High-precision Web Application Monitoring

Archana Nottamkandath[2000598]

A senior thesis submitted to the faculty of
Vrije University
in partial fulfillment of the requirements for the degree of

Masters in Parallel And Distributed Computing

Corina Stratan, Advisor
Guillaume Pierre, Co-Advisor

Department of Parallel and Distributed Computing Systems

Vrije University

August 2011

ABSTRACT

High-precision Web Application Monitoring

Archana Nottamkandath[2000598]
Department of Parallel And Distributed Systems
Masters in Parallel And Distributed Computing

In the cloud environment resource provisioning is driven from very coarse grained measurements. For instance, new resources are allocated to applications when the average load on the physical node exceeds a certain threshold. With the variety of services provided by the clouds, many companies and users are migrating their web applications to the cloud environment. As Web applications face significant variations in their request load, the amount of computational resources they use has to be dynamically adjusted. But the current resource provisioning algorithms do not take into account the actual response time of the deployed applications, and thus do not work well for the web applications. Thus we have solved the need for the high-precision monitoring of the web applications, by developing non-intrusive sensor modules and integrating with Ganglia monitoring framework for gathering high-precision metrics about the web applications, which aid in their efficient resource provisioning in the cloud environment. One of the main challenging metric which more clearly reflects the performance of the web application at the server side is the Client Perceived Response Time(CPRT). The results indicate that the Client Perceived Response Time measurements gathered at the server side are very similar to the actual response time as experienced at the client. The high-precision metrics have been integrated with the Ganglia monitoring framework with less overhead.

Keywords: High-precision, Monitoring, Clouds, Web server, Client Perceived Response

ACKNOWLEDGMENTS

First and foremost, I would like to Thank The Almighty for blessing me with the opportunity to pursue my Masters at Vrije University and being able to complete successfully. The support, prayers and love handed in bountiful by my parents and my relatives, made every hurdle vanish and helped me successfully achieve my dreams. My heartfelt gratitude to Corina Stratan, my supervisor who extended full support and dedication throughout the thesis. Through this thesis, she has helped me sharpen my research skills and provided great opportunity to experience, learn and grow. Sincere thanks to my Co-supervisor Guillaume Pierre, who has been very helpful and understanding, not only for the thesis, but throughout the entire Masters. His suggestions and guidance at the right moments have helped in moving a long way forward. Heartfelt thanks to my friends who have been like building blocks in life and extended all the warmth and love and encouragement at all times.They have made life a joyful roller-coaster ride. I would also like to Thank all the people who have helped me successfully complete this endeavor.

Contents

Chapter 1

Introduction

Clouds are gaining popularity as many users shift the development and deployment of various types of applications to the cloud environment, due to the many benefits such as greater flexibility, scalability etc offered to the application apart from the reduced hardware and infrastructure costs. The types of services provided by the clouds are classified mainly into Infrastructure as a Service(IaaS), Platform as a Service(PaaS) and Software as a Service(SaaS). Clouds provide a great platform where the resources allocated to the applications can be scaled up or down based on the consumption of the current resources, referred to as Resource Provisioning. However, in the current cloud environment, resource provisioning is mainly done based on monitoring the physical resource consumption of the nodes in the clouds. This means that measurements(metrics) such as CPU, memory, disk, network usage etc are used for the resource provisioning purposes. This approach works fine with computation intensive applications, but with other types of services offered by the clouds, it doesn't always give the best results. Most of the web applications are business critical applications, where a delay of even a few milli-seconds may result in huge loss to the company which owns the web application. Hence it becomes vital that these applications maintain their performance even in a cloud environment. This requires for high precision monitoring of the web application's performance for the resource provisioning purposes. This would mean that

the new metrics which clearly reflect the web application's performance should be developed for the provisioning purposes of these applications in the clouds. The main challenge in performing this task apart from ensuring that the web application maintains good performance, is to ensure minimum overhead in gathering the high-precision information. For accomplishing this task, we extend the Ganglia monitoring framework by developing non-intrusive sensors to gather information about the high-precision metrics from the web application. The various metrics which are measured are the response time of the server, successful requests, failed requests, average load on the server, geographical location of the clients etc. These metrics were relatively easy to measure compared to the Client Perceived Response Time. The main challenging metric which reflects the performance of the web application more precisely is the Client Perceived Response Time. This value gives the idea of the response time of the server as perceived by the client. We have devised methods through which we non-intrusively gather this metrics from the server. The experimental results indicate that the Client Perceived Response Time measured through our methods is very similar to the actual response time experienced by the clients with percentage difference less than 2%. Also, we have ensured that the high-precision monitoring framework has very less overhead. The thesis is organized as follows. Chapter 2 describes the background work, while in Chapter 3, we describe the main challenges of monitoring in a cloud environment, also describing the Ganglia monitoring framework. In Chapter 4, we discuss the design of our system, followed by the implementation techniques in Chapter 5. Chapter 6 discusses the evaluation techniques and the results of the work, with related work discussed in Chapter 7, followed by the conclusion in Chapter 8.

Chapter 2

Background

2.1 Moving to the Clouds

Cloud computing refers to the provision of computational resources on demand via a computer network. Huge infrastructural costs coupled with the maintenance overhead are some of the many reasons why industries are moving their applications to the clouds. While clouds provide the advantages of agility to the user and keep their costs to an optimum based on the consumption, they are also known to cause additional fear regarding the performance of an application in a third party environment. The services offered to the user are classified as follows.

2.1.1 Infrastructure as a Service(IaaS)

The computing infrastructure is fully outsourced as a service.The user can buy the infrastructure according to the requirements at any particular point of time instead of buying the infrastructure that might not be used for months. IaaS operates on a 'Pay as you go' model ensuring that the users pay for only what they are using. Virtualization enables IaaS providers to offer almost unlimited instances of servers to customers and make cost-effective use of the hosting hardware. IaaS

users enjoy access to enterprise grade IT Infrastructure and resources that might be very costly if purchased completely. Thus dynamic scaling, usage based pricing, reduced costs and access to superior IT resources are some of the benefits of IaaS. An Infrastructure as a Service offering also provides maximum flexibility because just about anything that can be virtualized can be run on these platforms. This is perhaps the biggest benefit of an IaaS environment. Examples include Amazon EC2 (1), RackSpace (19) etc.

2.1.2 Platform as a Service(PaaS)

Here, a development platform is provided for the developers. The code written by the end users is uploaded and presented on the web. Services to develop, test, deploy, host and maintain the applications in the same integrated environment is provided. It is a cost effective model for application development and delivery. The PaaS provider takes the responsibility for maintenance of the environment. One of the common example of this type of service is Google App Engine (8).

2.1.3 Software as a Service(SaaS)

In this case a software is rented from a service provider rather than purchasing it. The software is hosted on centralized network servers and is made available over the internet. This is the most popular type of cloud computing because of its high flexibility, great services, enhanced scalability and less maintenance. Examples include Google docs (7) and Yahoo mail (24). Both the application and the data are hosted by the service providers which facilitates the user to use the service from anywhere. Another advantage for the user is that they need not worry about the installation or updates.

2.2 Virtualized Severs

Since the advent of Virtual Machine's, time slicing and resource sharing have been widely common in cluster, grid and cloud environments. They enable the computing resources to be shared in a wise manner amongst a variety of applications. One of the biggest advantages of having server virtualization is benefit of cost and efficient resource utilization. The hardware maintenance costs is considerably reduced due to lesser physical servers. By implementing a server consolidation strategy, we can increase the space utilization efficiency of the data center. Updates and changes to the applications do not affect the others since each application would have its own 'virtual server'. Duplication of a developed standard virtual server build is easily possible. The new layer of complexity introduced through virtualization could lead to new set of problems.Apart from the vast list of advantages which prompt the users to move to a virtualized server environment, we also have to consider the side effects of this deployment as well. The very fact that the application's are not aware of the sharing can lead to performance degradation when some application's begin to have surge of memory or cpu usage or of I/O operations. Efficient resource provisioning algorithms have to be developed which would ensure that application's would not get into each others way. Resource managers have to be careful in allocating virtual machines to the physical servers to ensure best performance. Additional skills to management of Virtual machines need to be procured by the administrators of the data centers in order to perform complex root cause analysis of the problems that might arise in the shared environment. The performance of virtualized servers is widely dependent on an excellent underlying monitoring system to input data to the resource management module.

2.3 Web Applications in the clouds

In comparison to the traditional computationally intensive tasks run in a cluster or grid environment, clouds provide a new paradigm shift where companies/users can develop and run any sort of applications. Web applications are popularly used by industries to promote their business and almost every business needs its own web application. Cloud providers providing PaaS offer the ideal platform of tools to develop the web applications, ensure their availability and later help in scalability of the application depending upon on the surge in the usage, and the application developer has to pay only for the resources consumed. However, the web applications are quite different in nature as compared to the traditional computation intensive applications. The main differences of the web applications compared to the computation intensive applications is the necessity of these applications to deal with flash crowds, the difficulty to predict the server response time due to virtualization and the diversity in the types of requests which arrive at the web server, the difficulty to predict the client-perceived response time due to the diversity geographical locations of the client's etc. As clouds are not traditionally meant to take care of each individual cloud user's requirements, the common set of algorithms used for cloud management and resource provisioning do not consider Service Level Agreement or performance of the web applications. This manifests the need to perform a more high precision monitoring of the web applications in the clouds in-order to ensure that the business demands are met.

Chapter 3

Monitoring Framework

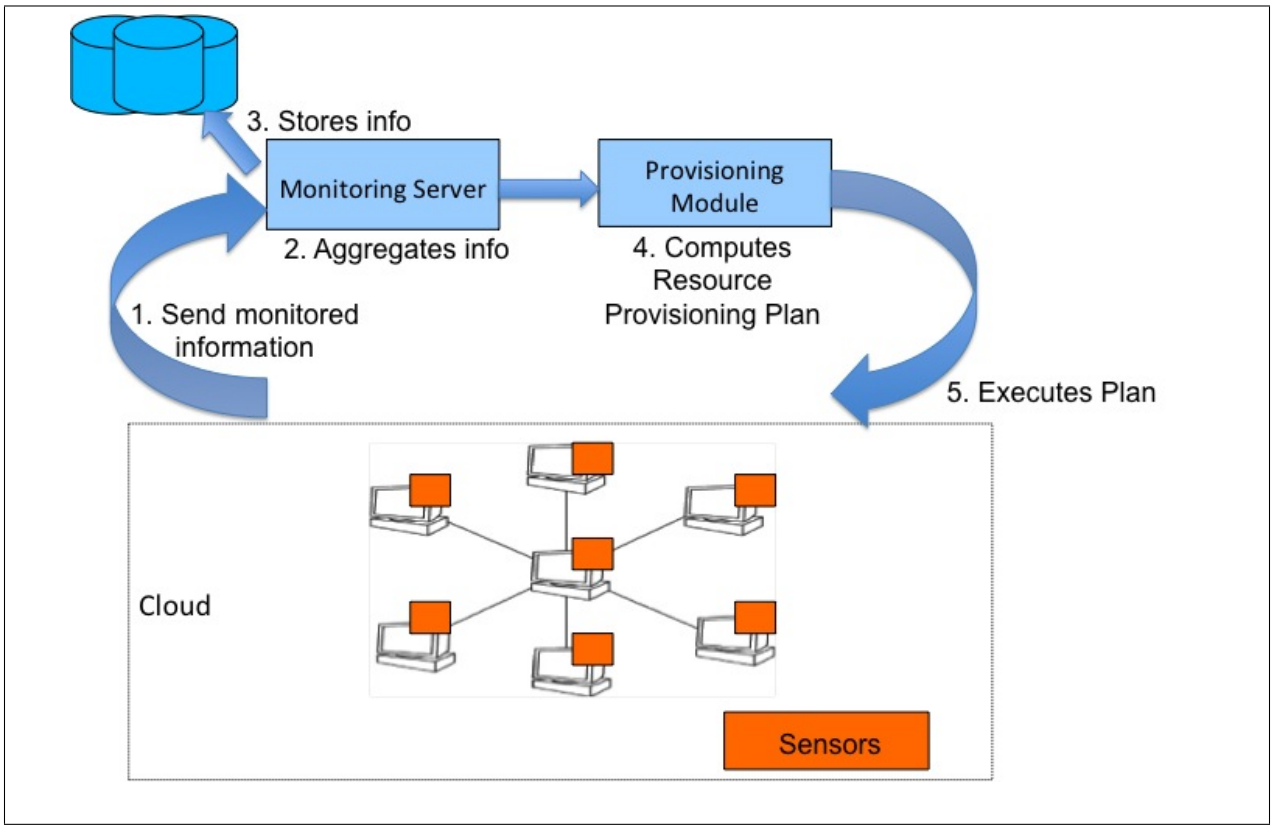

Figure 3.1 *Design of our Monitoring System Framework*

Systematic collection of information regarding the current and the past state of the cloud re-

sources is defined as the process of monitoring. Monitoring is very crucial in events of scheduling, data replicating, accounting, performance analysis and optimization of the resources, self-tuning applications etc. Since the applications are running in a dynamic environment, periodic monitoring plays a key role in deciding the health of the system by observing patterns in the resource usage for future planning. Since the clouds comprise of thousands of machines spanning widely in different geographical locations, it becomes necessary to have a monitoring framework which performs the tasks of gathering information locally, broadcasting the information, aggregating and storing the information. The general design of our monitoring framework used for resource provisioning purposes is outlined in the figure 3.1.

