Driving Resource Management With Application-Level Quality of Service Specifications

Michael J. Katchabaw, Hanan L. Lutfiyya, and Michael A. Bauer

Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7

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, great strides are being made towards acceptable quality of service solutions. As researchers in this area have recognized, there are still many challenging open problems needing to be addressed. One of the more interesting, yet difficult challenges is the specification of quality of service—quality of service solutions must handle quality of service specifications as application-level expectations, as opposed to low-level resource reservations. Doing so, however, has been proven to be a non-trivial task.

To address this problem, we have developed an application-driven approach to resource management to support quality of service. We present our general strategy, the design of a solution realizing this approach, and a preliminary prototype implementation based on this architecture. We describe experimentation and experience to date and evaluate our work and its effectiveness based on these preliminary results. Finally, we conclude with a summary of our work and outline our plans to evolve it in the future.

Key words: Application-level quality of service, Quality of service specification, Dynamic resource management.

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

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 [15], the applications being used are becoming larger, more complex, and multimedia in nature. To further complicate matters, these next generation applications such as video-on-demand, distance education, tele-medicine, tele-conferencing, electronic commerce, and several others [14] must co-exist with more traditional applications for transaction processing, data processing, and software development.

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.

It has become evident in recent years that the current approaches to resource management taken in systems software are not suitable to handle this kind of stress. The literature in this area provides considerable testimony to this fact [5, 9, 12, 24, 33, 38, 49]. Consequently, there has been considerable research effort towards developing better approaches to resource allocation and scheduling, with several promising results already realized. Nevertheless, many difficult outstanding problems remain to be addressed.

One particular problem that has proven to be challenging to solve involves the specification of quality of service. As mentioned in [46], "the human user of a multimedia application is the starting point for overall QoS considerations". In the end, it is users of applications that are interested in the level of quality of service being delivered. Consequently, quality of service must be considered from the user's perspective, based on the user's expectations associated with applications. In other words, quality of service specifications must be application-level expectations, as opposed to low-level resource reservations. Would we want the president of a large corporation to specify the number of CPU cycles required by his or her desktop video-conferencing applications to deliver acceptable quality? Certainly not. Instead, as stated in [35], we want users to be able to specify "*what* the performance expectation for their work is, *not how* to achieve this expectation". Unfortunately, providing this support is difficult to do in practice [5].

Our current work takes steps towards addressing the above problem. It focuses on the development of a model for resource management to support application-driven quality of service. Users are allowed to specify quality of service expectations for their applications in terms of application-level metrics, as opposed to low-level resource needs. In response to application requests, the underlying resource management system adjusts resource allocations given to these applications (*e.g.*, CPU cycles, memory, and network bandwidth) until the users' quality of service expectations are met. The system is dynamic in that it can adapt to meet changing resource demands and user needs, while at the same time minimizing wasted resources. No assumptions are made of the applications, environments, or resources involved, and our approach is designed to support the work of other researchers in this area, as well as our existing application management solutions [25]. Finally, our approach is structured to assist application developers in building quality of service enabled applications.

In this paper, we begin with a discussion of related work in this area. We then outline the general approach taken in our work, and introduce an architectural framework that realizes this approach. We describe a preliminary prototype system based on this design work capable of adaptive allocation of CPU cycles to applications based on application-level quality of service specifications, and present experimental results and evaluations demonstrating the suitability of our system to its task. The paper concludes with a project status report and our plans for future work.

2 Related Work

As mentioned in the previous section, there has been considerable research in recent years towards developing better approaches to resource allocation and scheduling to support quality of service. In this section, we take a look at some of this work.

There has been significant work done towards improving resource scheduling and allocation of system and network resources for quality of service [4, 5, 6, 7, 8, 9, 10, 12, 17, 21, 22, 23, 24, 29, 30, 32, 34, 37, 38, 42, 43, 45, 47, 49, 50]. Most, if not all, of the related work in this area require applications or their users to specify how to achieve quality of service expectations through low-level resource statements, however [5]. Specifications of quality of service are typically done through a statement of the execution rate, period, priority, deadline, time slice size, delay, packet size, or bandwidth required to meet expectations. As stated in [16, 48], this information is generally not available or known to application users or developers, and can be costly to determine in advance. To derive this kind of specification, users or developers would need intricate knowledge of the hardware architecture, network, and system software in the target environment, which is difficult to do for widely heterogeneous environments. Furthermore, users or developers would need to anticipate the infinite myriad of inputs to their applications to develop such requirements in advance. Otherwise, their resource specifications may either be insufficient, resulting in unacceptable quality being delivered, or overly extravagant, resulting in resource shortages and waste. Consequently, this approach to specification is inadequate from the perspective of both application users and developers.

Goal oriented resource management is an adaptive approach to resource management that combines areas such as operations research and stochastic control with resource scheduling and management [35]. In this approach, only performance goals for applications are specified, and the underlying system adjusts resource allocations until the goals have been satisfied. Consequently, this approach to resource management has the potential for giving insight into the problem we are currently examining. Interesting proposals in the area of goal oriented resource management have been presented in [3, 16, 35]. Along similar adaptive lines, work has also been done to apply economic models to resource management in [11, 39]. In both cases, however, much of this work is largely theoretic and algorithmic. Proof of concept has largely been in the form of simulation studies, with few general prototype implementations.

Interesting and innovating work has also been carried out towards quality of service specification and mapping through the use of perceptual quality metrics, which are based upon how humans perceive quality in their applications. Much of the literature in this area makes use of a five-level scale, with quality ratings ranging from imperceptible quality impairment (Level 5) down to very annoying quality impairment (Level 1) [2, 20, 44]. With these approaches, users can easily select the level of quality desired at a very high level of abstraction, perhaps using slide bars, radio buttons, or some other graphical mechanism. In work such as [1, 13], studies have been carried out to determine how human users would map these quality levels into lower-level quality metrics for a variety of applications. For example, in subjective tests detailed in [1], users of video applications rated quality provided by the applications at different frame rates and resolutions, using the five-level scale. These test results could be used to derive mappings to convert from quality levels to target frame rate ranges, with slight adjustments as necessary on a per-individual basis. Unfortunately, the work in this area also attempts to map these derived application-level metrics to resource requirements directly. This, however, encounters several of the same difficulties mentioned above with traditional quality of service specification techniques, as trying to determine the resource requirements for an application under all possible hardware and software configurations, workloads, and resource demands is extremely difficult, if not intractable.

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

In our current work, we will apply the experience and lessons learned from the related work described in this section to the development of a model for resource management taking significant steps towards supporting statements of application-level expectations as quality of service specifications. In the next section, we examine this model in detail.

