A Quality of Service Management Testbed

Michael J. Katchabaw, Hanan L. Lutfiyya, and Michael A. Bauer
Department of Computer Science
The University of Western Ontario

Abstract

Today's computing environments are becoming more and more distributed in nature. At the same time, the applications used in these environments are becoming more complicated and are being used in more mission critical roles in the enterprise. Consequently, users' demands for performance, reliability, and availability are increasing rapidly. To meet these needs, a high level of quality of service must be delivered to the user. Doing so, however, is not an easy task.

Because of considerable research effort into this area, we are making great strides towards acceptable quality of service solutions. To facilitate continuing work in this area, we must have access to a tightly controllable, highly portable, and flexible environment suitable for quality of service experimentation. Not only is such an experimental testbed useful for gaining valuable insight into developing better quality of service solutions, but it is also required for evaluating the merit of these solutions.

To address this problem, we have developed a testbed for carrying out experimentation into quality of service management. We present our initial requirements for the testbed, its design, and a prototype implementation. We describe experimentation to date and evaluate the testbed and its effectiveness based on these preliminary results. Finally, we conclude with a summary of our work and outline our plans to evolve the testbed to accommodate further experimentation and work in the future.

1 Introduction

In today's computing environments, providing users with the level of quality of service they expect is becoming increasingly difficult. The typical environment is widely distributed, consisting of hundreds or thousands of heterogeneous computing nodes, and is growing in both size and diversity at a rapid pace. As users move to these new environments, and the enabling technologies become more readily available [7], the applications being used are becoming larger, more complex, and multimedia in nature. Environments that once only saw applications such as plain-text electronic mail and news, and remote file and terminal access are now being faced with video-on-demand, distance education, tele-medicine, tele-conferencing, and several others [6].

Increasingly, these settings are playing mission critical roles in the enterprise, and their quality of service is essential. Indeed, users are demanding levels of performance, reliability, and availability comparable to those on traditional centralized systems, where quality of service problems are smaller, better contained, more readily identified, and easier to resolve. Consequently, meeting the quality of service expectations of users is becoming a considerable challenge for today's administrators.

Concentrated research efforts into quality of service management have yielded very promising results. Higher capacity gigabit networking technologies such as Asynchronous Transfer Mode (ATM) [17] over fiber-optic transports have alleviated network performance and reliability problems somewhat. Work focusing on network-level quality of service management has provided further solutions by developing novel approaches to resource allocation and reservation [3, 4, 13, 21], call admission [9, 18], dynamic reconfiguration around saturated or failed links [16], and so on. To help users deal with performance, reliability, and availability problems that occur on the end-systems in distributed environments, there is also a growing effort investigating end-system and operating system quality of service [1, 2, 5, 8, 14, 20]. Despite the success of this work, much of the research in this area is preliminary and there are still many open problems needing to be addressed.

To support further research into quality of service management, we must have access to a tightly controllable, highly portable, and flexible environment suitable for experimentation in this area. Such an experimental testbed, in conjunction with well developed modeling and simulation techniques, would be useful for gaining valuable insight into developing better quality of service solutions. To date, there has been significant work in both modeling and simulation to support the development of quality of service solutions, as evident in the citations above. Experimentation is as valued as these approaches for such analytical and development purposes [10], yet much work is still needed to support quality of service experimentation.

In addition to helping guide the development of quality of service management solutions, experimentation is also required for evaluating the effectiveness of these solutions. Without such experimentation, the solutions we develop may be limited to a single platform or situation, not scale to production environments, or fail outright when faced with either realistic or stressful operating conditions. As a result, an ex-

Appeared in the Proceedings of the Third International Workshop on Systems Management,
Newport, Rhode Island, April 1998

perimental quality of service management testbed is both desirable and necessary.

Our current work focuses on the development of a quality of service management testbed to meet these needs. The testbed allows us to design, execute, monitor, and analyse experiments investigating quality of service management, with the potential of doing so at greatly reduced costs. The testbed provides a closed environment consisting of a variety of hardware and software components which can be subjected to programmable and consistent workloads by the testbed. By executing distributed multimedia applications within this testbed, we can measure their quality of service under a variety of operating conditions. By also executing quality of service solutions within this testbed, we can measure the effectiveness of these solutions. Through the use of this testbed, we have been able to carry out both experiments that give insight into the quality of service provided by numerous applications in various scenarios, and experiments that evaluate the capabilities of quality of service solutions.

In this paper, we present a set of requirements for the quality of service management testbed. We outline a testbed design that meets these requirements, and the testbed prototype based on this design. We then describe a sample of results obtained from our experiments to date that demonstrate the suitability of the testbed to its task. The paper concludes with a project status report and our plans for future work.