3.1 Challenges Of Monitoring In Clouds

The main challenge in designing a monitoring system for high-precision monitoring of web applications in the clouds is ensuring that the overhead of the monitoring framework is kept to the minimum. Since the monitoring system is designed for a distributed environment, the main challenges that arise in building a distributed environment extend to the monitoring system as well. We discuss the main issues in the following section.

- **Adaptivity:** Since the monitoring framework works in a distributed and dynamic cloud environment, where number of nodes changes dynamically, our system must be able to detect the addition and removal of nodes in the system. In addition to the number of nodes, the system should be scalable to the new metrics added to the system for the precise monitoring of the web applications.The main challenge is that the system should be able to automatically detect and cope up with the changes in the number of nodes and metrics without much overhead that would affect the performance of the system.

- **Extensibility:** We define extensibility as the ability to incorporate heterogeneity into the monitoring system with minimum intrusiveness and maintaining good performance. Extensibility is proven through the ease with which we add extra functionality in terms of different types of precise monitoring metrics for the web application into our system.

- **Manageability:** The management overhead of the system should not scale linearly with the scalability of the system. The system should also allow for most of the management to be automated and not have a linear scale of manual management with scalability of the system.

- **Frequency of Data delivery:** We evaluated and decided upon the best periodicity interval of the data transfer in the case of both the static and the dynamic data. Static data in our case comprises of information which does not change drastically over a certain period of time such as the physical node characteristics. However, dynamic data is comprised of those information which is subjected to change very frequently, such as the data regarding the web server characteristics.

- **Data Aggregation and Storage:** The information regarding the various metrics are gathered from thousands of nodes through the monitoring framework. This information has to be aggregated and stored efficiently, as it serves as input for the resource provisioning algorithms in the future. The storage mechanism must ensure that it utilizes the storage space in a very efficient manner and also must be able to fetch the desired information with less overhead.

- **Non-Intrusiveness:** For gathering information regarding the various metrics, we need sensors which function in a non-intrusive manner. This means that they should be able to obtain the data without actually halting or affecting the actual performance of the system.

- **Communication Overhead:** The information gathered from the nodes should be exchanged with least overhead in the network. This would imply that the messages have to be trans-

ferred over the network in data formats which consume less bandwidth in the network. Minimum per node overhead and minimum overall system overhead should be ensured.

- **Fault Tolerance:** Distributed systems should be designed for fault tolerance, where the occurrence of faults should not affect the performance of the entire system. The affected nodes should be easily detected and the system should be able to continue to operate and offer useful services even in the presence of failures.

There are many existing frameworks for monitoring large scale distributed systems. One of the most efficient and commonly used one is Ganglia Monitoring Framework. Ganglia monitoring framework is designed to monitor standard metrics such as CPU usage, Memory usage, disk usage, network usage etc of the resources in the distributed environment. The advantage of Ganglia is the ability to develop user defined metrics and integrate it using Python modules into the framework. We discussed the Ganglia monitoring framework in detail in the section below.

3.2 Ganglia Monitoring System

Ganglia (11) is a scalable distributed system monitor tool for high performance computing systems. It is based on a hierarchical design targeted at a federation of clusters which relies on multicast-based listen/announce protocol to monitor state within the clusters and uses a tree of point-to-point connections amongst representative cluster nodes to federate clusters and aggregate their state. Technologies such as XML is used for data representation, XDR for compact,portable data transport, and RRD tool for data storage and visualization. Low per-node overheads and high concurrency is achieved through carefully engineered data structures and algorithms. In this section, we discuss the Ganglia monitoring framework in detail and also how Ganglia solves most of the problems of monitoring in a distributed environment. The architectural design of Ganglia monitoring framework as described in (11) is outlined in the figure 3.2.

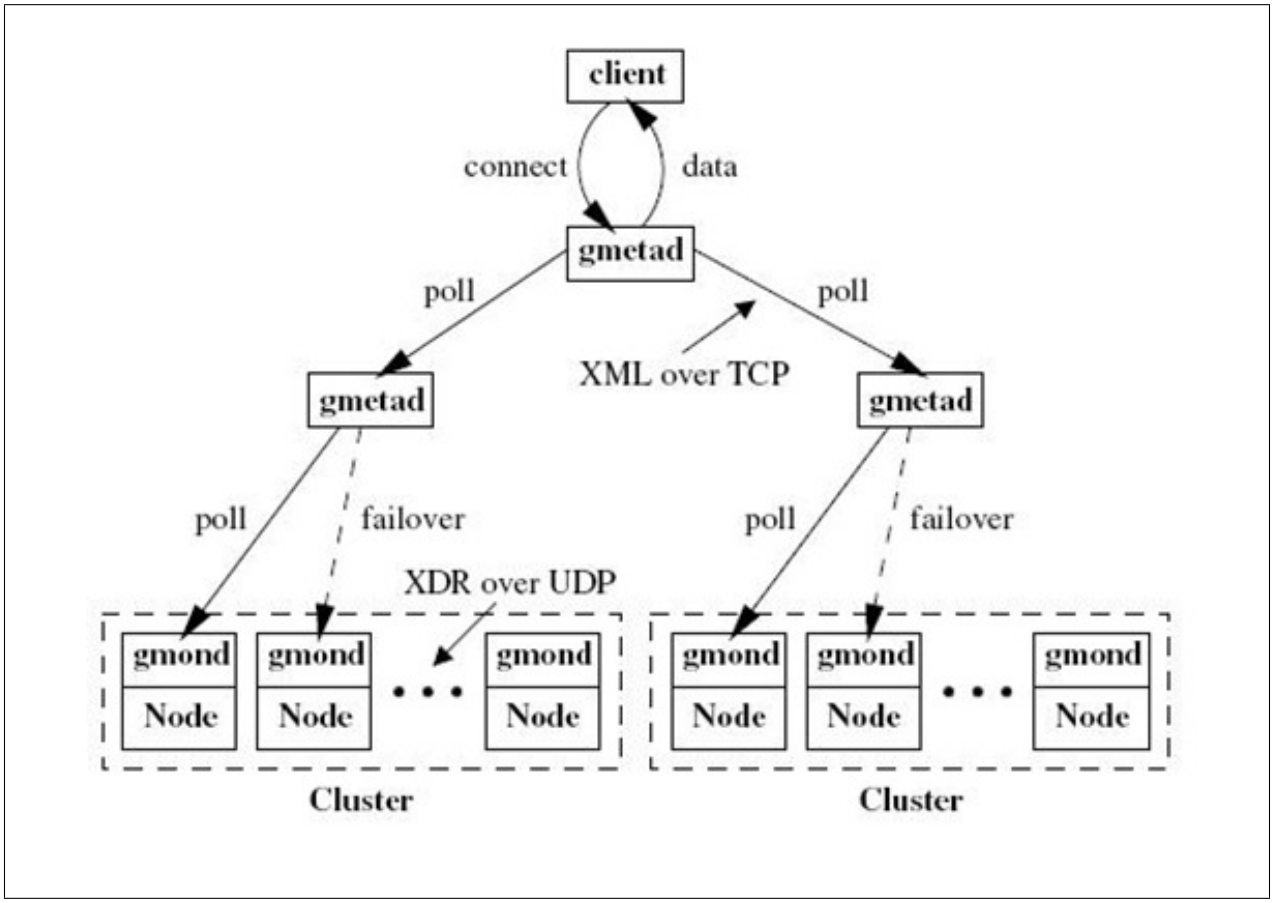

Figure 3.2 *Ganglia Monitoring System*

The Ganglia monitoring framework is widely used for monitoring the large scale distributed environments. Its popularity is mainly due to the characteristics outlined in the following section.

Multicast Data Exchange The data gathered locally at the nodes has to be exchanged between the nodes and later be aggregated at the head node, so that the resource provisioning decisions can be made based on the information gathered at the head node. Ganglia uses multicast data exchange mechanisms, where the data gathered locally at each node are multi-casted to all the other nodes in the same cluster. The multicast mechanism has many advantages, since it becomes easier to add and remove nodes to the cloud. If a new node is added to the cluster in the cloud, it would

broadcast the local information to all the other nodes in the same cluster and thus it becomes easier to detect the addition of node. Similarly, when a node stops multicasting, over a certain period, the other nodes would realize that the node has been dead or has un-joined the cluster group. The other advantage that arises from this mechanism is scalability, since it would become easier to scale the monitoring framework as more nodes are added to the cluster without encountering much overhead. The periodicity of the multi-cast exchange can be adjusted and this gives additional advantages, as we can set different periodicity intervals for the different metrics collected. Some metrics which remain constant over a longer interval can be collected with lesser frequency while metrics which change dynamically over time can be collected more frequently.

XML Data Formats: The data format in which Ganglia exchanges information amongst the nodes is XML. XDR format is used for the data transfer between the different nodes. XDR format allows data to be exchanged between different heterogeneous machine, thus making it possible to have different types of machines being a part of the cloud, also ensuring that not much bandwidth is consumed during the monitoring process.

Gmond and Gmetad: In-order to obtain the metrics non-intrusively from the nodes, Ganglia uses two types of daemons on the nodes. The daemon which runs locally on every node which is collecting the metric is Gmond. The Gmetad daemon runs on the head nodes of each cluster, it polls one of the nodes in the cluster, since every node has information about every other node in the cluster due to the multi-cast mechanism of Ganglia. The Gmetad node has to be given information about one or more of the nodes in the cluster, so that it can poll that particular node for the metric information. This feature is also in favor of fault-tolerance, since the gmetad can be information of more than one node, to poll the information from.

Aggregation and Storage: When the Gmetad daemon collects the information about the nodes in its cluster, it computes the aggregated information for the cluster, as it helps to store the information about the nodes efficiently. The storage is done using the Round Robin Database, where the

collected data is stored in a round robin fashion and the old data is over-written by the new ones. In this manner, the storage will be managed efficiently, since there wouldn't be growing storage concerns, linearly with the scalability of the system.

3.2.1 Limitation of Ganglia Monitoring Framework

Currently Ganglia has sensors to monitor the standard metrics such as CPU usage, memory usage etc. It does not monitor metrics which are suited for the provisioning of the web applications in the clouds. The standard metrics monitored by Ganglia do not precisely reflect the performance of the web applications. For example, we cannot predict correctly the number of requests on the web server based on the CPU usage of the physical node, nor can we predict the response time of the web server based on the standard metrics. This requires a call for more precise monitoring of the web applications, and these metrics have to be integrated to the Ganglia monitoring framework. In the next chapter, we discuss the metrics which are developed for the purpose of precisely monitoring the web applications.

Chapter 4

High precision monitoring

Since resource utilization indicator metrics such as CPU, memory, disk usage etc. are not a reliable reciprocal of web applications performance in virtualized cloud environments, the focus needs to shift to the precise metrics reflecting web applications performance. This is both a technical necessity and a political necessity as the users of the application are not going to accept any of the standard metrics other than whether the application feels fast or slow to them.

4.1 High Precision Monitoring Metrics

In the section below, we list out the metrics which would be useful for the precise monitoring of web applications along with the reasons why we find them to be important.

4.1.1 Server Response Time

Server response time is defined as the time taken by the server to process the request from the client. Since the web applications are running on the web server, the web server's performance plays an important role in deciding the performance of the web application.

4.1.2 Client Perceived Response Time(CPRT)

Client Perceived Response Time is defined as the time taken to process the request as perceived by the client. It comprises the delay at the client side, the network delay and the server side delay. This is considered one of the important metrics to decide the performance of the web application, since the performance of the web application is best decided by the client himself. So it becomes necessary to compute this value to decide how the server is performing from the client's perspective. It also becomes equally important to measure this value at the server side, without being intrusive to the client nor to the web application.

Challenges of obtaining CPRT

Unlike the server response time, which is relatively easy to obtain from the server side, obtaining the Client Perceived Response Time involves more challenges. The main challenge is that we are aiming to measure this value from the server itself. This would mean that we would have to devise efficient ways to extract this information without affecting the performance of the web application. Currently, this value is measured through methods such as embedded java script in web pages, polling from different geographical locations etc. which are described in detail in the Related works Chapter. However, due to the many drawbacks of these methods, we have decided to use packet sniffing methods at the server for calculating the CPRT values from the packets exchanged between the client and the server.

4.1.3 Precise Information from Server Logs

The web server logs are sources of vital information about the nature of requests arriving at the server and deciding upon the server's performance based on the processing information in the web logs. It becomes essential to parse the web logs for useful information. The web log parsers would help in obtaining aggregate information about the performance of the web server. For example, it

helps in gathering information about how many requests were successfully processed, how many requests could not be processed due to various types of errors such as 404(not found), 403(forbidden) etc. Based on this information, we can decide the performance of the web server over a period of time.

4.1.4 GeoIP

Since the web server receives requests from client's distributed geographically, it becomes necessary to understand the location of the client for the resource provisioning. This would help in placing more resources closer to the location of the clients from where the majority of the requests originate. For this purpose, we use GeoIP, defined as obtaining the Geographical location of the request based on the IP address of the request from the client. The IP address of the client can be easily obtained from the web server logs. Once the IP address is obtained, we would need to map it to the geographical location using the GeoIP database.