3 A Model for Application-Driven Resource Management

To address the problem outlined in the previous sections, we have developed a model for resource management to support application-driven quality of service. This approach is application-driven in that all expectations of quality of service are expressed in terms of application-level metrics, and that the process of quality of service provision is based on application requests to change resource allocations to meet these expectations. In this section, we first discuss the general strategy taken in our model, and then describe an architectural design that realizes this strategy.

3.1 General Strategy

The general strategy in our model is to dynamically adjust the resource allocations given to an application until the quality of service it delivers meets expectations. These expectations can be defaults selected by application designers, preferences chosen by the application user, or corporate policies set by administrative personnel. If the delivered quality of service exceeds expectations by a large margin, the application could do with a less generous resource allocation. On the other hand, if the delivered quality of service is below expectations, the application requires a better resource allocation, either through the allocation of additional resources, or a better distribution of previously allocated resources. This approach is similar to that suggested in [3, 16, 35].

For example, suppose we are considering the only resource available for allocation to be the CPU. To provide a process with a better resource allocation in this scenario, we allocate it more CPU; alternatively, to provide a process with a less generous allocation, we reduce its CPU allocation.

The general strategy for this approach is shown in Figure 1. As we will see in Section 3.2, this strategy can be carried out by multiple concurrent tasks. From this general strategy, we can see the following:

- We make no assumptions about the metrics being used to measure delivered

Given: Application p.
Initial resource allocation a.
QOS metric m.
QOS expectations e.

---

p is started with resource allocation a.
while (p **not done**) **do:**
    Execute code in p.
    $q \leftarrow$ QOS measured with m.
    if ($q > e$) **then**
        Determine allocation reduction r.
        $a \leftarrow a - r$.
    else if ($q < e$) **then**
        Determine allocation improvement i.
        $a \leftarrow a + i$.
    endif
endwhile

Fig. 1. Application-Driven Resource Management Strategy

quality of service—we can use virtually any metric as long as it is measurable within a program. This includes metrics such as: frames of media processed per second, application responsiveness, transactions processed per second, data loss, synchronization, jitter, and so on.

- We do not need to worry about mapping application-level quality of service expectations to lower-level resource requirements. This is due to the dynamic nature of our approach that adjusts resource allocations until a suitable allocation is made that meets expectations. As a result, we can specify at an application-level *what* our expectations are, as opposed to *how* to achieve them.
- If we are not meeting our quality of service expectations, this strategy will continue to adjust resource allocations until either our expectations are met, or there are no longer sufficient resources to meet our expectations. Similarly, if we are exceeding our quality of service expectations, this strategy will continue to adjust allocations until either our expectations are met, or the process has relinquished all resources. From this, we can see that we will make progress towards meeting our expectations until we either meet them, or we are prevented from doing so by either running out of resources to allocate, or by not being able to give up further resources from a process (which is not likely).
- Only the resources required to meet quality of service expectations are used, as any excess resources will be returned to the system for reallocation elsewhere. As a result, this approach results in less resource waste due to over-allocation.

- We do not need knowledge of the underlying system architecture, network, operating system, or resource allocation schemes. This approach will work as long as there is a way to adjust resource allocations dynamically, whether that is done through changing execution priorities, rates, periods, time slices, delays, bandwidth, or whatever underlying approach is used. As a result, this approach is portable to a wide variety of systems and scheduling paradigms.
- We can further improve this process by recording the resource allocations that best lead to satisfying quality of service expectations. When the application is next executed, we could select a better initial resource allocation using one of the allocations recorded during a previous similar run.

Analysing this strategic approach leads to a few natural questions. How would one realize this approach for practical use? What is the overhead in using this strategy? How long does it take this approach to converge to meet quality of service expectations? After convergence, how stable is this approach? How difficult is it to have existing applications make use of this approach? The last four issues are addressed by experimentation in Section 5, while the first is discussed in the next section.

3.2 Architectural Design

To support our strategic approach from Section 3.1, we have developed the architecture shown in Figure 2. This architecture is derived from our earlier work in application instrumentation [25]. This approach requires the insertion of code into applications at strategic locations to facilitate the collection of quality of service metrics and exertion of control over the applications. While some measurements can be taken by observing external application behaviour and rudimentary control can be achieved through operating system interactions, work in this area has found that these approaches are limiting in both accuracy and the kinds of metrics and control available. Since these limitations are unacceptable for quality of service management, instrumentation is the only option.

The application-driven resource management architecture we have developed is object-oriented: Classes of standard components can be developed for general tasks and new components can be specialized easily from existing standard classes. This allows application developers to reuse components easily to reduce development effort, while allowing them to build application-specific components tuned to their particular needs. The components of this architecture are discussed in the sections below.

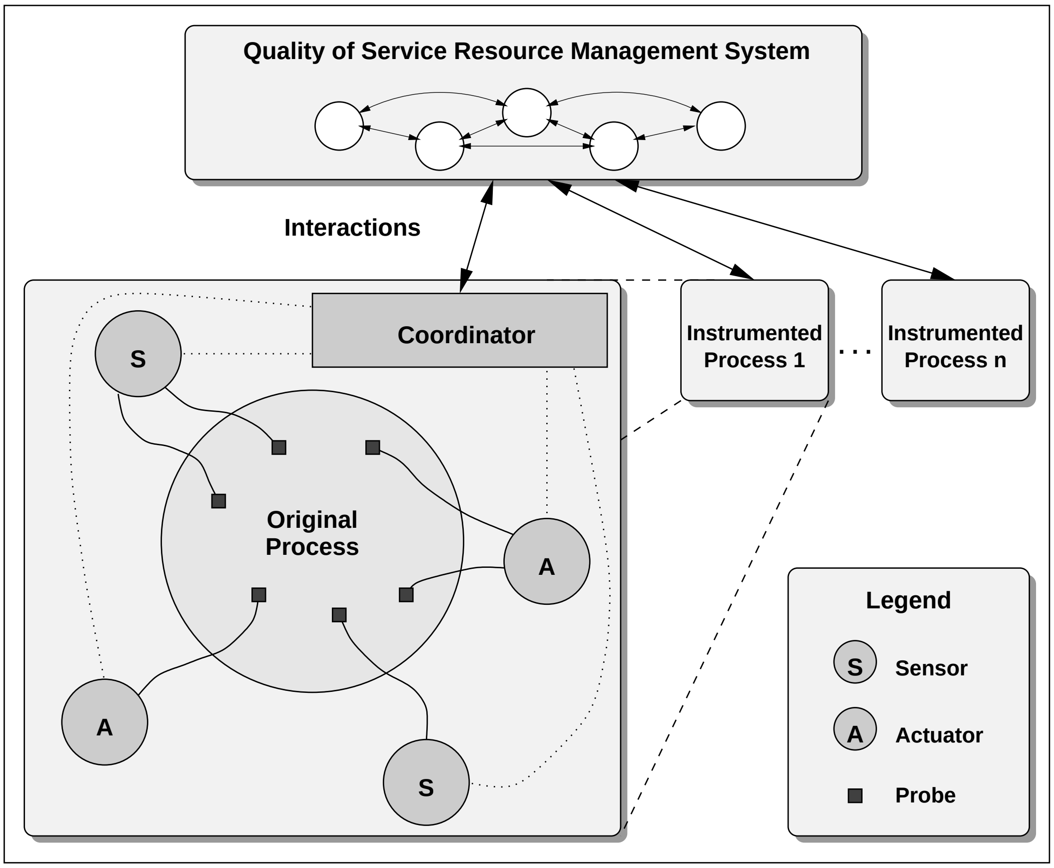

Fig. 2. Application-Driven Resource Management Architecture

3.2.1 Instrumented Process

An instrumented process is an application process with embedded instrumentation code. It is this instrumentation code that enables the management of the application process.