2 Requirements

The goal was to develop an experimental testbed that can be used to investigate quality of service management and evaluate quality of service solutions. The experiments of interest involve deploying a variety of user application components (typically some kind of distributed multimedia application) and quality of service solution components throughout a distributed environment. The distributed environment is then subjected to a variety of loads and the quality of service being delivered by the application components in the environment is measured and analysed. We identified the following key requirements for the testbed to support this kind of experimentation:

- The testbed should allow an experiment to be controlled. This includes the ability to start and stop an experiment, to activate and deactivate components of the distributed environment (such as computing nodes, disks, network links, and so on), to generate loads, and to launch and terminate components of applications (including both user applications and quality of service solutions) within the environment.

- The testbed should be isolated so that experiments are not affected by external influences and to permit tighter control over experiments.

- The testbed should be flexible and support a range of experimental parameters, allowing reconfiguration for a variety of experiments. It must maintain configuration information about the distributed environment and its applications.

- The testbed should provide a way to collect quality of service metrics and behavioural data on the distributed environment and its applications.

- Simple statistical analysis of collected data should be provided by the testbed.

- Higher-level applications should be able to use the testbed through a well-defined application programming interface (API).

- The testbed itself should have minimal interference with the performance of the distributed environment and its applications. The testbed must operate outside the execution path of the system being tested; it must not require intrusive instrumentation or code modification. All information must be gathered at arm's length through the use of operating system services and existing observable behaviour.

- To as great a degree as possible, the testbed design should be independent of the system being tested. It should not make assumptions about the type of distributed environment or the applications in use.

3 Testbed Overview

We have developed a generic architecture to meet the requirements outlined in Section 2. In this section, we provide an overview of the key components of the architecture and the testbed prototype we have developed. This testbed was adapted and extended from a previous experimental testbed; further details can be found in [11].

3.1 Architecture

The quality of service management testbed architecture shown in Figure 1 is the basis for our research and prototype development. This architecture consists of the testbed itself, a distributed environment containing load generators, environment components, and application processes whose quality of service is to be analysed, and testbed applications that make use of the testbed. These components are described below.

3.1.1 Testbed Components

The testbed consists of a set of core components.

Testbed Service Interface: The services provided by the testbed are accessed by testbed applications through the testbed service interface. This interface coordinates activities within the testbed by routing requests, replies, and reports to the appropriate components. Samples of services provided through this interface have been discussed in detail in [11].

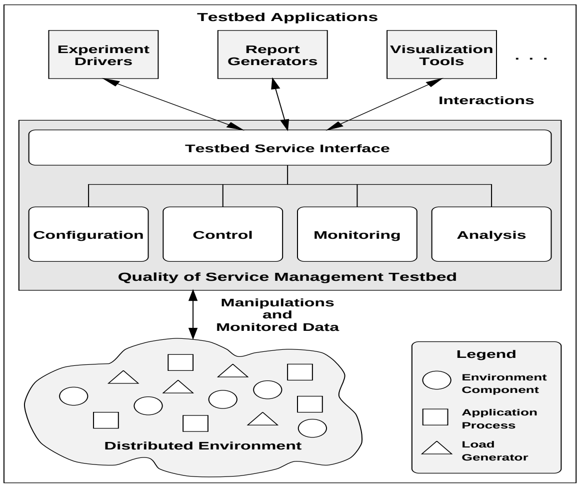

Figure 1: Quality of Service Management Testbed Architecture

Configuration: The configuration component is responsible for maintaining information on the configuration of the distributed environment and applications executing in it. Areas of responsibility include:

- Deployment — where environment components, application processes, and load generators are located in the distributed environment.
- Relationships — how environment components, application processes, and load generators are related to one another.
- Workload population — what kind of interactions occur between environment components, application processes, and load generators, what information is exchanged in each interaction, and how often these interactions occur.

Control: The control component provides a basic service for controlling experiments. It uses deployment and relationship information from the configuration component and provides operations to:

- Start an experiment and allocate the resources required, based on the defined configuration.
- Finish an experiment and deallocate experiment resources.
- Activate environment components according to their configuration.
- Deactivate environment components.
- Launch application processes and load generators according to their configuration.
- Terminate application processes and load generators.

Monitoring: The monitoring component collects the following information about environment components, application processes, and load generators in the distributed environment:

- Operational status information.
- Timing information, including responsiveness, throughput, and so on.
- Utilization statistics on how each environment component is used.

Analysis: Using configuration information and data collected from experiments, the analysis component can compute:

- Basic measures, such as totals, maximum values, minimum values, means, medians, and modes.
- Variability measures, such as standard deviation, sample variance, and skewness.

These functions can be applied to individual environment components or groups of components. More sophisticated analyses, such as regression analysis and trend computations, can be performed by higher level testbed applications.