4.2 Overcoming Monitoring Challenges

The main challenge of monitoring in the clouds is to maintain the overhead of the monitoring system to a minimum. Although the Ganglia monitoring framework solves most of the challenges described in Chapter 3, the main challenge that remains, is that of keeping overhead to a minimum by reducing the overhead locally at each node by the usage of non-intrusive sensors to gather the high precision metrics. This has been done using various tools for gathering High Precision Metrics as described in the next section. Python Modules were developed and integrated to Ganglia to obtain the new precise metrics.

Chapter 5

Implementation

In the previous sections, we have described the Ganglia monitoring framework and the high precision metrics which we had chosen for monitoring and resource provisioning of the web applications. In this section, we describe in detail, the various tools used for non-intrusively gathering the information from the web application.

Web Server

The web server we had used in our implementations is Nginx (21). Nginx is the web server we have used to deploy the web applications. It is a lightweight, high-performance web server. Nginx quickly delivers static content with efficient use of system resources. It can deploy dynamic HTTP content on network using FastCGI handlers for scripts. Nginx uses asynchronous event-driven approach to handling requests which provides more predictable performance under load.

5.1 Server Response Time

The server response time is the time taken for the web server to process the request from the client. The Nginx server has a module which can be enabled to write the response time of the web server

into the web server logs. Thus, with the help of this module, the response time would be written to the logs along with the other details of the request. However, since the web server received thousands of requests from the clients, it becomes necessary to gather the response time details from the log in an effective manner. For this,we use a tool called logtail, which extracts the log information efficiently and then computes the average response time for the requests at 5 minute intervals.

5.1.1 Logtail

Logtail (23) is a perl script which allows administrators to watch entries, as they are added, in any number of log files on one or more machines on a network. As new entries are added to the web server logs, logtail makes it easier to parse the new information and extract the response time information from the individual requests to compute the average response time.

5.2 Client Perceived Response Time

For measuring the value of the Client Perceived Response Time, we use packet sniffing techniques, where the packets exchanged between the client and the server are logged and the time taken for the various events like connection establishment, sending requests, processing the requests, terminating the connection etc, are computed. The packet sniffer used is called Justniffer.

5.2.1 Justniffer

Justniffer (16) is a TCP packet sniffer. It can log network traffic in a customizable way and also includes information such as the response times. It is very useful to troubleshoot the performance issues. It can collect low and high level protocol and performance info reconstructing the tcp flow in a reliable way using portions of the linux kernel code.

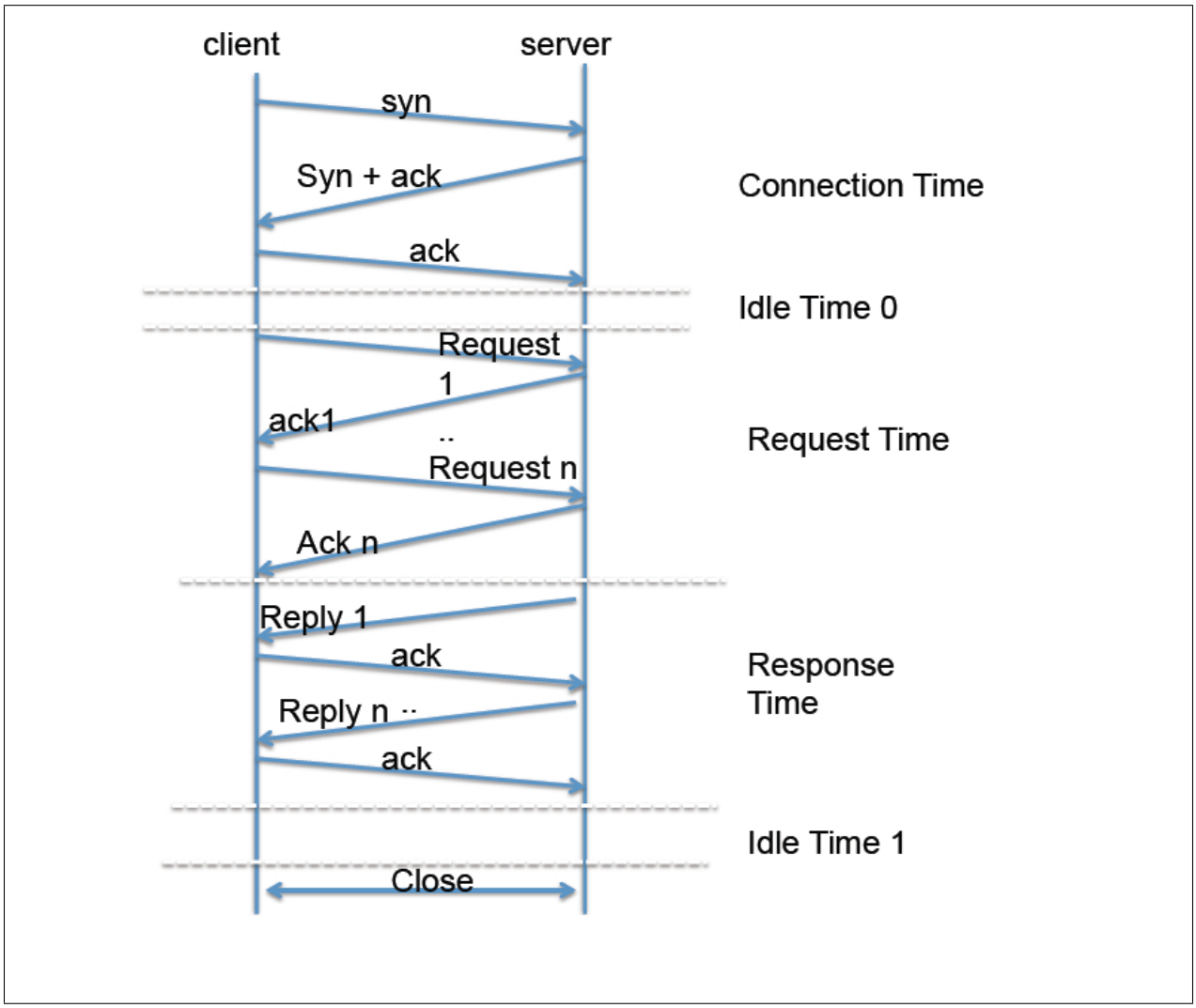

Figure 5.1 *Http Connection Establishment*

From the figure 5.1 taken from (16), we can observe that Justniffer computes the following values:

- Connection Time: The time taken to establish a connection between the server and the client.
- Idle Time 0: The time after the connection has been established and before the first request it send from the client.
- Request Time: The time taken by the client to send the requests to the server

- Response Time: The time taken by the server to respond to the client requests
- Idle Time 1: The time interval before the next requests are send by the client or before the client closes the connection.

The Client Perceived Response Time is computed as the sum of the above mentioned values. It comprises the client side processing delay, server side processing delay and also the network delays. This information is obtained at the server.

Drawbacks of Justniffer

Justniffer uses libpcap library for the packet capture. The journey of the packet during the packet capture is described as follows.

- The packet arrives at the network interface and an interrupt is sent to the kernel driver for that interface, to notify about the arrived packet.
- The packet is then placed in the kernel buffers with many CPU cycles involved during the traversal of the packet in the kernel.
- Context switch occurs, as the packet has to be passed on to the user space from the kernel space.
- The packet it passed on to the Justniffer application in the user space.

The main drawback is that the procedure mentioned above is repeated for every targeted packet arriving at the interface. This in-turn results in huge loss of packets in high speed networks, since the kernel may not be able to pass on all the packets to the Justniffer application in the user space, due to buffer size constraints leading to the loss of the packets. Thus, we need to enhance the functionality of Justniffer using kernel enhancements, so that we can reduce the packet loss. We have used two enhancement techniques for enhanced packet capture as discussed below.

Enhancement using Streamline

Streamline (2) is an operating system I/O subsystem that minimizes copying and context switching by using large shared ring-buffers for transport and by relocating I/O operations to the most suitable hardware and software environment. Streamline supports in-kernel execution, but also automatic offloading to co-processors, peripheral devices and even remote machines. By enhancing Justniffer using the Streamline kernel module, we can reduce the packet loss at the kernel.

Enhancement using PF_RING

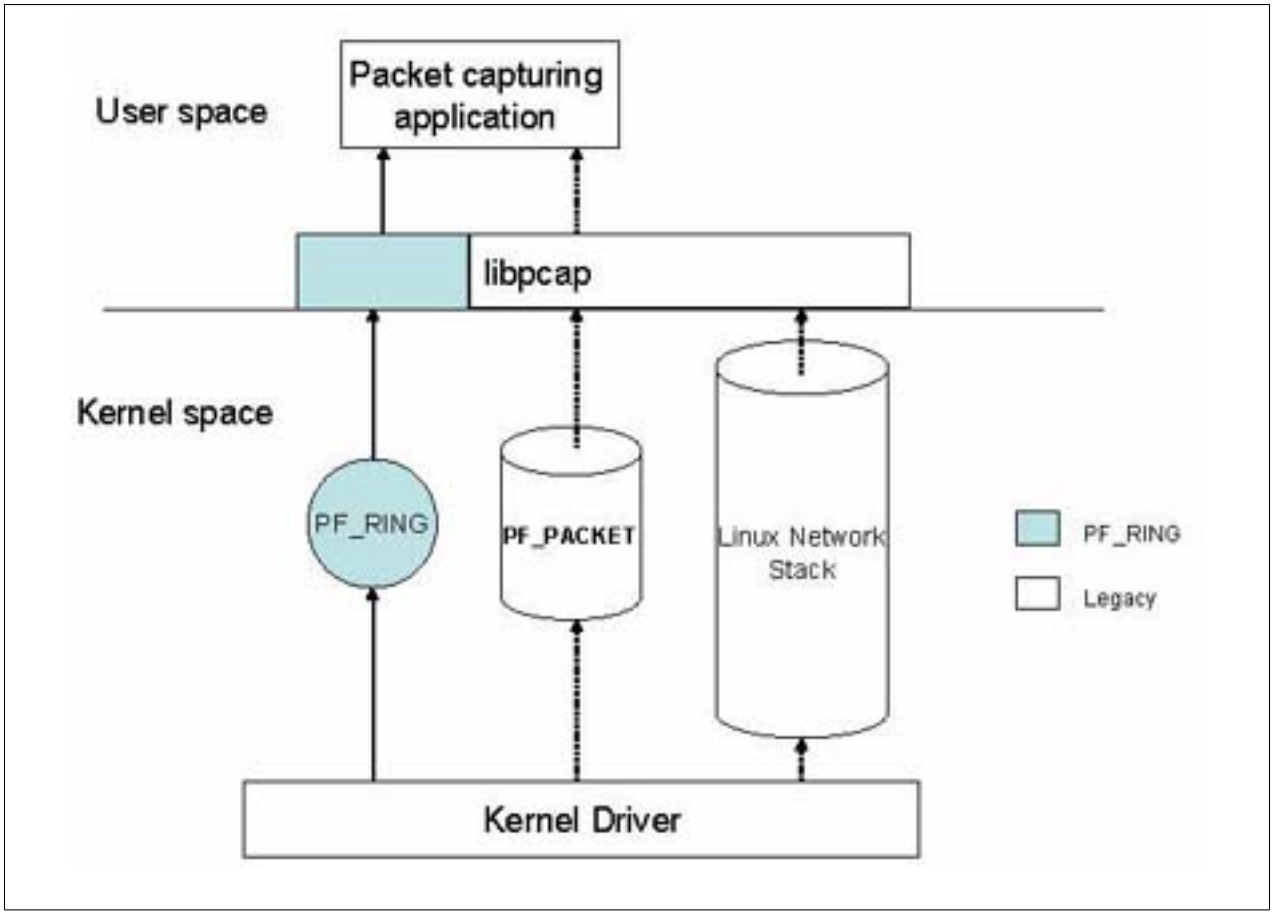

Figure 5.2 *PFRING Architecture*

Pf_ring (17) is a new type of network socket that dramatically improves the packet capture

speed with the help of ring buffers in the kernel(Figure 5.2 as cited in (17)). The packets captured by the network interface are placed in the ring buffers and then passed on to the Justniffer application in the user space. This reduces the packet traversal and placement in various kernel buffers, thus improving the efficiency of packet capture.

5.3 Precise Metrics From Server Logs

The web logs contain important information about the requests arriving at the server, and since the web logs are dynamically logging the request details, we would require a web log parser to extract useful statistical information about the web server's performance. For this purpose, we use the Analog web parser.

5.3.1 Analog

Analog (22) web parser uses as input the web server log files and generates statistical reports which help us in deciding the performance of the web server during a particular interval of time. Examples of the generated reports include the number of successful requests, failed requests, pages frequently accessed etc. This information proves useful while performing the resource provisioning of the web application in the clouds.

5.4 GeoIP

The web server logs also contain information about the IP address of the clients, which would be useful to extract their geographical location using the GeoIP database which maps the IP address of the clients to their geographical location. In order to achieve this, we used GeoIP database patch integrated into the Analog web parser, so that we can obtain statistical information about the num-

ber of client requests from different geographical locations.

Implementation Overview

Fig 5.3 shows the overview of the implementation of the system. It depicts the different sensor modules within a local node for gathering the high precision metrics and feeding to Ganglia.