The architecture components that comprise the instrumentation code are discussed below.

• **Coordinator:** The coordinator facilitates interactions between a resource manager and the instrumented process. Its role includes message routing, managing sensors and actuators, and handling initialization and termination activities within the process. All knowledge of the resource manager is confined to this component, effectively hiding it from the remaining instrumentation components.

• **Sensors:** Sensors are used to collect, maintain, and (perhaps) process a wide variety of metric information within the instrumented processes. Sensors get their input data from probes inserted at strategic points in the application code or by reading other sensors. Sensors provide their information to the coordinator in the form of periodic reports, alarm reports (when exceptional or critical circumstances arise), or in response to explicit requests. Sensors are dynamic components—they can be enabled or disabled, reporting intervals can be adjusted, thresholds can be modified, and sensor processing algorithms can be changed, all at run-time while the application process is executing.

By instrumenting processes with the appropriate sensors, application-level quality of service metrics, such as those described in Section 3.1, can be easily collected from executing processes. Specifying application-level quality of service expectations can be done by first formalizing expectations in terms of policies, using some high-level policy specification language [31], perhaps presented to the user in some graphical format. These policy specifications can be easily translated into thresholds which are supported directly by the sensors in our architecture. For example, if we were to use the approach taken in perceptual quality of service discussed in Section 2, we could translate user-selected quality levels into application-level quality metric ranges, and use these ranges as thresholds in our sensors. Consequently, by detecting when these thresholds are violated within sensors, we are able to determine when the quality of service delivered by an instrumented application deviates from expectations.

• **Actuators:** Actuators are used to encapsulate functions that can exert control over the instrumented process to change its operation or behaviour. Actuators, like sensors, function through the use of probes or other actuators, and possess similar dynamic properties. In this current work, they are not used extensively, but can be used to support quality of service negotiation, adaptation, and other functions in the future.

• **Probes:** Probes are embedded in process code to facilitate interactions with sensors and actuators. Probes allow sensors to collect metric information and allow actuators to exert control over process behaviour. Each probe is specific to a particular sensor or actuator. Probes are the only instrumentation component that must be injected into the original process source code—all other components need only reside in the same address space as the process, and do not require code modifications.

3.2.2 Quality of Service Resource Management System

A quality of service resource management system, or simply a resource management system, is a collection of applications responsible for managing a set of resources, such as CPU cycles, memory, network and disk bandwidth, and so on. This includes allocating and scheduling the resources, as well as accounting for their use. A resource management system can be structured as single or multiple resource manager processes spread throughout a distributed environment. How exactly the resource management system is structured, and how responsibilities for resources and applications are assigned to individual

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

resource manager processes are completely open. One can imagine a hierarchy of resource manager processes working with one another to manage the resources of a large distributed environment.

When an instrumented process is not meeting quality of service expectations, its coordinator will notify the resource management system to adjust resource allocations so that these expectations can be met. To do so, the resource management system must determine where in the distributed environment changes are needed, which resources should have their allocations changed, and how much the allocations should be changed. The adjustments carried out by the system depend upon the application making the notification, the resources in question, the allocation and scheduling paradigms used, and the underlying system and network architecture. Since we make no assumptions on how the adjustments are made, this architecture can support any underlying system and resource scheduler or manager.

The model of resource management discussed in this section is a significant step towards solving the problems outlined in Section 2. In the next section, we describe a prototype implementation of this approach.

4 Prototype Implementation

In this section, we describe a prototype system we have developed based on the generic architecture presented in the previous section. This prototype has been implemented for Solaris, and has already been partially ported to AIX and Windows NT. Additional porting efforts are currently under way.

4.1 Instrumentation Implementation

Instrumentation is provided to application developers through a C++ class library. This library contains a variety of standard sensors, actuators, and coordinators. Probes for sensors and actuators are provided as methods on these objects which are inserted into application source code to be executed along with the application code, or periodically through a timer.

In our class library, we provide sensors capable of measuring frame rates of delivered media, transaction processing rates, computational processing rates, interactive response times, and data loss, as well as a coordinator for our resource management system (discussed below). The library can also be specialized to develop new classes of instrumentation components to meet the unique needs of specific applications. This is done through providing a variety of abstract classes suitable for specialization.

Given: Video application v.
QOS expectations e.

1. Perform initialization for v.
2. Initialize coordinator c.
3. Initialize sensor s with e.
4. **while** (v **not done**) **do:**
5. Retrieve next video frame f.
6. Decode and display f.
7. Execute $s \rightarrow$ *probe*.
8. **endwhile**

Fig. 3. Instrumentation Example

As an example, consider the sample pseudo-code for a video playback application in Figure 3. This application has been instrumented with a sensor s, capable of measuring the frame rate of video delivered by the application. When the application starts, instrumentation components are initialized, and the given quality of service expectations e are translated into thresholds for the sensor s. Once initialized, the application retrieves video frames, decodes them, and displays them. After each frame is processed, a sensor probe is called, as shown on line 7 of Figure 3. The sensor s measures the time since the last probe invocation, and uses this value to determine the current frame rate, filtering out noise and unusual spikes in the measurements. If the given quality of service expectations e are not met, the sensor s will notify the coordinator c. The coordinator c will, in turn, notify the resource management system of the situation to have resource allocations adjusted appropriately, if necessary.

Further implementation details of our instrumentation and code examples can be found in [25].

4.2 Resource Manager Implementation

As proof of concept, we have developed a simple resource management system capable of dynamically changing resource allocations to instrumented processes. As mentioned previously, we are focusing on handling CPU allocations on a single system in our initial work—we are currently developing support for additional resources, including memory and network bandwidth, as well as support for managing and coordinating the resources of multiple systems [26].