3.1.2 Distributed Environment

The distributed environment shown in Figure 1 contains a variety of environment components, application processes, and load generators. Environment components consist of hardware and software resources and services used by application processes and load generators. This includes computing nodes, networks, operating system services, and distributed processing services, such as name services, repository services, and transaction services. Application processes are programs executing in the distributed environment that are either providing services to end users or quality of service solutions to these user applications. We are interested in measuring the level of quality of service provided to end users by user applications in the presence (or absence) of the quality of service solutions. Load generators are tools, both hardware and software, used to induce some kind of a load in the environment to utilize a variety of resources, such as CPU cycles, memory and buffer space, network bandwidth, disk throughput, video display bandwidth, and so on.

By interacting with the distributed environment components rather than application processes themselves, the testbed can configure, control, and monitor the quality of service provided by applications with minimal impact on their operation. For example, resource utilization statistics can be obtained through operating system services instead of querying individual processes, which would add unwanted load to the processes.

3.1.3 Testbed Applications

Users access the services of the testbed through external testbed applications. These applications invoke the testbed operations through the testbed's service interface. Examples of such applications, are shown in Figure 1, and explained below.

Experiment Drivers: Carrying out an experiment using the testbed involves selecting the desired configuration, executing the experiments, and analysing the collected results. Even the simplest analysis requires several experiments with different configurations to be repeated over extended periods of time. Repeating the experiments manually would be extremely tedious. An experiment driver facilitates and automates this process.

Report Generators: Report generation applications can be used to aggregate the data from experiments, perform high-level analysis, and present the results.

Visualization Tools: Experiments can be configured and controlled more intuitively by manipulating visual abstractions of the components involved. Data can be presented more concisely and understandably in graphical forms.

4 Experiment Quality of Service Metrics

To measure the level of quality of service provided by user applications in the testbed, we must use metrics to capture, measure, and compare the levels of quality provided. By comparing the quality provided under a variety of operating conditions, we can deduce the resource needs and preferences of the application. This in turn can be used to develop better resource allocations to maintain quality of service in various usage scenarios, and to dynamically refine these allocations when the scenarios change. By measuring the quality of service provided by an application in the presence of a quality of service solution, we can evaluate the effectiveness of the solution. By further varying the loads applied by the testbed during experimentation, we can evaluate the robustness of the solution.

There are several potential ways of measuring quality of service. Using the working definition of quality of service found in [19], "the set of those technical and other parameters of a distributed multimedia system, which influence the presentation of multimedia data to the user, and in general the user's general satisfaction with the application," we can see that there are both qualitative and quantitative aspects to measuring quality of service. Since assessing "general satisfaction" and other qualitative measures in a statistically reliable way tends to be quite difficult to do in an experimental environment, we are focusing in our current work on more quantitative measures. Consequently, we concentrate on a set of metrics which we classify as responsiveness metrics, productivity metrics, utilization metrics, and error or failure metrics [10].

4.1 Responsiveness Metrics

The responsiveness of an application is the time for an application to complete units of work in the process of providing its service. In a request-oriented application, each operation performed would be a unit of work. In a streams-oriented multimedia application, the delivery of each frame of multimedia information would also be considered to be a unit of work. Examples of responsiveness metrics include:

- The response time for a single application request.
- The start-to-completion time for a workload (that is, a "standard" batch of application requests).
- The delay between frames of multimedia information such as video and audio frames.
- Process initialization time, measured from the time the process starts execution until it is ready to perform useful work.

4.2 Productivity Metrics

Productivity (or throughput) metrics provide measures of the amount of work completed per unit of time. These metrics relate both to the rate at which application work is performed and to the amount of data passed between application components within a given period of time. Examples of productivity metrics include:

- For a server process, the number of application requests processed per second.
- The number of batches of application requests processed per second.
- The amount of data transferred between processes per second.

4.3 Utilization Metrics

Utilization metrics provide measures of environment resource consumption and, as such, are not direct measures of an application's quality of service. However, utilization metrics may produce predictive information about where quality of service problems may occur in the future, when there is a higher contention for resources. For example, if an application is found to be memory intensive, we can predict that the application's performance would suffer when memory becomes scarce, as it must wait for memory to become available, or when the system is highly utilized, as the operating system may not be able to service memory management requests as quickly. Furthermore, if an application is found to consume too many resources, it may impact the quality of other applications. Consequently, these metrics can still provide interesting information. Examples of these metrics include: CPU utilization, memory utilization, disk utilization, I/O utilization and network utilization.

Measurements of these resources can be collected either for individual services, or over the lifespan of a process. Depending on the analysis required, both measurements can be useful.

4.4 Error or Failure Metrics

An error is defined as a situation in which an application unit of work cannot be completed correctly. A failure is defined as a situation in which a unit of work cannot be completed at all. While errors and failures tend to be application-specific, they must be understood in order to completely assess the level of quality provided by an application. For each application-specific class of error or failure, we can measure metrics such as:

- The probability of an error or failure occurring for a given unit of work.
- The rate at which errors or failures occur.
- The mean time between errors or failures.
- The "down-time" during which the application is unavailable due to failure.
- The severity of the error or failure.