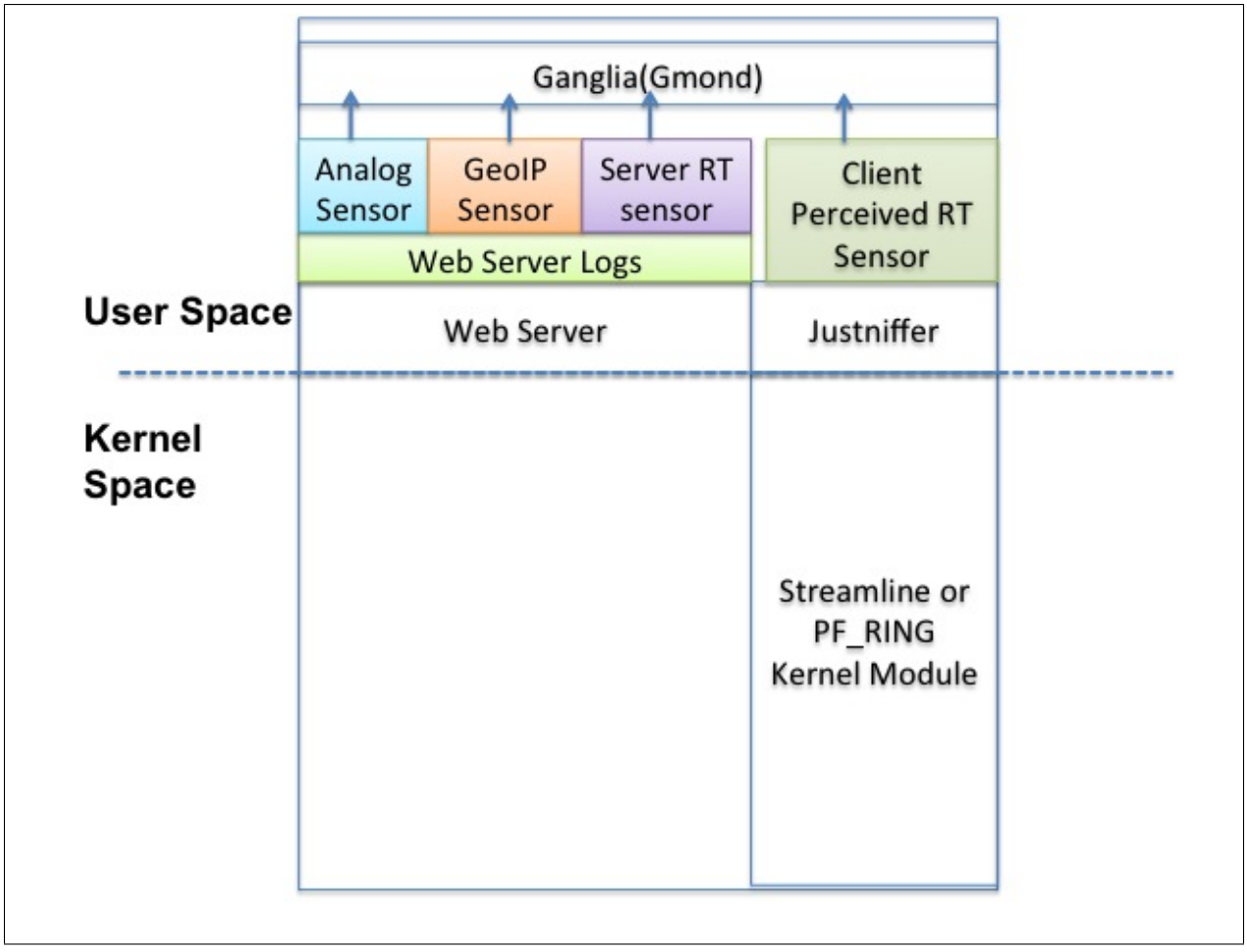

Figure 5.3 *Implementation overview of a single node*

Chapter 6

Evaluation

Evaluating the performance of a monitoring system spanning thousands of nodes in the dynamic cloud environment accessed by users all over the globe in an un-predictable and un-timely manner surfaces a lot of challenges. We have been focusing on developing a monitoring system that would ensure that the profit gained by the user's migration into the clouds is not drained out either by the poor performance of the business critical web applications or by the overhead induced by the monitoring system in the cloud environment. Tracking the end user's performance in a efficient and non-intrusive manner, simultaneously ensuring optimized utilization of the cloud resources has been the main motive of our scheme. We evaluate our scheme using Xen virtual machines deployed on DAS nodes configured with the Linux 2.6.26-2-xen-686 Kernel.

In this section we describe the various tools which we used in our experiments

1. *Httperf*

   Httperf (9) is a tool for measuring the web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring the server performance.

2. *Autobench*

   Autobench (12) is a simple perl script for automating the process of benchmarking a web

server(or for conducting a comparative test of two different web servers). The script is a wrapper around httperf. Autobench runs httperf a number of times against each host, increasing the number of requested connections per second on each iteration, and extracts the significant data from the httperf output, delivering a CSV or TSV format file which can be imported directly into a spreadsheet for analysis.

3. *Iperf*

   Iperf (15) is a tool used for measuring the TCP and UDP bandwidth performance between the client and the server machines. It reports the bandwidth, delay jitter, datagram loss in the network.

4. *cURL*

   cURL (20) is a command line tool for transferring data with URL syntax and supports various protocols. We had used cURL in our experiments on the client side in-order to make requests to the web server and measure the actual response time for processing the request from the client side. cURL provides the following information which was utilized to compute the response time at the client side.

   Connection time - The total time to establish connection between the client and the server

   Pre transfer time - The time between when the connection is established and before the first transfer of information takes place

   Total time - Time taken for the transfer of information after the connection has been established

   The response time is calculated from the above values using the formula:

   Response time = Connection time + Pre transfer time + Total time

5. *Firebug*

   Firebug (5) is a firefox plugin which helps to accurately analyze network usage and perfor-

mance. It also provides details of the connection time and the transfer time between the client and the server. We had used it in our experiments to verify the accuracy of the client perceived response time against the actual client response time provided by firebug. The output from firebug is illustrated in the figure 6.1.

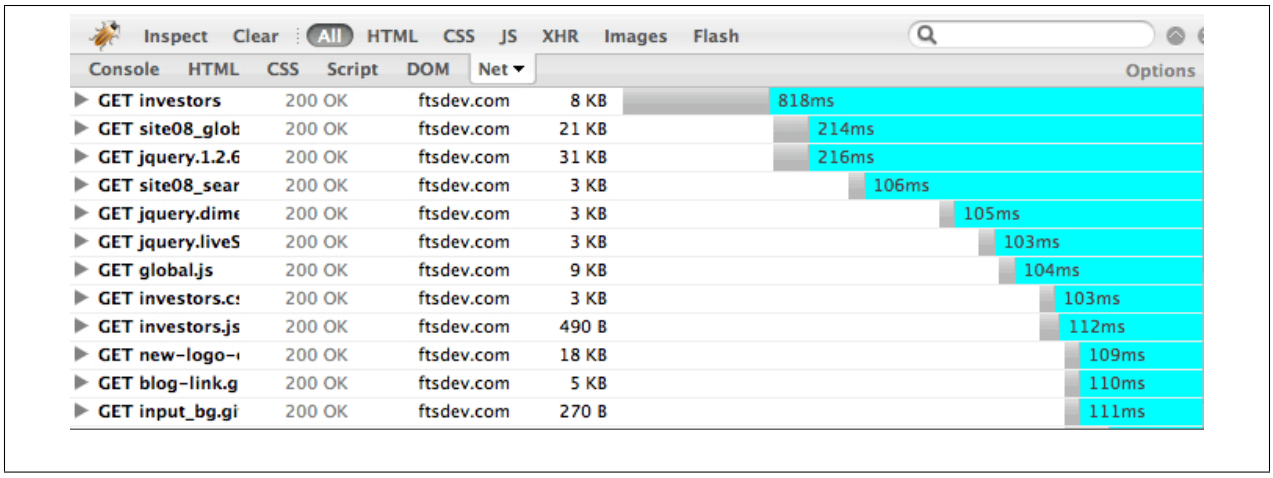

Figure 6.1 *Firebug*

6. *Tcpdump*

   Tcpdump (10) is a powerful command line packet analyzer which uses libpcap library for the network traffic capture. It provides useful statistics about the number of packets which were captured at the server and hence provide information which is useful for measuring the packet loss at the server side.

6.1 Accuracy of client perceived response times

In this test, we measure the percentage differences between the actual response time of the client, measured using cURL and firebug, against the Client Perceived Response Time(CPRT) values we had computed using Justniffer.

6.1.1 cURL vs Justniffer

Client host name	Client IP	Client RT(Curl)	Client perceived RT	diff(s)	% diff
napa1.tele.pw.edu.pl	194.29.150.137	0.081	0.079342	0.001658	2.0469135802
		0.078	0.078932	0.000932	1.1807631886
		0.081	0.079892	0.001108	1.3679012346
		0.081	0.078827	0.002173	2.6827160494
		0.08	0.078619	0.001381	1.72625
napa2.tele.pw.edu.pl	194.29.150.138	0.078	0.077819	0.000181	0.2320512821
		0.079	0.078065	0.000935	1.1835443038
		0.078	0.077653	0.000347	0.4448717949
		0.078	0.07773	0.00027	0.3461538462
		0.078	0.077767	0.000233	0.2987179487
Ricepl-1.cs.rice.edu	128.42.142.41	0.413	0.412777	0.000223	0.0539951574
		0.414	0.412768	0.001232	0.2975845411
		0.413	0.412933	0.000067	0.0162227603
		0.415	0.413585	0.001415	0.3409638554
		0.414	0.412823	0.001177	0.2842995169
Ricepl-2.cs.rice.edu	128.42.142.42	0.411	0.410492	0.000508	0.1236009732
		0.411	0.410454	0.000546	0.1328467153
		0.411	0.437619	0.026619	6.0826883659
		0.432	0.435787	0.003787	0.8690025173
		0.411	0.410426	0.000574	0.1396593674
planetlab1.aut.ac.nz	156.62.231.242	0.874	0.87336	0.00064	0.0732265446
		0.874	0.873491	0.000509	0.0582379863
		0.873	0.873247	0.000247	0.0282852389
		0.873	0.873321	0.000321	0.0367562443
		0.873	0.873206	0.000206	0.0235912259

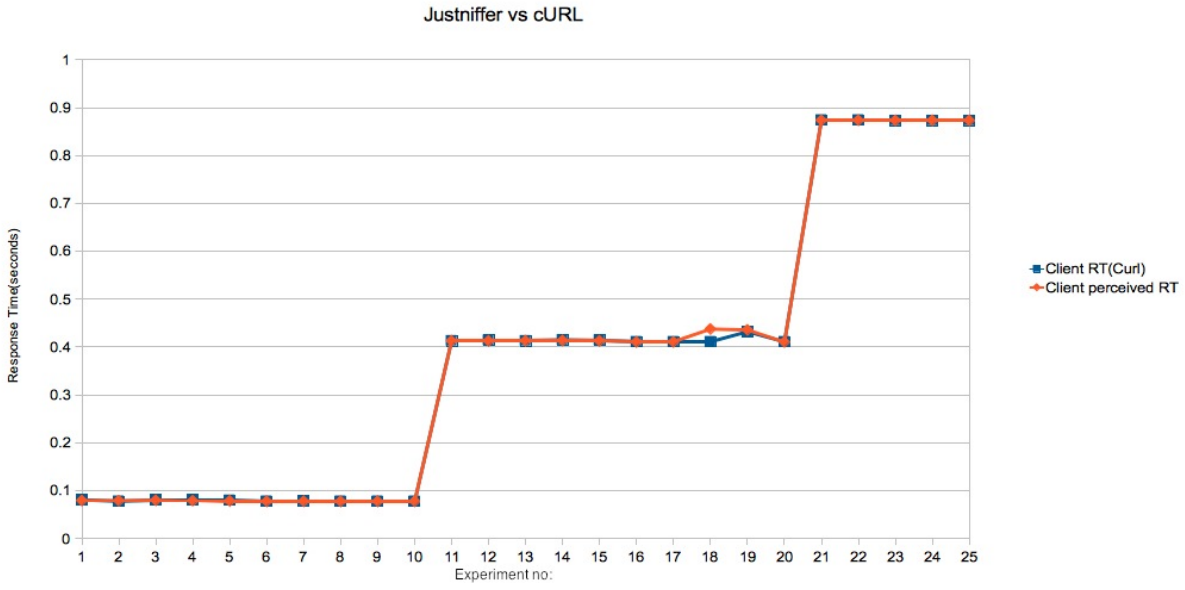

Figure 6.2 *Justniffer Vs cURL*

The client requests were sent from the planet lab nodes in different geographical locations using cURL and response time at the clients were measured using cURL, while Justniffer was used to

compute the Client Perceived Response Time(CPRT) at the server side. In figure 6.2, we observe that the measurements from cURL and Justniffer are very similar.

6.1.2 Firebug Vs Justifier

The experiment in the above section was repeated in a similar manner, except that in this case, we used firefox browser with firebug plugin to send the requests from the client side. However we observed that there were more differences between the values obtained from firebug and justniffer. This is because firebug is a firefox plugin and we would also have to consider the performance of the firefox application as well while computing the response time. These factors seem to increase the actual response time observed at the client as shown in figure 6.3.

