupVirtualMachine.sh

```
#!/bin/bash
date +%s%t%N
xterm -hold -e "ssh cew-research2.cs.wpi.edu 'vmware-cmd
/home/vmware/VM1/VM1.vmx start'" &
#curl -T xmltest.xml -u eamiller:[3ll10t] http://130.215.29.126/
./getPageVirtual.cgi
date +%s%t%N
```

A.3 Starting a physical server

startupPhysicalServer.sh

```
#!/bin/bash
date +%s%t%N
xterm -hold -e "ssh cew-research2.cs.wpi.edu 'JAVA_ROOT=/usr/local/java;export
JAVA_ROOT;JAVA_HOME=/usr/local/java;export
JAVA_HOME;JDK_HOME=/usr/local/java;export
JDK_HOME;JRE_HOME=/usr/local/java/jre;export
JRE_HOME;JAVA_BINDIR=/usr/local/java/bin;export
JAVA_BINDIR;PATH=/usr/local/java/bin:"$PATH";/home/eamiller/jboss/bin/run.sh -c
node1'" &
curl -T xmltest.xml -u eamiller:[3ll10t] http://130.215.29.126/
./getPage.cgi
date +%s%t%N
```

A.4 Retrieving page from a virtual machine

getPageVirtual.cgi

```
#!/usr/bin/perl
use LWP::Simple;
#continuously poles until 404 error is not received, ie. when 'TPC-W' is in the title
#$url = $1
#for some reason argument passing isn't working, so hard-coded and perl script
duplicated
$url = 'http://130.215.29.102:8180/tpcw/TPCW_home_interaction';
do{
        $html = get $url;
}while ("TPC-W" ne substr($html, 21, 5))
```

A.5 Retrieving page from a physical machine

getPage.cgi

```
#!/usr/bin/perl
use LWP::Simple;
#continuously poles until 404 error is not received, ie. when 'TPC-W' is in the title
#$url = $1
#for some reason argument passing isn't working, so hard-coded and perl script
duplicated
$url = 'http://130.215.29.123:8180/tpcw/TPCW_home_interaction';
do {
        $html = get $url;
}while ("TPC-W" ne substr($html, 21, 5))
```

A.6 XML Configuration of CSS

```
<?xml version="1.0" standalone="yes"?>
<config>
  <service name="pm2-1">
      <ip_address>130.215.29.123</ip_address>
        <port>8180</port>
      <action>active</action>
   </service>
  <owner name="eamiller">
     <content name="agilitytest">
        <add_service>pm2-1</add_service>
        <action>active</action>
     </content>
   </owner>
</config>
```

Appendix B. Analysis Scripts and Commands

B.1 Summarize user perception of response time

SummarizeResults.cgi

```
#!/usr/bin/perl

$file = "./3virtNoSuspend";
open (DATA, $file);
$line = <DATA>;
while (substr($line, 0, 8) ne "dat.wirt") {
   $line = <DATA>;
}

$line = <DATA>;
while (substr($line, 0, 2) ne "];"){
   push (@values, substr($line, 8));
   $line = <DATA>;
}

while(substr($line, 0, 8) ne "dat.wirt"){
   $line = <DATA>;
}

for ($j = 0; $j <14; $j++){
   $line = <DATA>;
   $i = 0;
   $temp = $line;
   while (substr($temp, 0, 2) ne "];"){
         $temp = $line;
         $values[$i] = $values[$i] + substr($line, 8);
         $i++;
      $line = <DATA>;
   }

   while(substr($line, 0, 8) ne "dat.wirt"){
      $line = <DATA>;
   }
}

$sum = 0;
foreach $element (@values){
   $sum += $element;
}

$k = 0;
```

```
$total = 0;
foreach $element (@values){
   #add in an extra point to staircase the graph
   if ($element != 0){
         printf ("%8d %15f \n", $k*50, $total/$sum);
   }
   $total += $element;
   printf ("%8d %15f \n", $k*50, $total/$sum);
   $k++;
}

close(DATA);
```

B.2 Convert to CDF

cdf.pl

```
#!/usr/bin/perl

# Mark Claypool
# v1.1
# Last significantly modified: April 2003

# Cumulative Density Function (CDF) (and CCDF).  Input is a series of
# numbers, one per line. It can accept lines with more than one number
# in each column, but it will only process the first.  Output is of
# the form: value -> percent

require 'getopts.pl';

&Getopts('ch');
if ($opt_h) {
  &usage;
}
if ($opt_c) {
  $ccdf = 1;
} else {
  $ccdf = 0;
}
if ($#ARGV >= 0) {
  &usage;
}

# count occurrences of each number
$i = 0;
$line = <STDIN>;
while ($line) {
```

```
    $line =~ /(\d+)(.*)$/;
    @num[$i] = $line;
    $i++;
    $line = <STDIN>;
}

# print cdf
$i = 0;
while ($i <= $#num) {
#  printf("%d\n", $i);
  if ($ccdf == 1) {
    printf("%f\t%f\n", @num[$i], 1 - ($i + 1) / ($#num + 1));
  } else {
    printf("%f\t%f\n", @num[$i], ($i + 1) / ($#num + 1));
  }
  $i += 1;
}

exit;


################################################################################
##
# usage
# print a usage maessage and quit
sub usage
{
    print STDERR "cdf.pl: compute cummulative density data\n";
    print STDERR "Usage: cdf.pl <flags>, where flags are:\n";
    print STDERR "       [-c]\tcompute complimentary cdf data\n";
    print STDERR "       [-h]\tthis help message\n";
    exit;
}
```

B.3 Sample server performance data

```
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 1  0  39964 984956 142996 427632    0    0     0     1    1    2 23 21 56  0
 3  0  39964 987748 143016 427620    0    0     0   165 5800 6011 49  6 45  0
 2  0  39964 939004 143024 427632    0    0     0    15 5169 5192 61  7 33  0
 1  0  39964 914060 143024 427632    0    0     0    16 6107 6517 35  7 58  0
 0  0  39964 914400 143032 427624    0    0     0    22 6192 6801 33  7 60  0
 6  0  39964 923948 143036 427632    0    0     0    14 6145 6579 35  7 58  0
 4  0  39964 930272 143040 427632    0    0     0    16 6121 6366 34  7 60  0
 2  0  39964 925188 143044 427632    0    0     0    14 6033 6241 38  7 54  0
 4  0  39964 921344 143048 427632    0    0     0    30 6049 6204 40  7 54  0
```

B.4 Commands to analyze TPC-W output

```
./SummarizeResults.cgi > results.txt
cat results.txt | sort –n | ./cdf.pl
```

B.5 Command to pull out CPU data from server data

```
awk 'BEGIN{i=1} {printf("%d\n", $13); i=i+1}' exampleStats > exampleStatsCPU
```

B.6 Command to pull out memory data from server data

```
awk 'BEGIN{i=1} {printf("%5d %8d\n", i*5, $4); i=i+1}' exampleStats >
exampleStatsRAM
```