5 Experiment Parameters

The quality of service metrics presented in the previous section identified what we want to know about the level of quality of service provided by applications in the testbed. These are the observable outputs of experiments. We now focus on the parameters that influence this quality of service. These parameters correspond to the inputs to experiments.

Identifying quality of service parameters is critical to the planning, design, and execution of experiments. These parameters generally fall into one of two categories—system parameters or workload parameters.

5.1 System Parameters

System parameters are those related to hardware, the operating system, middleware services, and so on. They are outside of applications being monitored and from an experimentation point of view, they are considered to be relatively static. These parameters include:

System Level Parameters:

- Number of systems involved
- Architecture of each system
- Speed of each system's processor(s)
- Speed of each system's coprocessors
- Amount and speed of system memory
- Speed of disks on each system
- Use of hardware caches on each system

Network Level Parameters:

- Speed of the network
- Network reliability
- Network protocols used
- Network topology/configuration (subnets, routers, bridges, etc.)

Operating System and Middleware Parameters:

- Operating system overhead
- Effects of software caches
- Middleware overhead
- Nature/speed of other services (file, time, name, etc.)

5.2 Workload Parameters

Workload parameters represent the variable aspects in the experiments; factors that can be controlled or can be varied from one experiment to the next. We include characteristics of the applications to study as well as aspects of system state that are affected by the workload imposed by the applications and any load generators in use. These parameters include:

Application Parameters:

- Application type
- Process types (client, server, peer)
- Resource requirements
- Communication methods
- Configuration (number of processes, locations, etc.)
- Request types (including rates, patterns, and distributions)

Load Parameters:

- Number of users, number/type of processes, etc.
- CPU utilization
- Memory usage
- Network utilization
- Disk usage
- Video display utilization
- Operating system and middleware service utilization

6 Prototype Implementation

In this section, we describe a prototype testbed environment based on the generic architecture presented in Section 3.1. This prototype is currently being used to conduct a variety of quality of service management experiments.

6.1 Target Environment

The prototype was developed in a laboratory setting with a dozen hosts of various architectures, including IBM RS/6000, MIPS RC2030, Sun UltraSparc, Sun SPARCstation, Sun 3/60, and Sequent Symmetry, spread over several subnets, both Ethernet and OC-3 ATM. This setting provides diversity in processors, memory, device configuration, flavours of the UNIX operating system (AIX, RISC/os, Solaris, SunOS, and DYNIX/ptx), communication infrastructure and middleware (sockets, ONC RPC, DCE, and CORBA).

6.2 Testbed and Testbed Services

To meet the flexibility and portability requirements demanded by the heterogeneity of the environment, we implemented the testbed as a structured collection of UNIX shell scripts. Since the testbed operates outside of the application processes whose quality of service we are assessing, most of its interactions are with the operating system or by recording already available application behaviour. The ease of access to system services and utilities, and the fact that the performance of the testbed is of little concern, made shell scripts an obvious choice.

The testbed architecture was preserved by structuring each service as a collection of scripts providing the required operations. The interface to the testbed is defined by script names and their arguments. Testbed applications that invoke these scripts can be written in a variety of languages (for example, shell scripts, C, Tcl, Perl, or Java).

The configuration service stores configuration information as a set of flat files. This information is manipulated when the appropriate configuration operations are invoked. Workloads are stored as either data files or scripts, depending on the type of component for which a workload is generated. Control service operations access configuration files and workload scripts. For example, to start an experiment, this service needs to know which processes to start, where to start them, and what workloads to assign. The monitoring service operation collects specified data querying local and remote systems using UNIX system utilities. This data is stored in monitored data files. Finally, analysis service operations are carried out by scripts that process monitored data.

6.3 Measurable Applications

For initial testbed validation and experimentation, we have collected and written several applications whose quality of service can be measured. By using these applications in the testbed and applying load to the environment, we can determine the level of quality service provided in a variety of operating conditions. Measurements can be taken from all applications without the addition of management code, either through querying the operating system or observing and recording the behaviour of the application using methods already part of the application.

We have incorporated a variety of monomedia and multimedia applications into the testbed. This allows for a wide range of experiments, as each application has different resource requirements and different quality of service needs.

The monomedia applications that are available in the testbed are largely text-based and tend not to be diverse or demanding in their use of resources. They include a network data transferring program, a file manipulation program, a file transfer application, a matrix multiplication program, a scheduler application, a database application, and several others.

The multimedia applications included in the testbed incorporate several presentation methods, including text, graphics, audio, and video. As a result, they tend to use a wide range of resources, and can be very resource intensive. They include relatively simple applications for capturing, editing, and presenting graphics, audio, and video, as well as more complex applications such as a video conferencing package, a video-on-demand system, and several networked video games.