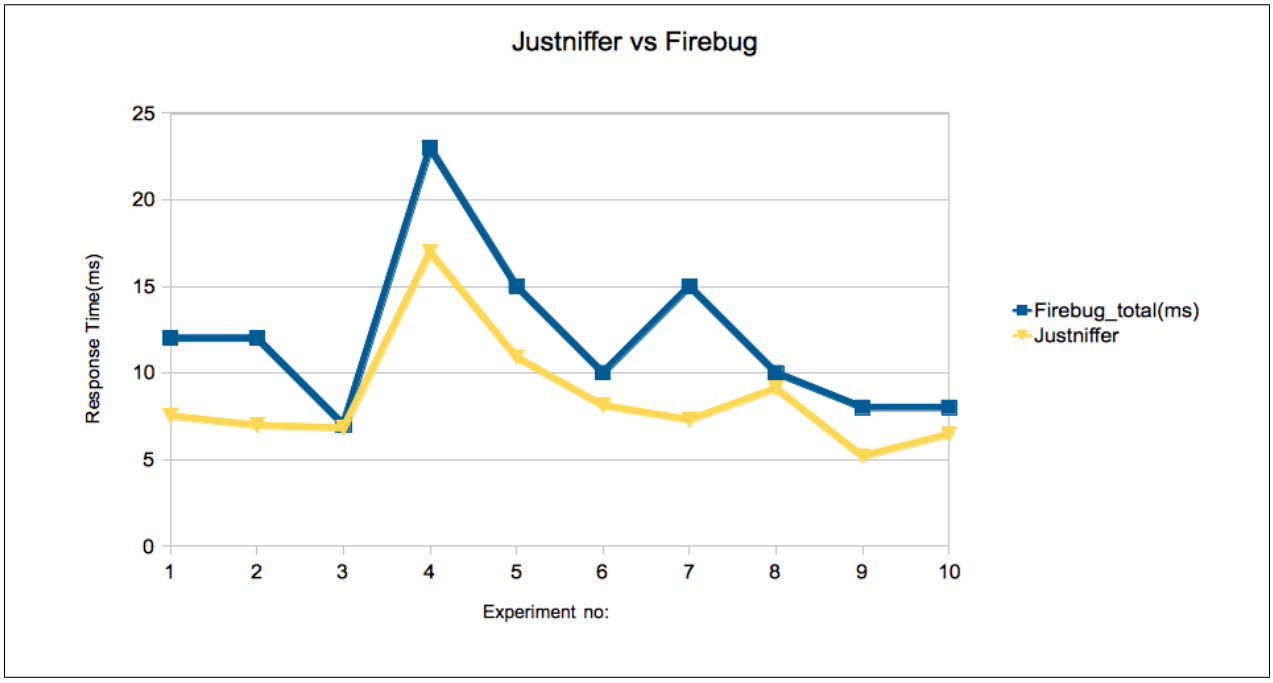

Figure 6.3 *Justniffer Vs Firebug*

6.2 Accuracy of Client Perceived Response Time under High Loads

In the earlier section, we have seen that the Client Perceived Response Time value as measured by Justniffer at the client side is very similar to the actual Response Times as seen at the client side for single requests sent to the server. In this section, we observe how the CPRT measured by Justniffer varies with increased load arriving at the server. For these experiments, we use the Httperf tool to flood the requests to the server, and calculate the CPRT using Justniffer at the server side. Httperf reports the average response time at the client side for all the requests that were sent in that particular test run. In figure 6.4 we flood the server with 100 requests/s and observe the accuracy of Justniffer under high loads.

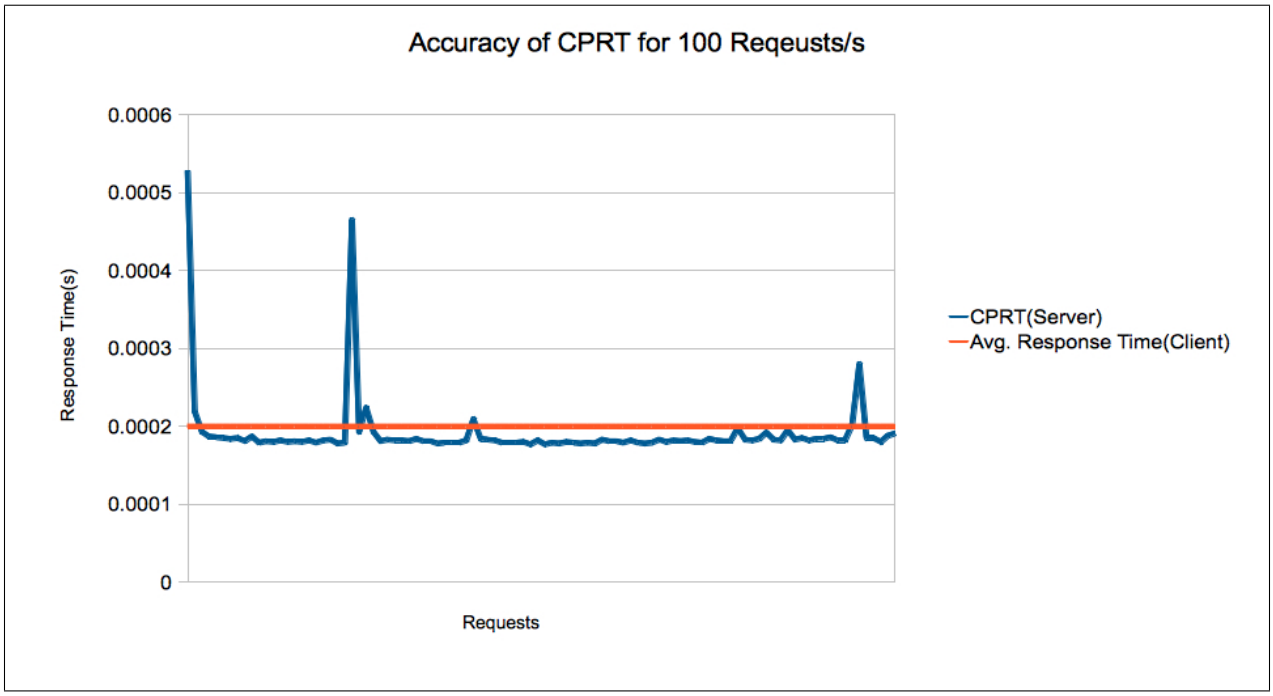

Figure 6.4 *Accuracy of CPRT with 100 requests/s*

The CPRT values calculated by Justniffer is almost very similar to the average response time value calculated at the client side using the httperf tool. Now, we increase the load to the server

and observe the accuracy of the CPRT for higher load values.

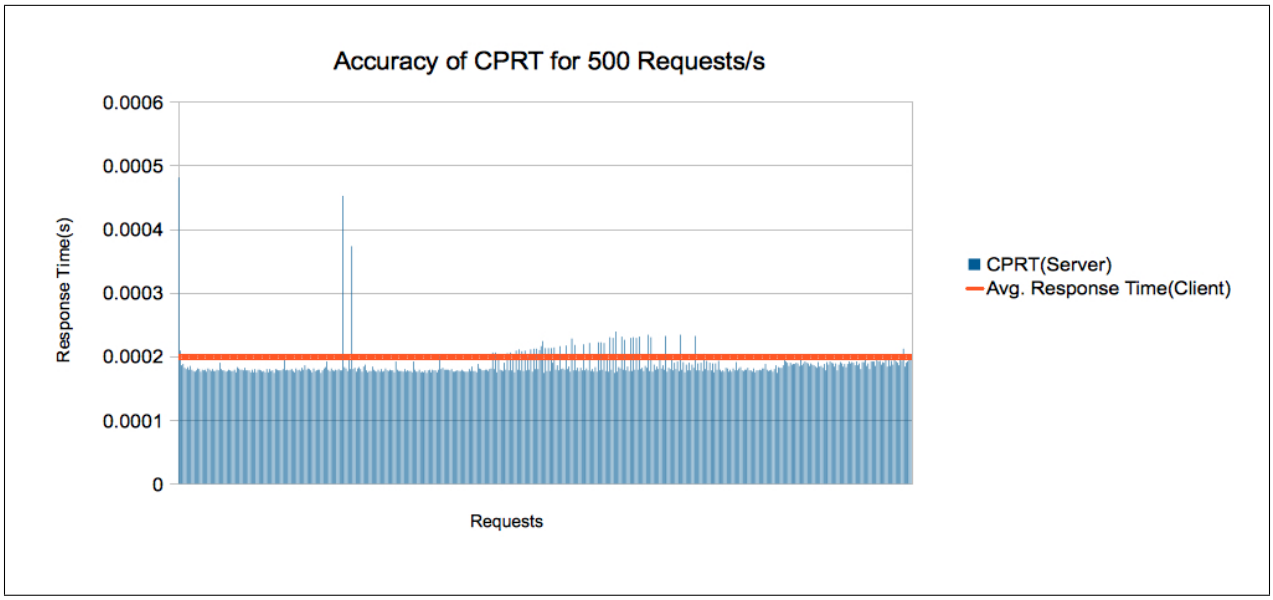

Figure 6.5 *Accuracy of CPRT with 500 requests/s*

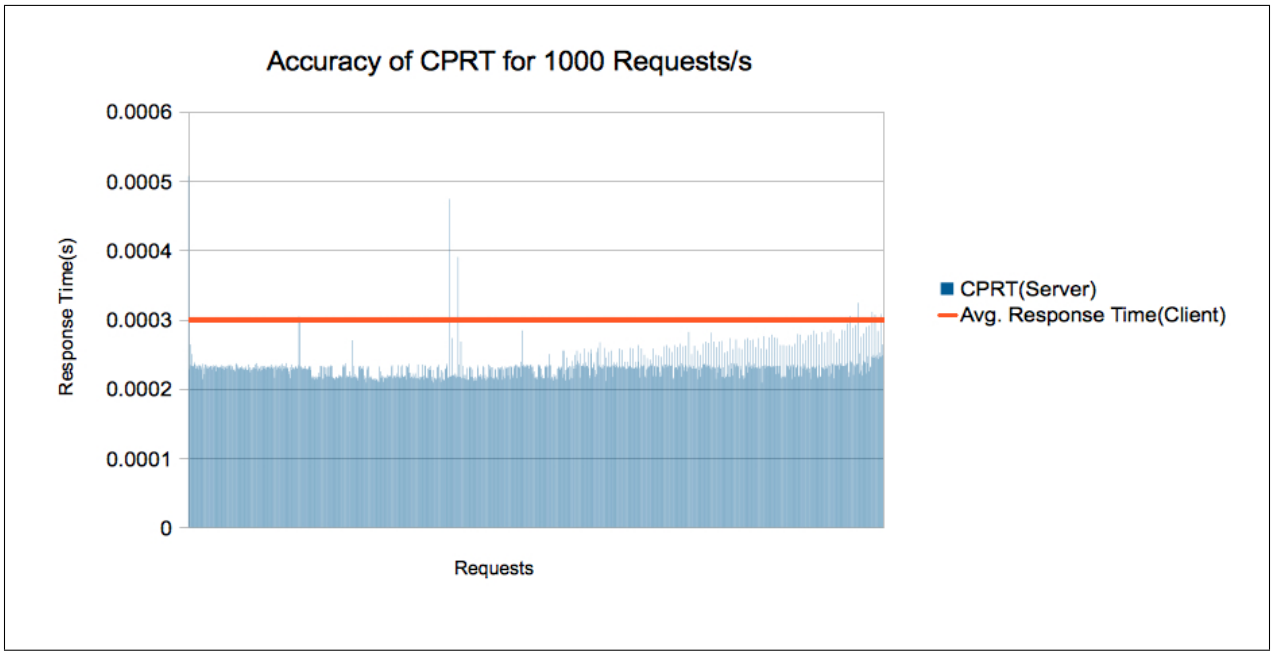

Figure 6.6 *Accuracy of CPRT with 1000 requests/s*

In the figure 6.5, we can observe that the CPRT values measured at the server is very close to the average response time value measured at the client side. From the graph, we can also observe

that over time, the response time of the server increases. This is because, since the server is loaded with many requests, it becomes a bit slower in processing the requests and thus the response time of the server increases. This is more clearly depicted in the figure 6.6. In this case, most of the first half of the requests have a steady CPRT, while over time due to the increasing load, the response time for the other half of the requests is higher, thus bringing the average value of the response time of the requests to 0.003 seconds.
In all the above experiments, we can clearly observe that Justniffer performs well under high loads and calculates the Client Perceived Response Time with good accuracy at the server side.

6.3 Scalability

Scalability in terms of monitoring systems can be defined as the ability of the monitoring system to sustain good performance under the circumstances of growing number of the resources, events or the users. Monitoring systems should ensure good performance with minimum intrusiveness upon the entities being measured.

6.3.1 Increased number of connections to the Web servers

In this experiment, we observe the behavior of the web server to increasing number of connections as well as requests. First, we perform this experiment using the ideal scenario, of increasing the number of connections as well as the number of requests using the Autobench tool. We start with 10 connections/second with one request per connection, and increase it to 50 connections/second with a step rate of 10. We notice that the average reply rate is same as the request rate, and that there are no errors, implying that there is no packet loss. The results obtained are plotted below in figure 6.7. Now, we increase the connection rate as well as the request rate to observe the effect of increasing load on the web server(figure 6.8). In this case, we increase the number of

connections/second from 500 until 1000, with a step rate of 50. One request is sent through each connection.