The approach in this prototype is to develop a user-level CPU resource management system, as was done in [5, 24]. The resource management system is structured as a single resource manager daemon process executing at the highest possible real-time priority on the system [40]. Essentially, this means that the manager will execute whenever it is in a runnable state. This, along with locking it into memory, provides excellent responsiveness. (We could also execute the manager at the highest possible time-sharing priority available, with a slight cost in responsiveness.)

Instrumented processes communicate with the resource manager using semaphores and shared memory registered at the initialization of the processes. When the quality of service delivered by a process deviates from expectations, a sensor threshold is violated. The sensor computes how far it is from its target level of quality, and passes this on to the coordinator for the process. The coordinator takes this value and signals the resource manager using semaphores to inform the system of the need for a resource allocation adjustment.

In our initial prototype, the resource manager adjusts allocations dynamically through changes to the time-sharing priorities assigned to processes using the *priocntl()* system call. When the coordinator for an instrumented process notifies the resource manager that it is not meeting quality of service expectations, the system will increase or decrease the time-sharing priority for the process, depending upon whether the process was not meeting or exceeding expectations respectively. The amount of increase or decrease is proportional to how far a process is away from meeting expectations—the further away, the larger the adjustment. Thus, as an instrumented process executes, its time-sharing priority will be dynamically adjusted, moving the process towards the CPU allocation necessary to meet, but not exceed, quality of service expectations.

The prototype implementation described in this section is a solid proof of concept of our general application-driven model from Section 3. In the next section, we present experimental results and evaluations that exercise this prototype, thereby demonstrating the utility of our application-driven approach to quality of service management.

5 Experimental Results and Experience

An initial set of experiments was conducted to evaluate the operation of the prototype instrumentation and resource manager discussed in the previous section. 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, flexibility, and suitability to task of our work. All of the experiments discussed in this section were carried out using

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

a Quality of Service Management Testbed developed as part of our previous work in this area [28].

This section presents results from experimentation with our application-driven resource manager to date. In particular, we focus on the results from two of the applications we have instrumented thus far, as well as an evaluation of some of the characteristics of our solution.

5.1 MPEG Video Playback Experimentation

A considerable portion of the literature in this area uses the Berkeley MPEG video player [36] as a test application in their experimentation, so we have done the same. We took the latest available version, version 2.3, and instrumented it. To monitor its performance, we added a sensor to measure its video playback throughput, in frames per second. The resulting program was executed on one of our UltraSparcs using Solaris 2.5.1 and our resource manager to assist in dynamically scheduling the application. We instructed the player to display a 320×240 8-bit video file from disk, and we gave it a quality of service policy dictating that it play the video with a mean frame 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). Our experimentation consisted of executing the instrumented video player several times under various constant processor loads generated by a CPU load generator (for example, generating a CPU load of 5.0 implies that there is a mean run-queue length of 5.0). These experiments were repeated in the presence and absence of our resource manager for comparison purposes, and replicated five times for statistical confidence.

Figure 4 compares the mean video playback throughput, in frames per second, for the video player under normal scheduling and with our manager 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 our resource manager 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 player. Consequently, our manager improves the quality of service provided substantially over normal Solaris scheduling.

As indicated in [33], the variance of the playback rate, or 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 mean jitter measured, both with and without our resource manager in place. For these experiments, jitter

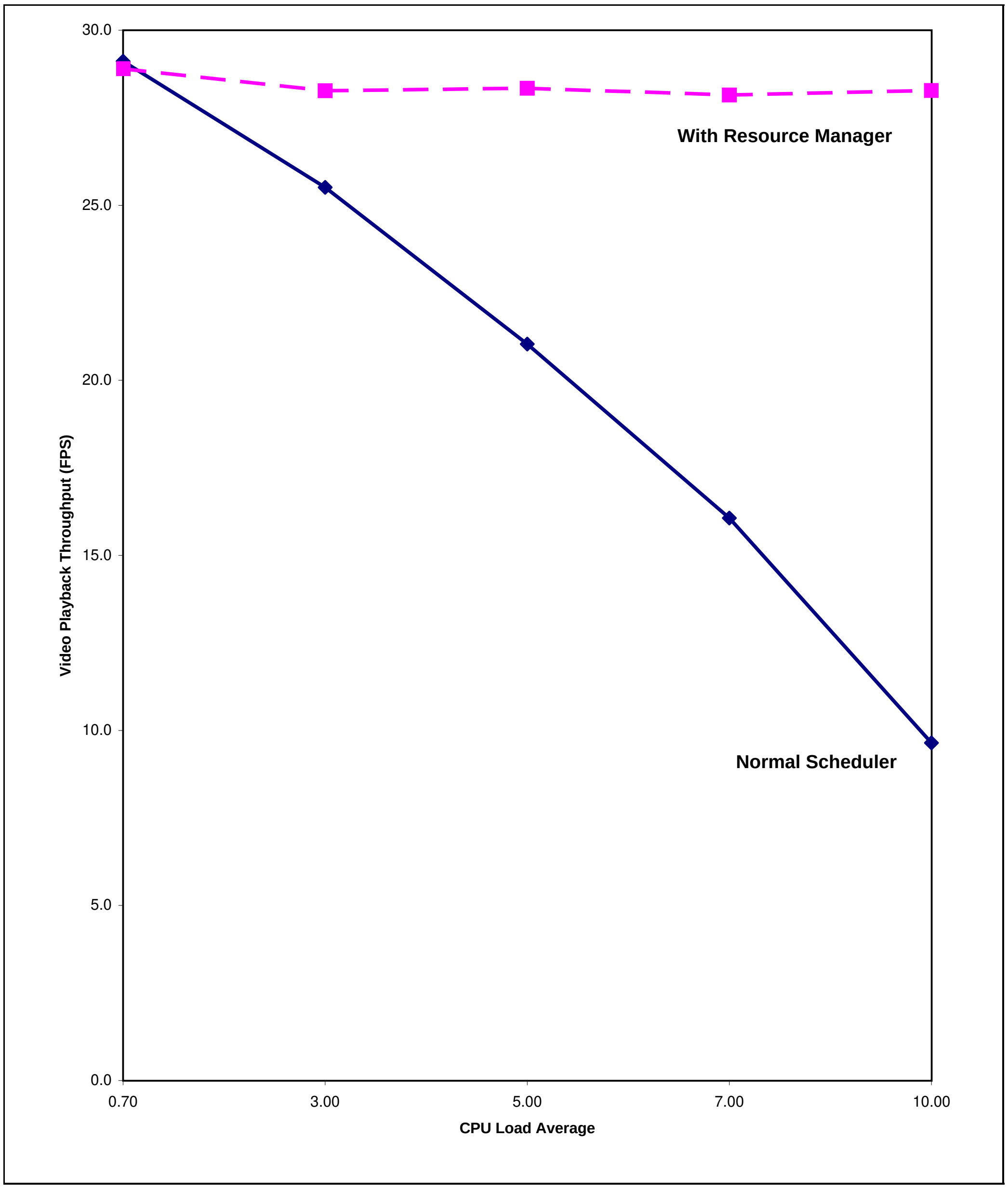

Fig. 4. Video Playback Throughput Comparison

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 our manager in place, the increase in jitter was marginal.