6.4 Quality of Service Solutions

To validate the testbed's effectiveness for evaluating quality of service solutions, we have attempted to gather as many solutions as possible. To date, we only have a limited number of such solutions at our disposal due to limited general availability. The solutions we have access to tend to be quite focused and concentrate on a particular aspect of the general quality of service management problem, such as reserving or scheduling for a particular resource in a distributed environment.

We are continuing to develop our own solutions and are actively seeking additional solutions from other leading researchers in this area. Nevertheless, we have enough for initial experimentation purposes.

6.5 Load Generators

To be able to apply loads to distributed environment components for experimentation, several load generators have been developed. By generating a load using these programs, we can provide a variety of operating conditions under which measurable applications and quality of service solutions can execute in the testbed.

Using these load generator programs, we can induce loads on processors, memory, network links, disks, video systems, and other resources. These programs can be configured to allow for different loads to be generated in a variety of user-definable patterns. These programs can be used individually or in concert with one another to allow for a wide range of loads to be generated in the distributed environment. For example, using our CPU load generator, we can produce constant loads ranging from very light to incredibly heavy, random loads that vary uniformly between arbitrary bounds, and random loads that vary according to a Poisson distribution, for more realistic scenarios[10].

6.6 Testbed Applications

For initial testing and experimentation, a basic experiment driver and a simple report generator were constructed. Tools have also been built to convert experiment results into spreadsheets and graphs for simple visualization. More powerful visualization applications are under development. The experiment driver uses scripts to guide the execution of several experiments in sequence. To complete each phase of an experiment, the driver makes use of the appropriate service scripts provided by the testbed. Each experiment consists of eight phases:

1. Starting the experiment and gaining sole access to the testbed.
2. Configuring the testbed, environment components, application processes, quality of service solutions, and load generators, depending on the goals of the experiment.
3. Starting the environment components, application processes, solutions, and load generators used in the experiment.
4. Monitoring the started application processes and collecting the required information.
5. Terminating the environment components, application processes, solutions, and load generators and collecting the state information needed.
6. Analysing the collected data as necessary.
7. Finishing the experiment.
8. Generating reports summarizing the results of the experiment.

7 Experimental Results and Experience

An initial set of experiments was conducted to verify the operation of the testbed and identify areas for improvement. Based on the success of these tests, more rigorous experiments were planned and executed. The results of these experiments proved to be quite interesting, demonstrating the utility and flexibility of the testbed.

While several dozens of experiments have been carried out to date, we will focus in this section on two sets of experiments illustrating the usefulness of the testbed. The first set dealt with analysing the impact of different operating conditions on the quality of service provided by a variety of applications, which led to the development of a quality of service scheduler in our laboratory environment. The second set of experiments discussed in this section dealt with evaluating the effectiveness of this quality of service solution.

7.1 Quality of Service Impact Analysis Experiments

In these experiments, we varied the operating conditions of several applications to analyse the effects on the quality of service provided. This was done by configuring the testbed to apply a variety of loads to the distributed environment in which the applications executed, and measuring the results. From these experiments, one can gain insight into the resource needs of the applications and the resource allocations and dynamic adjustments required to meet the demands of quality of service policies. In this section, we examine two such experiments.

In the first experiment, we investigate the impact of applied processor and network load on the quality of service provided by a video-on-demand application. This application consists of a server process sending video streams to clients on remote hosts connecting to the server. The video streams in this case were pre-recorded 320×240 8-bit video streams without audio in an MPEG-1 format. The application was executed in our testbed environment with the video-on-demand server on an UltraSparc serving video over the OC-3 ATM network to a single client on a Sparc-5 workstation. Four sets of replicated experiments were conducted: one with no additional load whatsoever, one with the network load being maximally utilized by a network load generator, one with the CPU load raised to a level of 10.0 on both hosts by a CPU load generator (meaning that there was a mean run-queue length of 10.0 on both machines), and the last with both the network load and CPU load being raised as above.

Figure 2 shows the average throughput under the four configurations described above. Linear regression of these results indicates that CPU load was the most significant factor, accounting for almost 77% of the variation in quality of service provided. Network load alone accounted for only 10% of the variation, while the combined effect of both loads accounted for only 12%. (Any remaining variation is caused by errors or unknown factors.) As a result, while both network and CPU load impact the quality of service provided by this application, CPU load has a far larger impact.