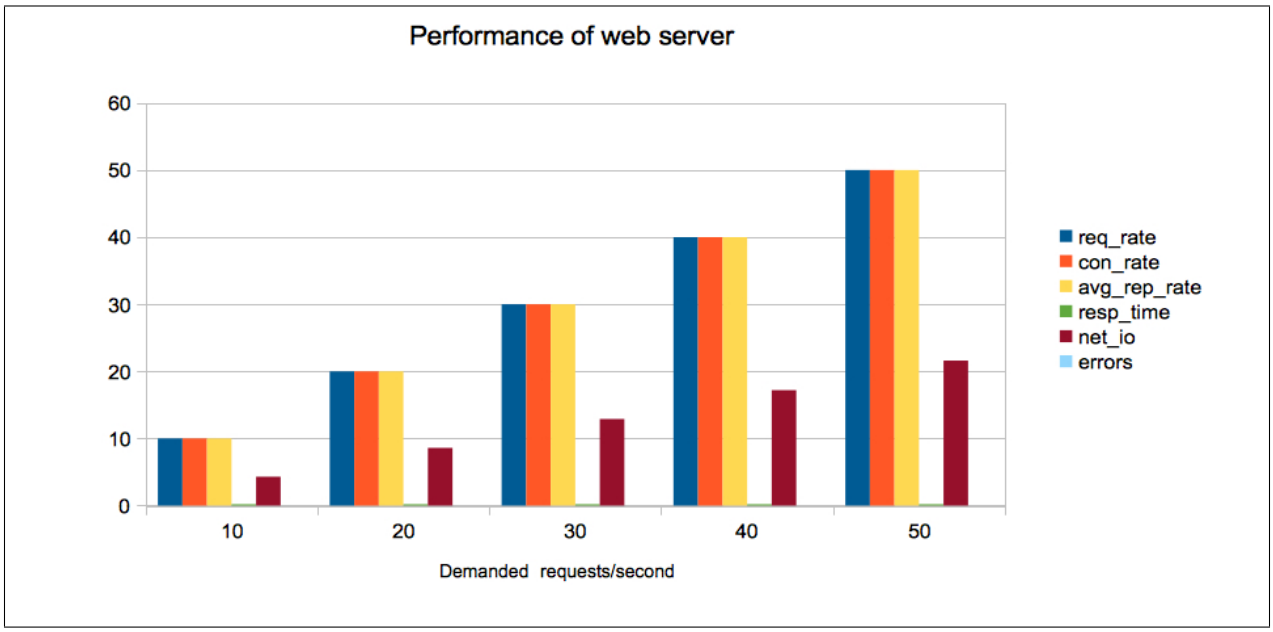

Figure 6.7 *Performance of Web Server*

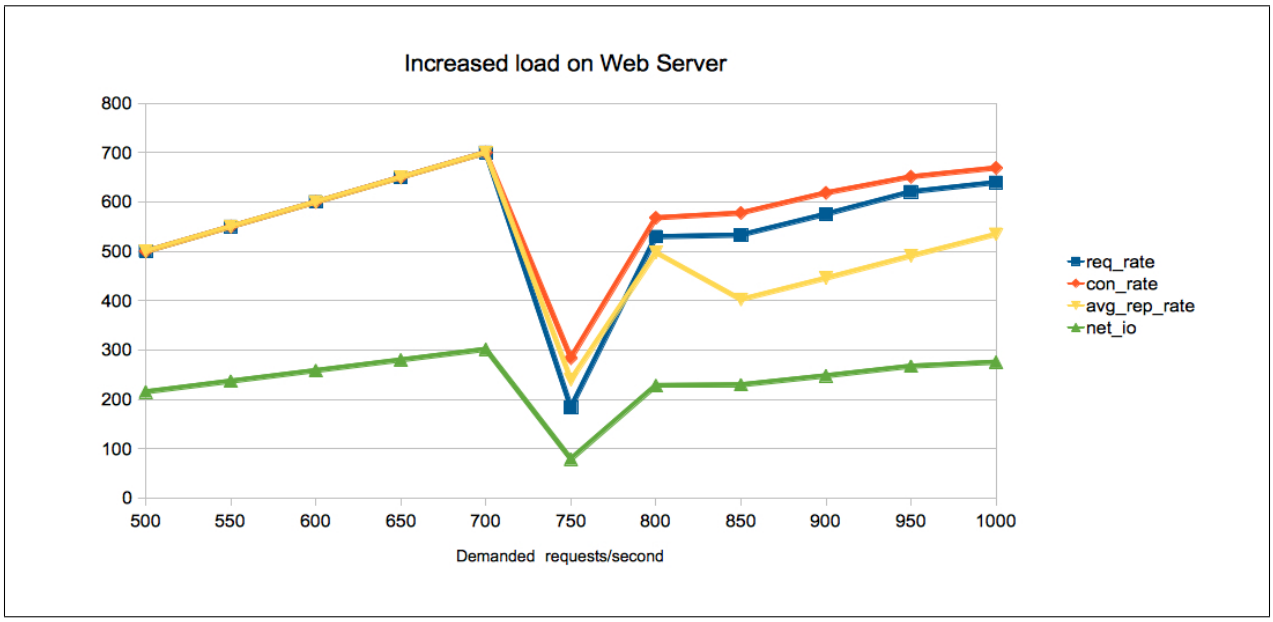

Figure 6.8 *Increased Load on Web Server*

Initially, in the graph we observe that the connections/second and the requests/second are in-line with the reply/second. This continues until we reach 700 connections/second, after which we observe a drop in the number of connections/second and the number of requests/second. This is due to saturation at the client side, and the client becomes unable to send the demanded requests/second. Later at a demanded request rate of 800 requests/second, we observe that the client can only send 500 requests/second and can only establish 550 connections/second. However, the server responds at a reply rate of 500 replies/second, but afterwards, the rate of replies is lesser than the rate of requests arriving at the server. This is because the server is also getting saturated and suffers from packet loss, which justifies the reason for the lesser reply rate as compared to the request rate(figure 6.9)

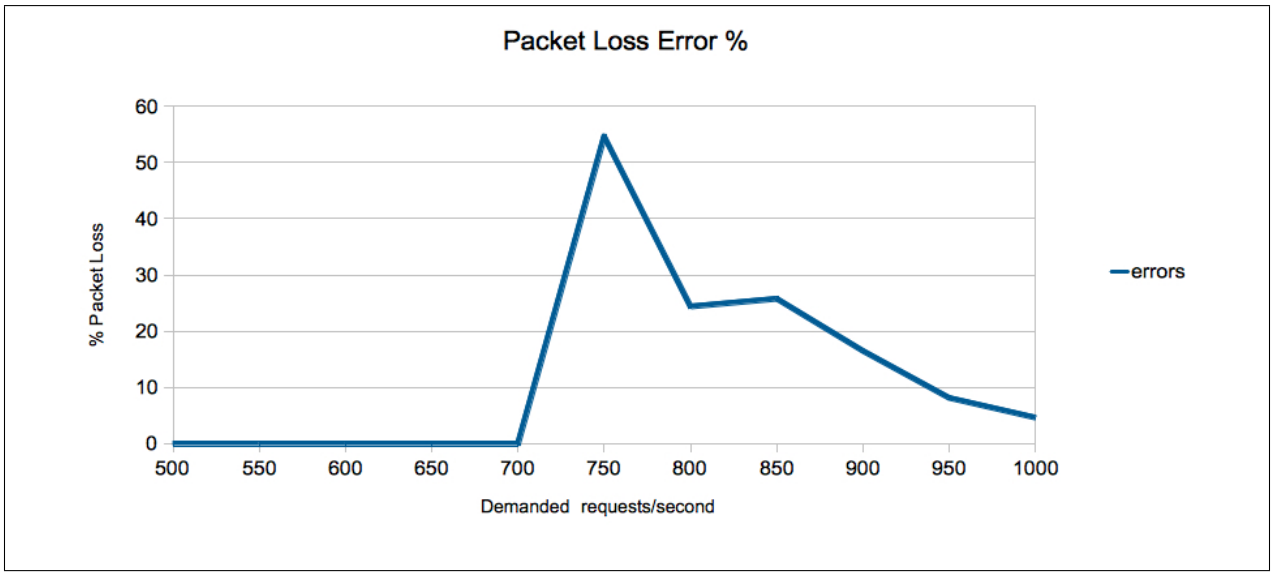

Figure 6.9 *PacketLoss Error*

Also, we observe a linear increase in the response time from the server at high loads, which is plotted in figure 6.10

As the server becomes flooded with requests, it takes more time to process the packets and thus the response time increases as well.

Through this experiment, we have proven another aspect of scalability, that there is less linear

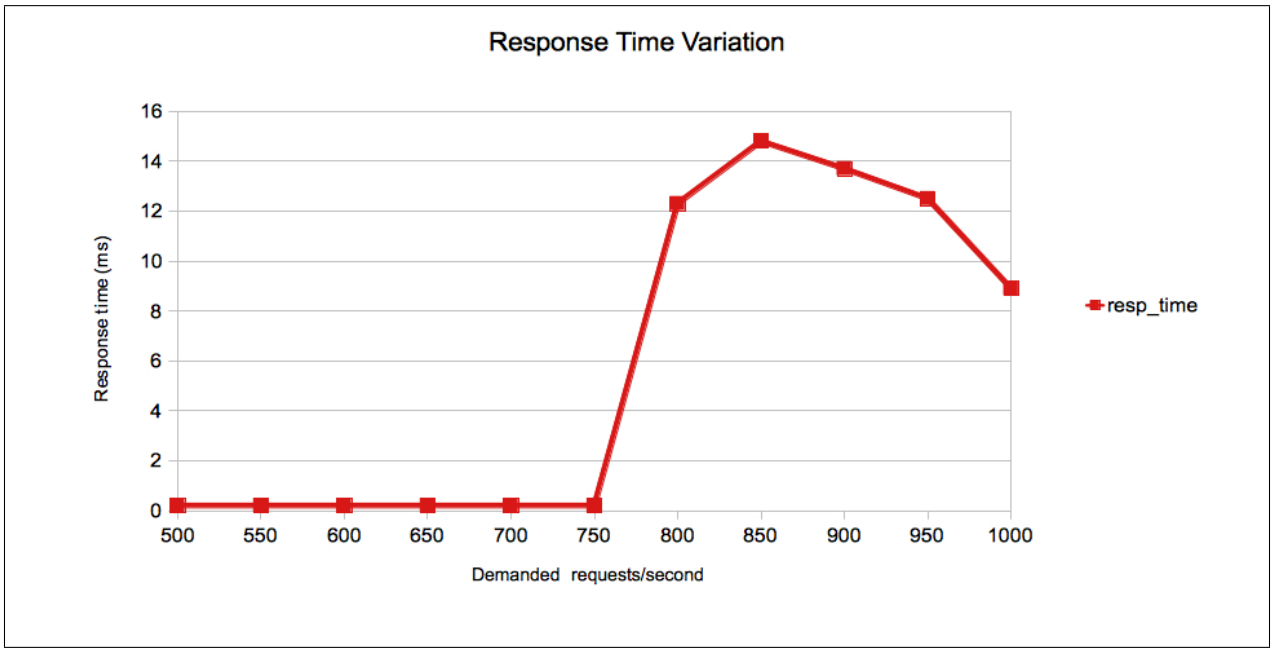

Figure 6.10 *Response Time Variation with Load on Server*

overhead of the monitoring system with respect to increasing number of resources.

6.4 Overhead of the system

One of the main goals of the monitoring system is to gather the required information with least overhead. The overhead of the sensor used to measure the Client Perceived Response Time(CPRT) is calculated in this section. In the ideal case, we do not use the kernel enhancements at the server side and measure the overhead of the justniffer application on the system, mainly in terms of its CPU and Memory consumption. In-order to measure the overhead, we flood requests to the server and use Justniffer to calculate the Client Perceived Response Time at the server side. In figure 6.12, we can observe the CPU and the memory consumption of the justniffer application. However, we should note that the average response time of the requests starts at a high value of 18ms and increases linearly upto 23ms, as we increase the load on the server from 5000 requests/s to 7000 requests/s. In-order to observe the impact of the kernel enhancement of PF_RING kernel

module on the resource consumption, we repeat the experiment after inserting the kernel module.