Since the above experiments involved generating a constant CPU load on the workstation, we decided to repeat the above experiments using a randomly varying load. Doing so would provide a more realistic operating environment, and would test the stability of our approach in maintaining a steady level of service in the face of dynamic system conditions. Another batch of experiments were carried out, this time with CPU loads generated according to a

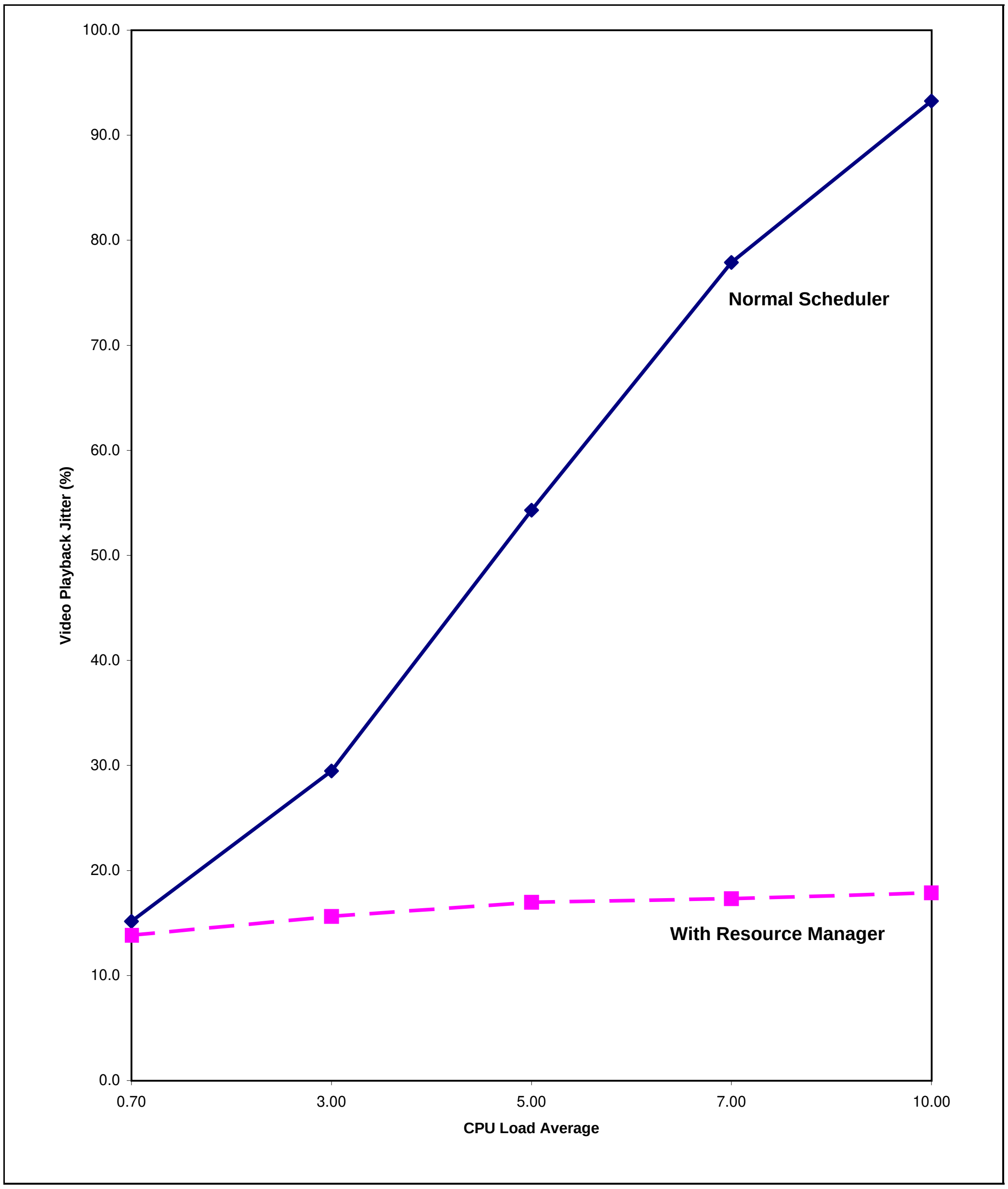

Fig. 5. Video Playback Jitter Comparison

Poisson random distribution with mean loads selected to match the cases for the original experiments. A randomly varying load had virtually no impact on video playback throughput. With our resource manager, jitter increased only marginally by approximately 2%. Under normal Solaris scheduling, however, there was an increase in jitter of up to 32%. As a result, our solution is reasonably stable.

5.2 DOOM Experimentation

With the success from our video player experimentation, we decided to instrument a more complex, resource intensive, and interactive application. With the recent availability of its source code, we chose Id Software's popular game DOOM [19]. The general goal of the game is to reach the exit of each detailed level of the game, eliminating all monsters encountered on the way. The game is graphically intense, and can require a great deal of computing power to play it. Interactivity is a must to maintain the immersiveness of the game.

We took the DOOM source code and instrumented it, as we did the Berkeley MPEG video player. We added a sensor to monitor the frame rate of the game, as this is a measure of the overall performance of the game. The resulting program was executed on one of our UltraSparcs using Solaris 2.5.1 and our resource manager to assist in dynamically scheduling the application. We instructed the game to triple the size of its display to a resolution of 960×600 with an 8-bit colour depth, and played the first level repetitively, following the same sequence of actions. We gave it a quality of service policy dictating that it maintain a mean frame rate of 25 frames per second, allowing for a deviation of 3 frames per second from this target (giving a range of 22 to 28 frames per second). Our experimentation consisted of executing the instrumented DOOM application several times under various processor loads generated by a CPU load generator. These experiments were repeated in the presence and absence of our resource manager for comparison purposes, and replicated five times for statistical confidence.

Figure 6 compares DOOM's mean throughput, in frames per second, for the game under normal scheduling and with our manager in place. From this figure, we can see that throughput once again dropped dramatically under an increasing CPU load when normal Solaris scheduling was used. With our resource manager in place, however, throughput remained reasonably consistent around 24 frames per second – well within the acceptable limits set by the quality of service policy in place. This is impressive considering that DOOM was apparently skipping every other frame of animation when using the normal Solaris scheduler, as only half as many frames were processed when completing the level with the same sequence of actions as when our resource manager was present. Consequently, our manager improves the quality of service provided substantially over normal Solaris scheduling.