In the second experiment discussed here, we again study the impact of applying both processor load and network load; this time on the quality of service provided by a World Wide Web system transferring files between two hosts. The files used were 10 megabyte binary files, and were transferred using the FTP protocol. The server providing the files was located on a Sparc-5 workstation in our testbed environment, and served the files across the OC-3 ATM network to a single client on an UltraSparc. For this experiment, the transferred files were stored on disk once received,

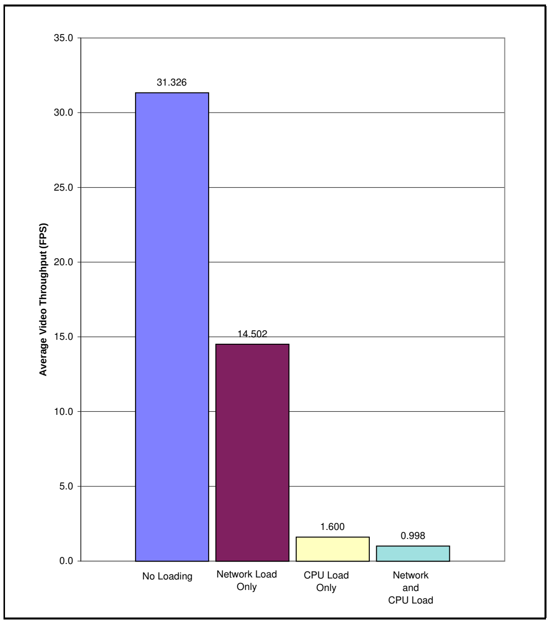

Figure 2: Video-on-Demand Experiment Results

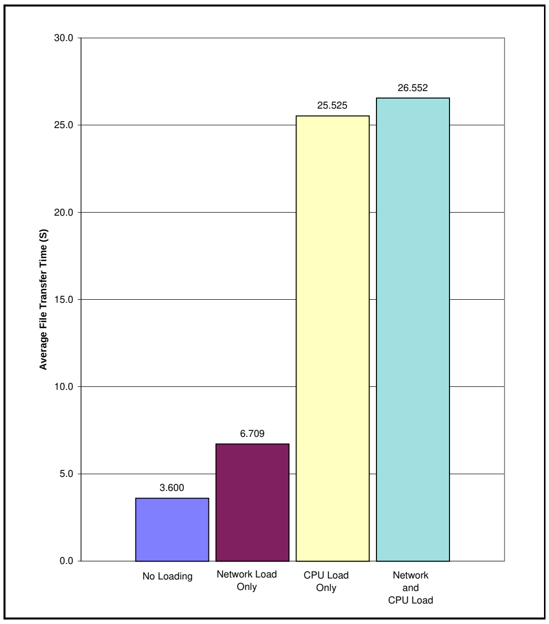

Figure 3: File Transfer Experiment Results

and not processed. Four sets of replicated experiments were conducted: one with no additional load whatsoever, one with the network load being maximally utilized by a network load generator, one with the CPU load raised to a level of 10.0 on both hosts by a CPU load generator, and the last with both the network load and CPU load being raised as above.

Figure 3 shows the average transfer time under the four configurations described above. Linear regression of these results indicates that CPU load was the most significant factor, accounting for 98% of the variation in performance. Network load alone accounted for less than 1% of the variation, as did the combined effect of both loads. (Any remaining variation is caused by errors or unknown factors.) As a result, while both network and CPU load impact the quality of service provided by this application, CPU load has a far larger impact.

This is a surprising result. In the previous experiment, one could explain the effects of a high CPU load on the video-on-demand application easily, since the decoding and displaying of real-time video could require considerable processor capacity. In this experiment, no processing or presentation of the transferred data was done, yet a reduced number of CPU cycles available still caused a more dramatic drop in quality of service. This leads one to the conclusion that the application simply was not receiving sufficient CPU cycles to sustain a high transfer rate, leading to the loss in quality of service. This was also likely the case in video-on-demand experiment.

From this we can see that for these networked applications, we must make sure that we allocate enough CPU resources, in addition to network bandwidth, to meet our quality of service policies. Furthermore, if quality of service begins to degrade, the above results indicate that any corrective actions must check for processor contention and investigate the allocation of additional CPU resources, in addition to, and perhaps in preference over, network analysis. By continuing this type of experimentation, making use of additional applications in other scenarios, further insight into application quality of service can be obtained. The experimental testbed could easily facilitate such experimentation.

7.2 Quality of Service Solution Evaluation Experiments

In these experiments, we varied the operating conditions under which a quality of service solution attempted to maintain the quality of service provided by various applications. This was done by configuring the testbed to apply a variety of loads to the distributed environment in which the solution and applications executed, and measuring the results. From these experiments, one can determine the effectiveness and robustness of the quality of service solution. In this section, we examine one such experimental evaluation.