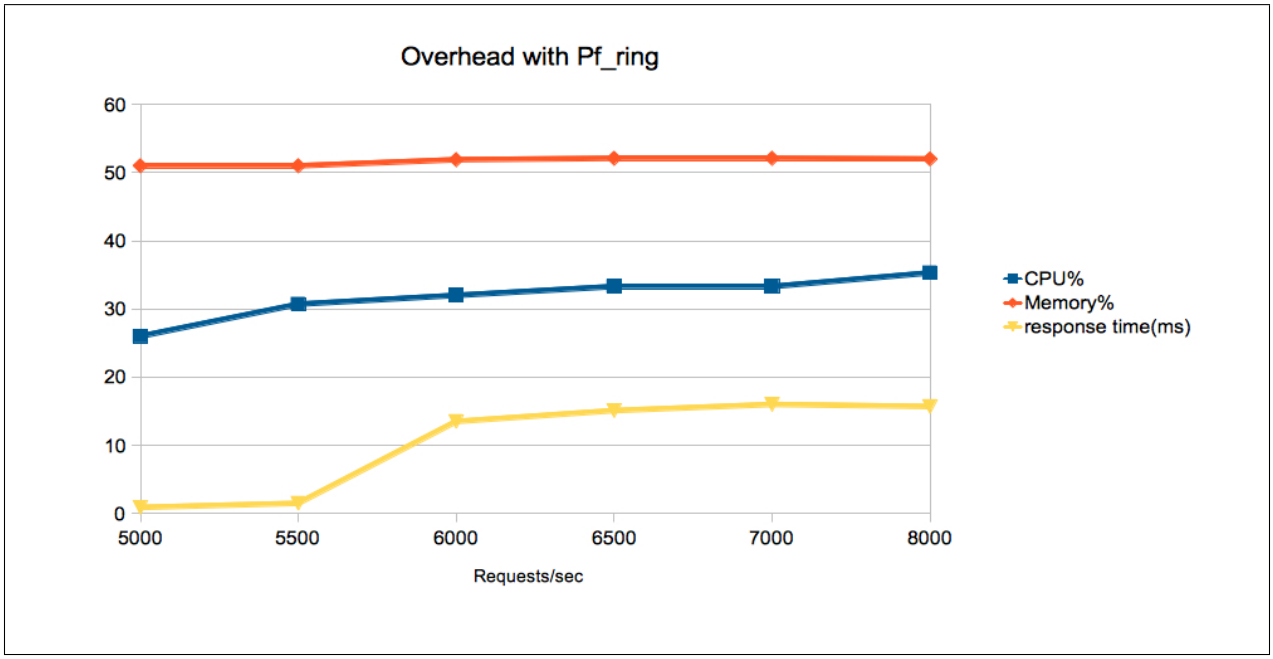

Figure 6.11 *Overhead of system with PFRING kernel module*

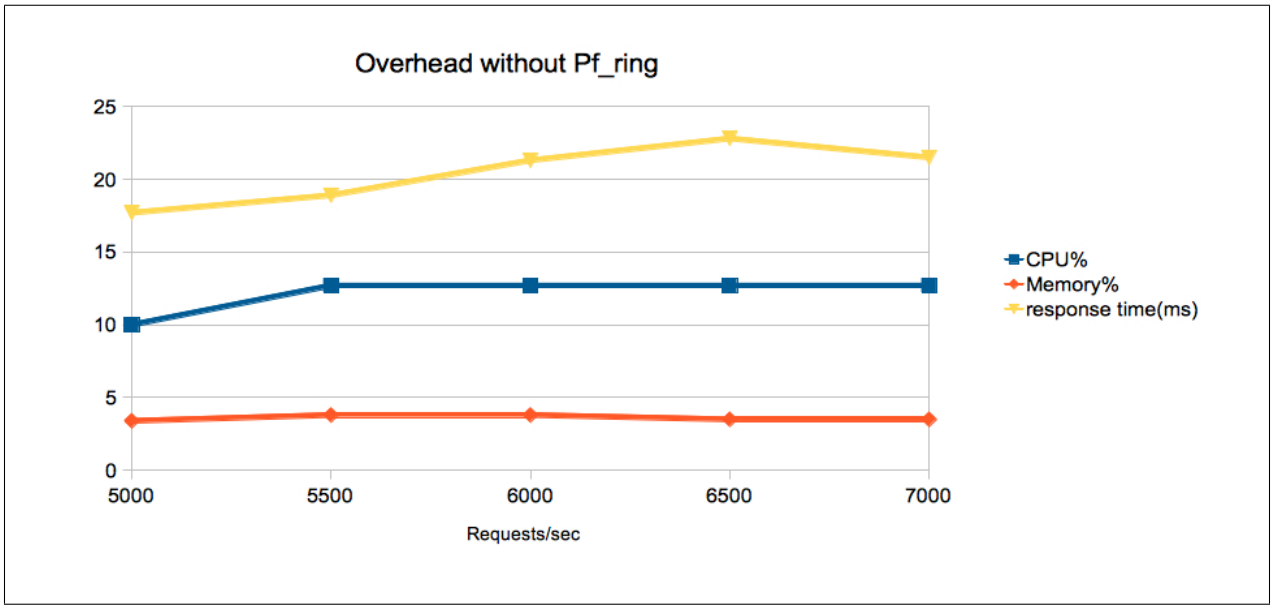

Figure 6.12 *Overhead of system without PFRING kernel module*

In figure 6.11, we clearly observe that the resource consumption is much more than in the case

without any kernel enhancement. But we observe that the response time of the web application is considerably much better compared to the case without the kernel enhancements. The average response times of the requests begins close to 1ms and increases until 15ms, while without the kernel enhancement, we had seen the average response time start from 15ms and increase higher. The reason for this lower response time is the fact that PF_RING enhances the packet capture mechanism at the kernel level thereby reducing the load on the node on which the server is running and helps the server perform better, which is reflected with a lower response time by the web server for the same number of requests flooding the server. Based on the results obtained from this experiment, we come to a point where we have a trade-off between the increased overhead to better performance of the system. In this case, it would be better to choose the better performance of the web server through enhancements in the kernel, over the overhead incurred in terms of the CPU and memory consumption.

6.5 Reduced packet loss

The libpcap packet capture library used by Justniffer does not perform well under high loads and is subjected to packet loss. We created an enhanced Justniffer using kernel module enhancements of PF_RING and Streamline. Since Tcpdump uses the same libpcap library used by Justniffer for the packet capture, the results of the packet capture provided by Tcpdump would be similar to the results we obtain from using Justniffer, thus helping us to understand the packet loss at the server using Justniffer. Autobench tool was used to progressively flood the web server with requests from the client by increasing the number of connections as well as the number of requests to the server. The command used to perform the test, executed from the client side is

autobench –single_host –host1 xen02.das3.cs.vu.nl –uri1 / –low_rate 500 –high_rate 1000 –

rate_step 50 –num_call 100 –num_conn 1000 –timeout 5 –file result.csv

We start by establishing 500 connections/second to the server and increase this progressively at the step rate of 50, until 1000 connections/second are established. Each connection comprises 100 requests. If the reply takes more than 5seconds to reach the client, it is considered as a time out and the results of the entire experiment is stored in the file result.csv.

We measured the bandwidth between the client and the server to be 0.01 GB per second(10.24MB/second) as shown in figure 6.13. This was obtained by performing tests using IPerf tool, and observing the bandwidth between the client and the server for an interval of 10 seconds.

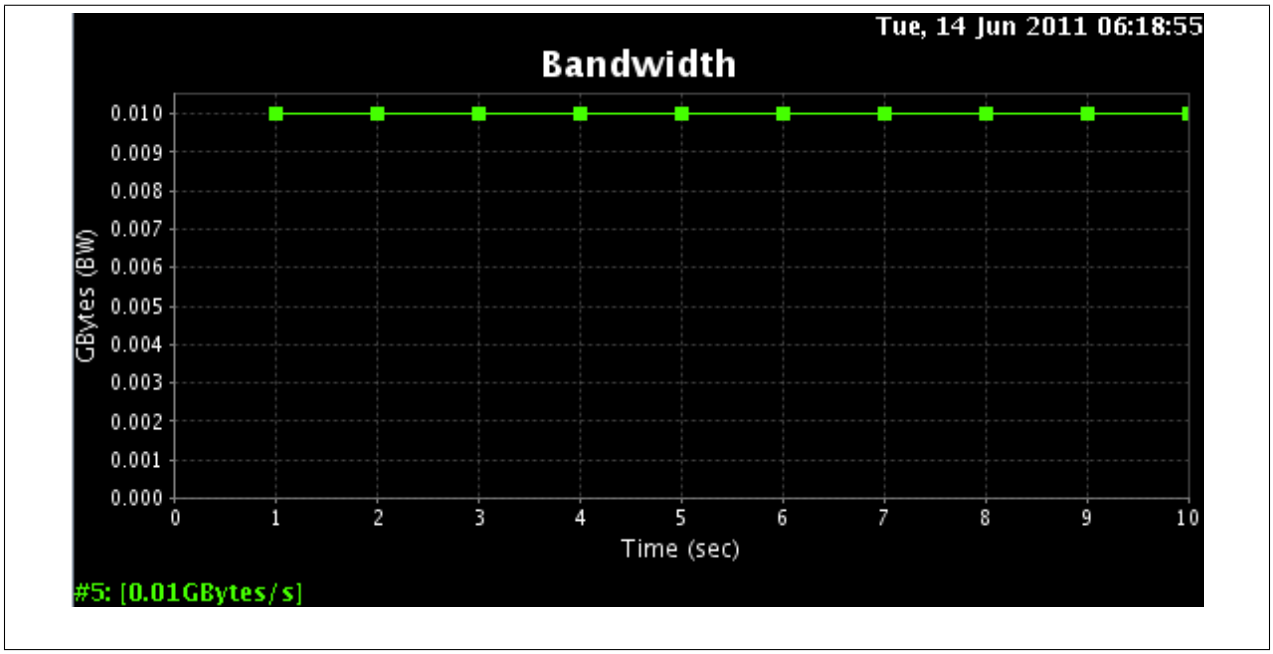

Figure 6.13 *Available Bandwidth between client and server*

For measuring the packet loss percentage, we flood the server using the autobench benchmarking tool at the client. We ran 5 experiments successively using the autobench command mentioned above, each time varying the num_conn parameter. This parameter specifies the number of con-

nections the client establishes with the server in each run. In the first run, we set this to 1000 and increased it in steps of 1000 till 5000 connections were reached in the fifth test run.

The packet loss at the server was measured using Tcpdump tool. First we measured the packet loss at the server without any kernel enhancements at two different server machines. Then we inserted the PF_RING kernel module and Streamline kernel module, at the server side and measured the packet loss in each case. The results of the experiments are plotted below in figure 6.14 and figure 6.15. The packet loss occurs due to lack of space in the kernel buffer, when there is a surge in the number of requests to the server. In the graph we can clearly observe that, with the kernel enhancements of PF_RING and Streamline, we obtain no packet loss. This is due to the fact that these kernel modules introduce lesser overhead in terms of memory utilization and context switching during the journey of the packet from the kernel driver to the user space libpcap application. This shows that Enhanced Justniffer works well for measuring the Client Perceived Response Time(CPRT) metric at the server side, even for high workloads.