As we did for the video player, we carried out a second analysis on DOOM to measure the mean jitter in the time elapsed between successive frames. As can be seen from Figure 7, there was substantially more jitter using the normal Solaris scheduler compared to when our resource manager was in place. In addition to this higher jitter, we also found that DOOM lost keystrokes under

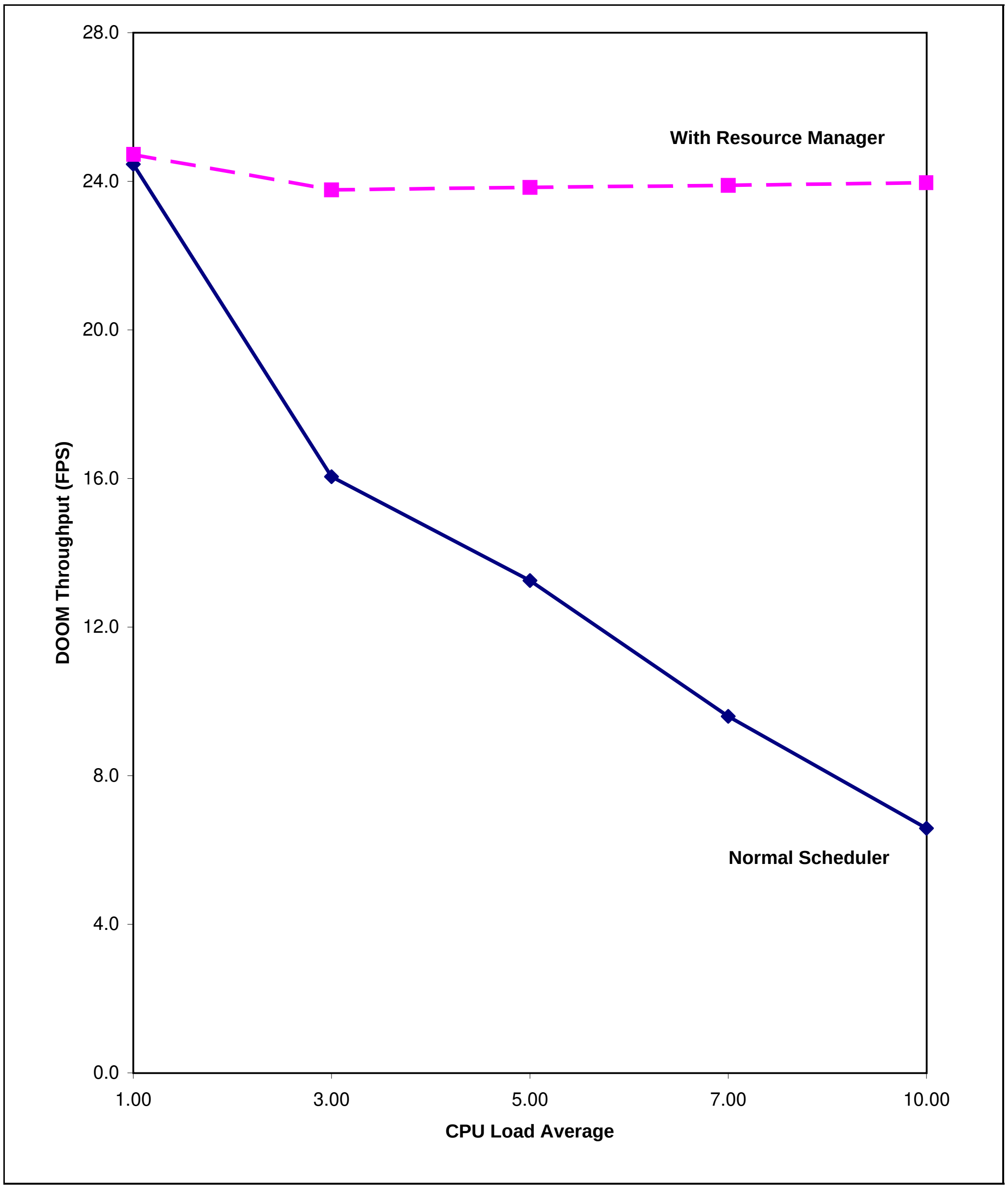

Fig. 6. DOOM Throughput Comparison

high loads when using the normal Solaris scheduler. This problem was not encountered when our manager was in use. This is a considerable head-ache for a highly interactive application such as DOOM (especially when one is surrounded by monsters and cannot get his weapon to fire!). The overall playability of the game was improved substantially by our resource manager.

We are now proceeding with the instrumentation of a wide variety of applications. This includes a numerical analysis application, a database server, a web server, a web client, and a few electronic commerce applications [27]. The results from preliminary experimentation with these applications show improvements in delivered quality of service similar to those presented above.