In this evaluation, we studied a quality of service scheduler developed in our own laboratory environment [12]. This solution was largely derived from experience using the testbed and previous experimentation, including the experiments outlined in Section 7.1. This scheduler dynamically adjusts the processor scheduling of instrumented Solaris processes according to the quality of service policies for the application. Since this scheduler at this point is only concerned processor allocation on a single host, a standard MPEG-1 video player was instrumented and given a workload of the same 320×240 8-bit video streams used in previous experiments. The quality

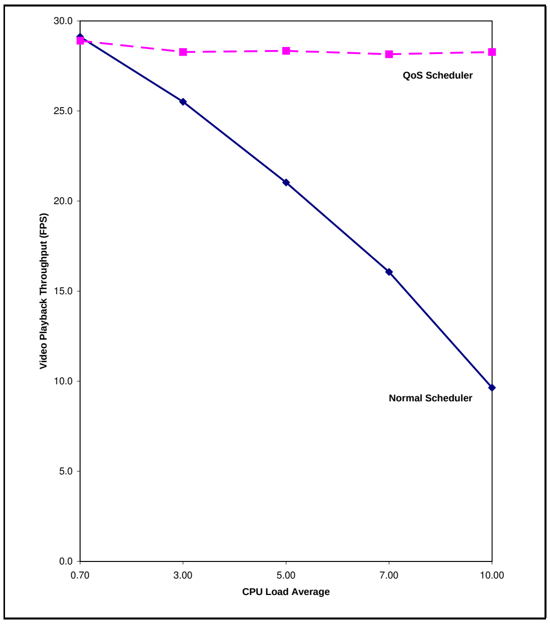

Figure 4: Video Playback Throughput Comparison

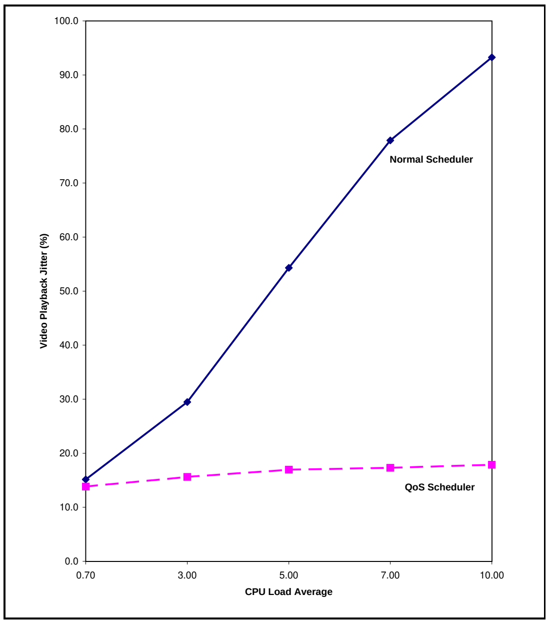

Figure 5: Video Playback Jitter Comparison

of service policy to adhere to dictated a target video playback rate of 27 frames per second, allowing for a deviation of 2 frames per second from this target (giving a range of 25 to 29 frames per second). This application, along with the quality of service scheduler were executed on an UltraSparc host in our testbed environment. Experimentation consisted of executing the instrumented video player several times under various processor loads generated by a CPU load generator. These experiments were repeated in the presence and absence of the quality of service scheduler for comparison purposes, and replicated five times for statistical confidence.

Figure 4 compares the video playback throughput, in frames per second, for the video application under normal scheduling and with the quality of service scheduler in place. From this figure, we can see that video throughput dropped dramatically under an increasing CPU load when normal Solaris scheduling was used. With the quality of service scheduler in place, however, throughput remained reasonably consistent around 28 frames per second – well within the acceptable limits set by the quality of service policy for the video application. Consequently, the quality of service scheduler improves the quality of service provided substantially over normal Solaris scheduling.

As indicated in [15], the variance of the playback rate, also known as jitter, can have a more significant impact on perceived quality of service than merely the rate alone. Using the testbed, another analysis was carried out using the same approach to experimentation; this time, measuring the jitter in the time required to display each frame of video. Figure 5 shows the jitter measured, both with and without the quality of service scheduler in place. For these experiments, jitter was computed as the ratio between the mean variation in display time and the mean display time, to normalize results. As can be seen from Figure 5, jitter increased substantially as load increased when the normal Solaris scheduler was being used. With the quality of service scheduler in place, the increase in jitter was marginal.

From this evaluation, it appears that the quality of service scheduler is reasonably effective at improving the quality of service provided by applications making use of it. Naturally, more extensive experimentation is needed to verify this preliminary finding. This experimentation, however, can be easily accommodated using the testbed. As a result, we can conclude that the quality of service management testbed is well suited to its task.

8 Concluding Remarks

The work we describe here is part of an ongoing structured attack on the quality of service management problem. There are many complex issues yet to be resolved. We believe that by developing an experimental testbed environment, we can facilitate investigations and evaluations that contribute to the development and acceptance of quality of service management solutions.