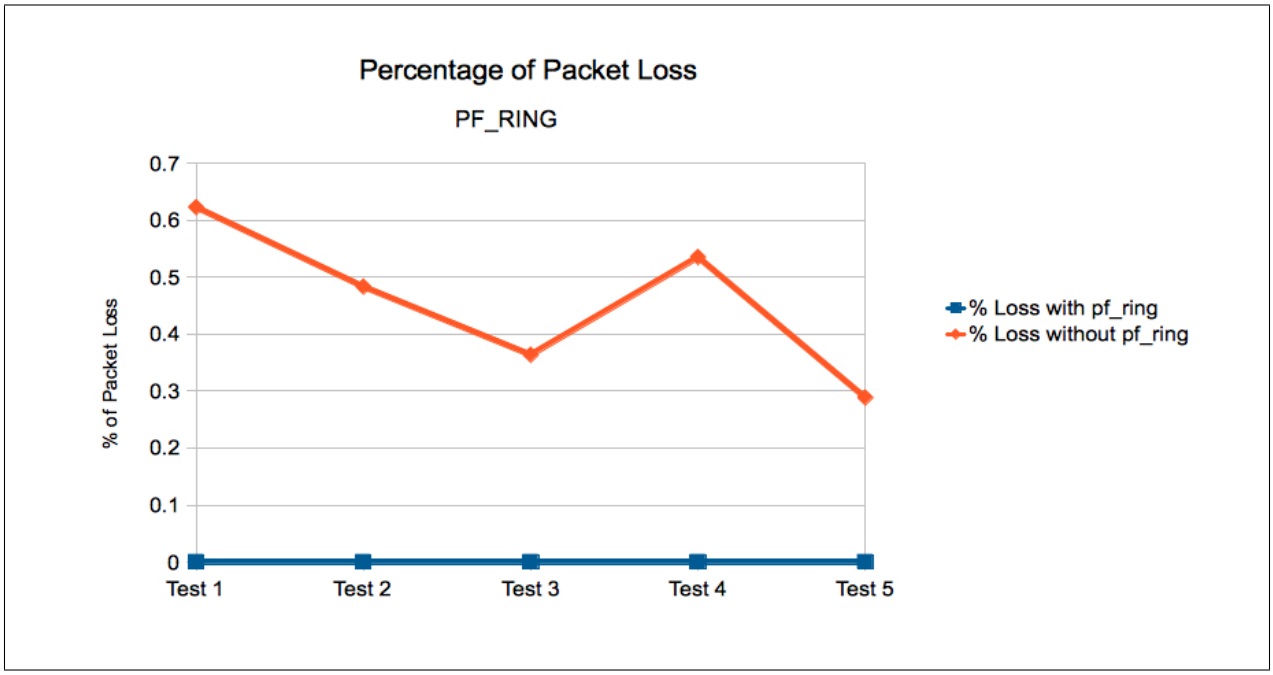

Figure 6.14 *Packet loss with PFRING enhancement*

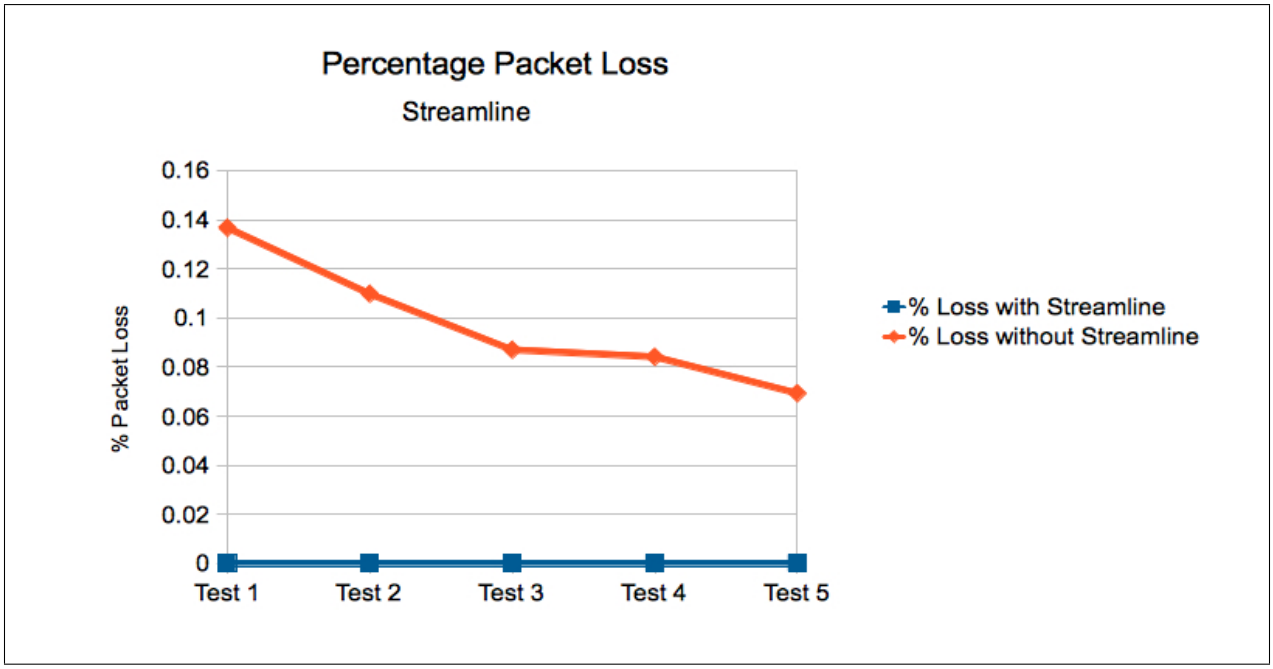

Figure 6.15 *Packet loss with Streamline enhancement*

Chapter 7

Related Work

7.1 Distributed Monitoring Systems

Apart from Ganglia, there are many other distributed monitoring systems which are also used in large scale distributed environments. We describe a few of them in detail below (25).

GlobusMDS

The Monitoring and Discovery Service(MDS) (4) constitutes the information infrastructure for the Globus (6) toolkit. MDS is based on two protocols: the Grid Information Protocol(GRIP) and the Grid Registration Protocol(GRRP). The former allows query/response interactions and search operations. GRIP is complemented by GRRP, which is for maintaining the soft-state registrations between MDS components. The MDS framework consists of information providers(sensors), Grid Resource Information Services (GRIS-producers) and Grid Information Index Information(GIIS-republishers). Producers collect events from information providers, either from a set of shell scripts of from loadable modules via an API. In addition, producers provide their events to republishers or to consumers using GRIP, and register themselves to one or more republishers using GRIP.

Republishers form a hierarchy in which each node typically aggregates the information provided by lower level republishers. Republishers use GRIP and GRRP as part of the consumer and producer interfaces. Consumers submit the queries to either producers or republishers, or discover producers through republishers, in any case using GRIP. The Light Weight Directory Access Protocol(LDAP) is adopted as a data model and representation, a query language and a transport protocol for GRIP, and as a transport protocol for GRRP. However, the performance of the OpenLDAP's update operation-which is by far the most frequently used, has been very much criticized.

MonALISA

MonALISA(Monitoring Agents using a Large Integrated Services Architecture) (14) is a Jini-based extensible monitoring framework for hosts and networks in large-scale distributed systems. It can interface with locally available monitoring and batch queueing systems through the use of appropriate modules. The collected information is locally stored and made available to higher level services, including a GUI front-end for visualizing the collected monitoring events. MonALISA is based on the Dynamic Distributed Services Architecture (DDSA) which includes one station server per site or facility within a grid, and a number of Jini lookup discovery services. The latter can join and leave dynamically, while information can be replicated among the discovery services of common groups. A station server hosts, schedules and restarts if necessary, a set of agent-based services. Each service registers to a set of discovery services wherefrom can be found from other services. The registration in Jini is lease-based, meaning that it has to be periodically renewed, and includes contact information, event types of interest and the code required to interact with a given service. A client, after discovering a service through the lookup service, downloading its code and instantiating a proxy, can submit real-time and historical queries or subscribe for events of a particular type. In addition, non-java clients can use a WSDL/SOAP binding. Although MonALISA provides a general purpose and flexible framework, it can be argued that Java restricts

the overall performance. Also Jini is using multicast, which is not always available, and places scalability limits.

Paradyn

Paradyn (13) is a performance analysis toolkit for long running, parallel, distributed and sequential applications. It supports dynamic instrumentation, that is, insertion, modification and removal of instrumentation code during program execution. Scalability concerns arising from performance analysis of applications with hundreds of processes motivated the development of Multicast/Reduction Network. MR-Net is a communication system , with support for multicast and data aggregation services, for use in parallel tools-available separately and as part of Paradyn. In-addition to Paradyn's end-user GUI, parallel processes of an application and a back-end per process, MRNet provides a program that can be run in many instances in potentially different hosts to form a custom hierarchy of internal processes that transfer data from producers to a single consumer and vice versa. Producers and consumers can use MR-Net's communication facilitates using the provided C++ library. Paradyn with MRNet is reported to have significant improved scalability with respect to a variety of performance metrics.

Relational Grid Monitoring Architecture(RGMA)

RGMA (3) combines grid monitoring and information services based on the relational model, since they believe that the difference between information and monitoring services is that the data involved in the latter have to be timestamped. Database producers are employed for static data stored in databases, whereas stream producers for dynamic data stored in memory resident circular buffers. New producers announce their relations using an SQL "create table" query, offer them via an SQL "insert" statement, and "drop" their tables when they cease to exist. A consumer is defined as an SQL "select" query. In order for a component to act as either a consumer or a producer,

it has to instantiate a remote object (agent) and invoke methods from the appropriate (consumer or producer) API. The global schema includes a core set of relations, while new relations can be dynamically created and dropped by producers as previously described. Republishers are described as one or more SQL queries that provide a relational view on data received by producers or other republishers. The registry holds the relations and views provided by database producers, stream producers and republishers. RGMA is implemented in Java Servlets and its API is available for C++ and Java, while wrapper API implementations exist for C,Perl and Python. The RGMA implementation is considered stable but suffers in terms of performance.

7.2 Client Perceived Response Time

Client Perceived Response Time has been considered as an important metric for reflecting the performance of the web applications and many efforts have been taken to obtain this metric from the server side, most of which are listed below.

A number of companies provide active probing of a web site by periodically measuring response times at a geographically distributed set of monitors. There are several limitations with this approach. First, no real web traffic by the actual clients is measured, only the response time for transactions generated by the monitors are reported. Second, any approach based on coarse-grained sampling may suffer from statistical biases. Third, monitors are limited to performing transactions that do not affect other users or modify state in backend databases.

The second approach involves instrumenting Web pages with client-side scripting that gathers client response time statistics. This approach can be used to track actual client transactions. However, client-side scripting is a post-connection approach and therefore does not account for delays due to TCP connection setup or waiting in kernel queues on the Web server, which can be significant when network and server resources are overloaded. Client-side scripting cannot be applied

to non-HTML files that cannot be instrumented, such as PDF and Postscript files. It may also not work for older browsers or browsers with scripting capabilities disabled, such as mobile devices.

The third approach requires the web server to track when requests arrive and complete service, either at the application-level or at the kernel level. This approach has the desirable properties that it only requires information available at the Web server and can be used for non-HTML content. However, application-level approaches do not account for network interactions or delays due to TCP connection setup or waiting in kernel queues on the Web server.

The fourth approach is to simply log network packets to disk, and then use the log files to reconstruct the client response time. This kind of analysis is performed offline, using multiple passes and limited to analyzing only reasonably sized log files.

Previous work done in this area such as Ksniffer (18), is implemented as a kernel module in the physical machine and computes the client perceived response time using packet capture methods in the kernel. This system has not been tested to capture the packets from virtual interfaces, which is a necessity, since virtualization is crucial part of any cloud environment. Moreover, this implementation is not open source. Thus, this calls for a better and more efficient approach for measuring the Client Perceived Response Time in an efficient and non-intrusive manner in the Cloud environment for the resource provisioning purposes.

Chapter 8

Conclusions and Future Work

We have designed, implemented and evaluated a high-precision monitoring framework for monitoring the web applications in the cloud environment. This has been implemented in a non-intrusive manner as extension python modules to the Ganglia Monitoring System. This high-precision monitoring system has been designed to monitor a variety of metrics which reflect the performance of the web application in a cloud environment, and these results will help in better provisioning of the cloud resources to the web applications, ensuring their good performance. Guarantee is given to the owner of the web application of desired performance, while simultaneously the resources can be better provisioned by the cloud providers. The evaluation results show that the developed enhanced monitoring system induces less overhead in the cloud environment. An efficient method has been devised for measuring one of the main challenging metrics, the Client Perceived Response Time. It involves using a set of loadable kernel modules for its measurement. The Client Perceived Response Time, measured at the server side ,is very similar to the actual Response Time as perceived by the user of the web application.

Future work includes identifying more new metrics which would reflect the performance of the web application that would aid in the better resource provisioning of the web applications in the cloud environment. These metrics have to be gathered in a non-intrusive manner from the nodes

in the clouds. Also with the advent of time, efficient and faster packet capturing filters will be developed and these can be used for the measurement of the Client Perceived Response Time. Since cloud computing is emerging as a widely researched area, there is a lot of scope of research in the area of monitoring of web-applications in the cloud environment.

Bibliography

[1] Amazon. Amazon ec2. *http://aws.amazon.com/ec2/*, Last updated 2011.

[2] Herbert Bos, Willem de Bruijn, Mihai Cristea, Trung Nguyen, and Georgios Portokalidis. Ffpf: Fairly fast packet filters. In *Proceedings of OSDI'04*, 2004.

[3] Andy Cooke, Alasdair J G Gray, Lisha Ma, Werner Nutt, James Magowan, Manfred Oevers, Paul Taylor, Rob Byrom, Laurence Field, Steve Hicks, Jason Leake, Manish Soni, and Antony Wilson. R-gma: An information integration system for grid monitoring. In *Proceedings of the 11th International Conference on Cooperative Information Systems*, pages 462–481, 2003.

[4] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid information services for distributed resource sharing. 2001.

[5] Mozilla Firefox. Firebug: A firefox pluggin. *http://getfirebug.com/*, 2005.

[6] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A security architecture for computational grids. 1998.

[7] Google. Google docs. *https://docs.google.com/*, Last updated 2011.

[8] Google. Googleapp engine. *http://code.google.com/appengine/*, Last updated 2011.

[9] HP Labs. Httperf: A tool for measuring web server performance. *http://www.hpl.hp.com/research/linux/httperf/*, 2008.

[10] Luis Martin. Tcpdump: A powerful command line packet analyser. *http://www.tcpdump.org/*, Last updated 2010.

[11] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 30:2004, 2003.

[12] Julian T J Midgley. Autobench: A perl wrapper around httperf. *http://www.xenoclast.org/autobench/*, Last modified 2004.

[13] Barton P. Miller, Mark D. Callaghan, Jonathan M. Cargille, Jeffrey K. Hollingsworth, R. Bruce, Irvin Karen, L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. The paradyn parallel performance measurement tools. *IEEE Computer*, 1995.

[14] Harvey B. Newman, Iosif Legrand, Philippe Galvez, Ramiro Voicu, and Catalin Cirstoiu. Monalisa : A distributed monitoring service architecture. *CoRR*, cs.DC/0306096, 2003.

[15] NLANR/DAST. Iperf: Tool to report bandwidth, delay, jitter, datagram loss. *http://sourceforge.net/projects/iperf/*, Last modified 2011.

[16] Oreste Notelli. Justniffer: A tcp packet sniffer. *http://justniffer.sourceforge.net/*, 1997.

[17] ntop.org. Pfring. *http://www.ntop.org/PF_RING.html*, 1998.

[18] David Olshefski, J.Nieh, and E.Nahum. Ksniffer: Determining the remote client perceived response time from live packet streams. In *Proceedings of OSDI'04*, 2004.

[19] Rackspace. Rackspace. *http://www.rackspace.com/*, Last updated 2011.

[20] Open Source. Curl: Command line tool for transferring data. *http://curl.haxx.se/*, Last updated 2011.

[21] Igor Sysoev. Nginx web server. *http://wiki.nginx.org/*, 2002.

[22] Stephen Turner. Analog: A web parser. *http://www.analog.cx/*, 2005.

[23] John Walker. Logtail: Watch multiple log files on multiple machines. *http://sourceforge.net/projects/logtail/*, 2002.

[24] Yahoo. Yahoo mail. *https://mail.yahoo.com*, Last updated 2011.

[25] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems (FGCS) Journal, Volume 21, Issue 1, Pages: 163-188, Elsevier Science, The*, 21:163–188, 2005.