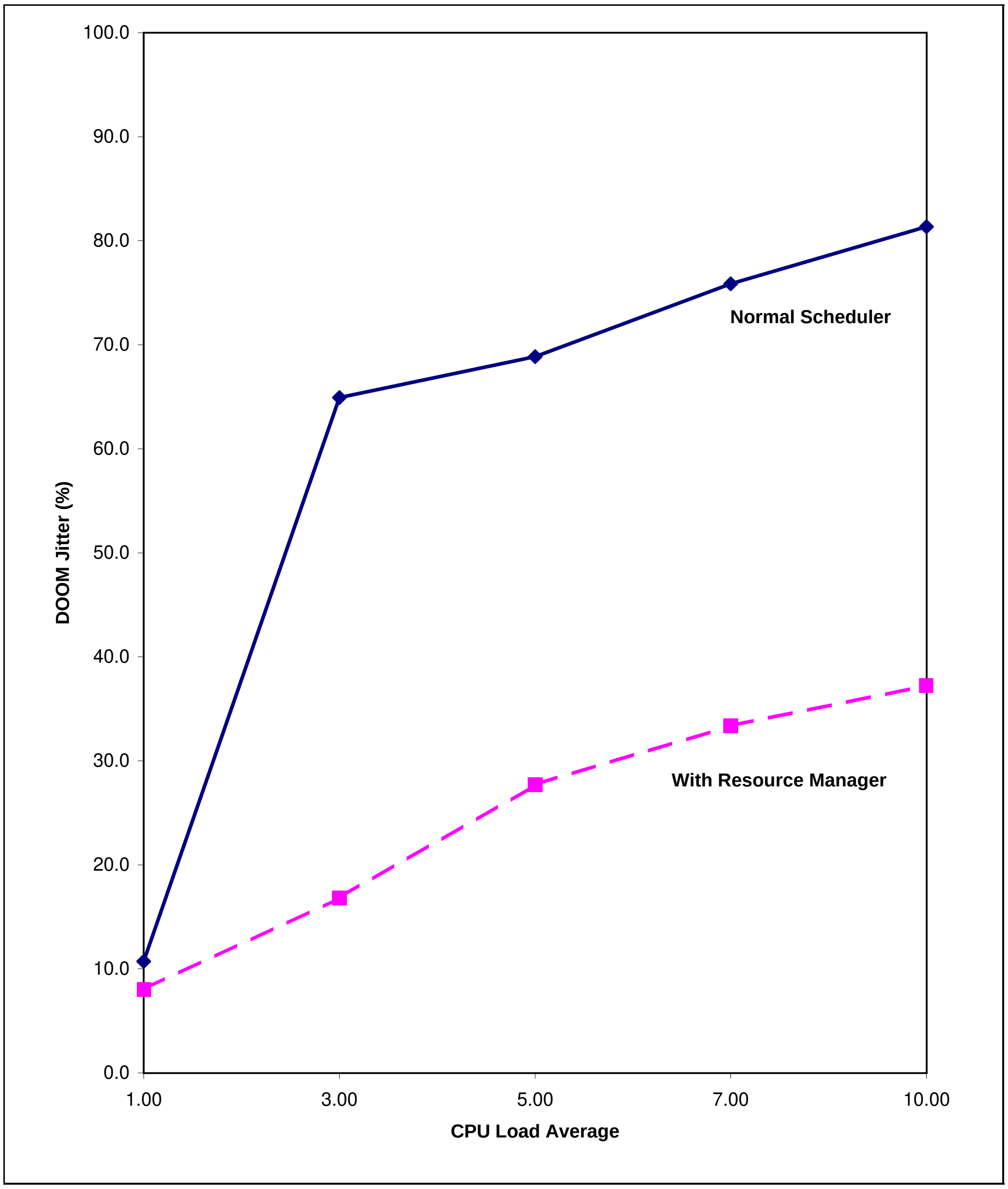

Fig. 7. DOOM Jitter Comparison

5.3 Other Evaluations

The experiments described in the previous two sections have shown that our approach can improve the quality of service provided by instrumented applications with a reasonable degree of stability. We were able to do so through application-level policies specifying quality of service expectations, and our resource manager was able to dynamically adjust resource allocations until these expectations were met. Other issues raised in Section 3.1 and elsewhere have been addressed through additional experiments, as discussed below.

How much overhead does this approach to resource management incur? From measurements taken during experimentation, the overhead from our approach is minimal. An instrumented process on our UltraSparc system requires approximately 400 microseconds more time to initialize itself and report to our resource manager. If the level of quality of service delivered meets expectations, one pass through the instrumentation code requires only 11 microseconds, on average. On the other hand, whenever a quality of service violation is detected, it takes on average 474 microseconds to report the violation to our resource manager, and to have it make appropriate priority adjustments. This is acceptable since quality has already degraded at this point, and the additional time to correct the problem is necessary.

How long does it take for the delivered level of quality of service to converge to the expected level, and how stable is it? In the experiments we conducted in the previous sections, the time for convergence was imperceptible—we converged to expectations before any problems were visible. Furthermore, under randomly fluctuated loads, we could not perceive any deviation from expectations, as we showed through experimentation above. The only visible deviations from quality of service expectations took the form of jitter in our application results. One must keep in mind, however, that our quality of service policies did not explicitly specify jitter constraints, and specified only raw performance constraints, which were met. (This alone is sufficient to reduce fluctuations in performance causing jitter, but is not enough to completely eliminate it.) Once we have completed work on sensors capable of measuring jitter, then we can make use of them to further reduce jitter as well.

The leading threat to convergence and stability of our current prototype comes from interference created by other applications. Because our current resource manager implementation adjusts only time-sharing priorities to change CPU allocations, we currently lack fine grained control and precision. This is not a flaw in our approach or architecture, but a shortcoming of the time-sharing approach to scheduling. We are, however, working on integrating real time algorithms into our prototype resource manager, which will correct this deficiency. (It is important to note, however, that our current approach still offers substantial improvements over existing time-sharing approaches for delivering quality of service. As a result, there is still hope for providing quality of service on systems without real time scheduling support.)

How well does this approach support other kinds of management? In addition to quality of service instrumentation, we provide sensors, actuators, and other coordinators for fault, security, performance, and configuration management in our instrumentation class library. Furthermore, because quality of service instrumentation is derived from base instrumentation abstract classes, existing management applications for fault management, performance management, configuration management, and so on can be used with applications making use of quality of service instrumentation. As a result, any quality of service management solutions making use of this instrumentation can be easily integrated with other management solutions. Initial experiments integrating our quality of service work with our general application management work from [25] have proven quite successful.

How difficult is it to develop quality of service enabled applications using this approach? Developing such applications using this approach is quite simple. All of the applications we are instrumenting are third party applications of which we have no prior knowledge. Nevertheless, instrumenting them for quality of service took very little time. For example, we were able to instrument the source code for DOOM and begin experimentation in less than an hour. The instrumentation that we have developed is quite easy to work with and has a well defined interface to ease use. To further facilitate developing instrumented applications, we are currently building tools to automate the process.

Our approach saves time and developer effort as developers do not need knowledge of the underlying hardware, systems software, and scheduling paradigms in use. Instead of spending time coding complex and potentially error-prone or non-portable routines to overcome deficiencies of the underlying system, they can use their time to develop their application, and use our instrumentation libraries and tools to accomplish the rest.

6 Concluding Remarks

The work we describe here is part of ongoing work on addressing issues in quality of service management. There are many complex issues yet to be resolved. We believe that by developing a dynamic approach to resource management that takes into consideration both the user's perspective and application developers' concerns, we can facilitate the development and acceptance of quality of service management solutions and practical applications that use them.

This paper focuses on the development of a model for resource management to support application-driven quality of service. The most significant contribution of this paper is our application-driven resource management architecture, presented in Section 3. The object-oriented design and approach brings reusability and allows application-specific components to be able to be tuned to particular needs. The instrumentation components of the architecture enable the management of the application process, and even without any prior knowledge of third party applications (which is generally the case), it is quite easy to integrate these components into the architecture to support resource management. Also different from related work in this area, we focus on the problem of what user performance expectations are rather than analyzing methods

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

on how to achieve these expectations. Furthermore, the architecture supports dynamic resource management to accommodate and adjust to changing user needs and resource demands, is applicable in a wide variety of scenarios, and facilitates the development of quality of service enabled applications. Using a prototype system based on this architecture, we have also been able to carry out a number of experiments and have realized several useful results, thereby demonstrating the utility of our work.