The research detailed in this paper focuses on the development of a quality of service management testbed to enable experimentation in several areas of quality of service, including analysing the quality of service provided by distributed multimedia applications in a variety of operating conditions, and evaluating the effectiveness of quality of service solutions. We first presented a core set of requirements for the testbed, and then we described a design satisfying these requirements. Using a prototype testbed based on this design, we have been able to carry out a number of experiments and have realized several useful results, thereby demonstrating the utility of the testbed.

We plan to continue experimentation to validate the results discussed in this paper and to generate new results and insights. We need to expand the set of applications, quality of service solutions, and load generators available for experiments, and make full use of the heterogeneity of the testbed environment. More advanced testbed applications are needed for better reporting and visualization. Since, in the end, it is the user's perception of quality of service that matters most, techniques for reliably measuring some of the more qualitative quality of service metrics must be examined. We also need to compare empirical testbed results to those available from modeling or simulation to determine the soundness of both our results and the models and simulation techniques being used, and to make suitable refinements where necessary.

Through this work, we hope to gain insight into the quality of service management problem. With this experience, we can further pursue solutions that effectively and completely address the variety of quality of service problems that occur in today's distributed computing environments.

References

[1] A. Campbell. *A Quality of Service Architecture.* PhD thesis, Lancaster University, January 1996.

[2] P. Druschel. *Operating System Support for High-Speed Networking.* PhD thesis, The University of Arizona, August 1994.

[3] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic Models for Allocating Resources in Computer Systems. *Market Based Control of Distributed Systems*, Editor Scott Clearwater, World Scientific Press, 1996.

[4] D. Ferrari, A. Banerjea, and H. Zhang. Network Support for Multimedia. *Technical Report TR-92-072, The International Computer Science Institute, Berkeley, California*, November 1992.

[5] S. Fischer, A. Hafid, G. V. Bochmann, and H. De Meer. Cooperative QoS Management for Multimedia Systems. *Proceedings of the Fourth IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Canada, June 1997.

[6] N. D. Georganas. Multimedia Applications Development: Experiences. *Journal of Multimedia Tools and Applications, Volume 4, Number 3*, May 1997.

[7] N. D. Georganas. Multimedia: Enabling Technologies and Applications. *Proceedings of SIMTEL'96*, Vina del Mar, Chile, May 1996.

[8] R. Gopalakrishnan and G. Parulkar. Efficient Quality of Service Support in Multimedia Computer Operating Systems. *Technical Report WUCS-TM-94-04, Department of Computer Science, Washington University*, August 1994.

[9] A. Gupta. *Multi-party Real-time Communication in Computer Networks.* PhD thesis, University of California Berkeley, February 1996.

[10] R. Jain. *The Art of Computer Systems Performance Analysis.* Wiley Professional Computing, 1991.

[11] M. J. Katchabaw, S. L. Howard, A. D. Marshall, and M. A. Bauer. Evaluating the Costs of Management: A Distributed Applications Management Testbed. *Proceedings of the 1996 Centre for Advanced Studies Confererence*, Toronto, Canada, November 1996.

[12] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. A Dynamic Quality of Service Scheduler. *Technical Report (In Preparation), Department of Computer Science, The University of Western Ontario*, 1998.

[13] A. A. Lazar and G. Pacifici. Control of Resources in Broadband Networks with Quality of Service Guarantees. *IEEE Communications Magazine, Volume 29, Number 9*, September 1991.

[14] K. Nahrstedt. *An Architecture for End-to-End Quality of Service Provision and its Experimental Validation.* PhD thesis, University of Pennsylvania, July 1995.

[15] J. Nieh, J. G. Hanko, J. D. Northcut, and G. A. Wall. SVR4UNIX Scheduler Unacceptable for Multimedia Applications. *Proceedings of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, Lancaster, England, November 1993.

[16] T. K. Park, C. Kim, and J. W. K. Hong. Design of Run-Time Configuration Manager for a Multimedia Network Environment. *Proceedings of the Second IEEE Workshop on Systems Management*, Toronto, Canada, June 1996.

[17] M. De Prycker. *Asynchronous Transfer Mode.* Ellis Horwood, New York, 1996.

[18] S. Rampal and D. S. Reeves. Routing and Admission Control Algorithms for Multimedia Traffic. *Computer Communications, Volume 18, Number 10*, pages 755–768, October 1995.

[19] A. Vogel, B. Kerhevré, G. V. Bochmann, and J. Gecsei. Distributed Multimedia Applications and Quality of Service: A Survey. *Proceedings of the 1994 Centre for Advanced Studies Confererence*, Toronto, Canada, November 1994.

[20] L. C. Wolf. *Resource Management for Distributed Multimedia Systems.* Kluwer Academic Publishers, Norwell, Massachusetts, 1996.

[21] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource Reservation Protocol. *IEEE Network Magazine, 9 (5)*, September 1993.