We have identified several avenues for continuing and future work in this area:

- It is important to continue our work to support additional resources distributed across multiple hosts, as well as our porting efforts to different platforms.
- As eluded to in the previous section, we plan to continue instrumenting applications and experimentation to further validate our work and analyze the effects of interference between multiple applications on convergence and stability.
- We are also working on several real-time extensions to our current prototype resource management system.
- We also need to extend our work to handle overload conditions when there simply are not enough resources to meet demand; we are currently examining admission control mechanisms and other approaches such as [41] as viable options. Utilizing actuators to facilitate quality of service adaptation and renegotiation is another interesting prospect.
- The work we have done to date is reactive—when quality of service violations occur, we correct them. Another approach that we are investigating is proactive quality of service similar to [18], where potential problems are detected and handled before they actually occur.

Through this work, we have gained valuable 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 computing environments.

References

[1] M. Alfano. Design and Implementation of a Cooperative Multimedia Environment with QoS Control. *Computer Communications (21), Elsevier Science*, Fall 1997.

[2] A. Basso, İ. Dalgiç, F. Tobagi, and C. van den Branden Lambrecht. Study of MPEG-2 Encoding Performance Based on a Perceptual Quality Metric. *Proceedings of the 1996 Picture Coding Symposium*, Melbourne, Australia, July 1996.

[3] K. Brown, M. Mehta, M. Carey, and M. Livny. Towards Automated Performance Tuning for Complex Workloads. *Proceedings of the 20th International VLDB Conference*, 1994.

[4] S. Chen and M. Thapar. I/O Channel and Real-Time Disk Scheduling for Video Servers. *Hewlett Packard Labs Technical Report HPL-96-126*, August 1996.

[5] H. Chu and K. Nahrstedt. A Soft Real Time Scheduling Server in UNIX Operating System. *European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, Darmstadt, Germany, September 1997.

[6] H. Chu and K. Nahrstedt. Memory Management for Soft Real-Time Multimedia Applications. *Technical Report, University of Illinois at Urbana Champaign*, October 1997.

[7] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. *Internet RFC 1883*, December 1995.

[8] L. Delgrossi, R. Herrtwich, and F. Hoffmann. An Implementation of ST-II for the Heidelberg Transport System. *Proceedings of the 1992 IEEE Globcom Conference*, Orlando, Florida, 1992.

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

[10] D. Engler, M. Kaashoek, and J. O'Toole Jr. Exokernel: An Operating System Architecture for Application-Level Resource Management. *Proceedings of the Fifteenth Symposium on Operating Systems Principles*, December 1995.

[11] 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.

[12] 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.

[13] C. Gbaguidi, O. Verscheure, and J. P. Hubaux. A New Flexible and Modular QoS Mapping Framework based on Psychophysics. *Proceedings of the 1997 Management of Multimedia Networks and Services Conference*, Montréal, Canada, July 1997.

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

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

[16] L. Georgiadis, C. Nikolaou, and I. Viniotis. Adaptive Scheduling Algorithms that Satisfy Average Response Time Objectives. *IBM Research Report RC 14851 (#66555)*, August 1989.

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

[17] P. Goyal, X. Guo, and H. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, Washington, October 1996.

[18] C. Hood and C. Ji. Automated Proactive Anomaly Detection. *Proceedings of the Fifth International Symposium on Integrated Network Management*, May 1997.

[19] Id Software Inc. *DOOM.* December 1997.

[20] ITU-R Recommendations BT.500-8. *Methodology for the Subjective Assessment of the Quality of Television Pictures.* February 1998.

[21] K. Jeffay and D. Bennett. A Rate-Based Execution Abstraction for Multimedia Computing. *Lecture Notes in Computer Science 1018:64-78*, 1995.

[22] M. Jones, J. Barrera III, A. Forin, P. Leach, D. Roşu, and M. Roşu. An Overview of the Rialto Real-Time Architecture. *Proceedings of the Seventh ACM SIGOPS European Workshop*, September 1996.

[23] M. Jones, D. Roşu, and M. Roşu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. *Proceedings of the Sixteenth ACM Symposium on Operation Systems Principles*, St. Malo, France, October 1997.

[24] J. Kamada, M. Yuhara, and E. Ono. User-level Realtime Scheduler Exploiting Kernel-level Fixed Priority Scheduler. *Multimedia Japan*, March 1996.

[25] M. J. Katchabaw, S. L. Howard, H. L. Lutfiyya, A. D. Marshall, and M. A. Bauer. Making Distributed Applications Manageable Through Instrumentation. *Journal of Systems and Software*, In Press for 1999.

[26] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. A Model for Resource Management to Support End-to-End Application-Driven Quality of Service. *Proceedings of the Tenth International Conference on Parallel and Distributed Computing Systems*, Las Vegas, Nevada, October 1998.

[27] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. A Model for Resource Management to Support Quality of Service in Electronic Commerce Applications. *To Appear in Technological Challenges of Electronic Commerce*, Publication Date To Be Determined.

[28] M. J. Katchabaw, H. L. Lutfiyya, and M. A. Bauer. A Quality of Service Management Testbed. *Proceedings of the Third IEEE International Workshop on Systems Management*, Newport, Rhode Island, April 1998.

[29] C. Lee, R. Rajkumar, and C. Mercer. Experiences with Processor Reservation and Dynamic QOS in Real-Time Mach. *Proceedings of Multimedia Japan*, March 1996.

[30] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, June 1997.

[31] D. Marriott, M. Mansouri-Samani, and M. Solman. Specification of Management Policies. *Proceedings of the Fifth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Toulouse, France, October 1994.

[32] C. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves: Operating System Support for Multimedia Applications. *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Boston, Massachusetts, May 1994.

[33] 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.

[34] J. Nieh and M. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. *Proceedings of the Sixteenth ACM Symposium on Operation Systems Principles*, St. Malo, France, October 1997.

[35] C. Nikolaou, D. Ferguson, and P. Constantopoulos. Towards Goal Oriented Resource Management. *First Workshop on Load Balancing on High Performance Parallel and Distributed Systems*, June 2, 1995.

[36] K. Patel, B. Smith, and L. Rowe. Performance of a Software MPEG Video Decoder. *Proceedings of the 1993 ACM Multimedia Conference*, Anaheim, California, August 1993.

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

[38] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. Baruah, J. Gehrke, and C. Plaxton. A Proportional Share Resource Allocation Algorithm for Real-Time, Time-Shared Systems. *Proceedings of the Real-Time Systems Symposium*, Washington, DC, December 1996.

[39] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An Economic Paradigm for Query Processing and Data Migration in Mariposa. *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, Texas, September 1994.

[40] Sun Microsystems Inc. *Multithreading and Real-Time in Solaris: A White Paper.* Mountain View, California, 1992.

[41] T. Tan and W. Hsu. Scheduling Multimedia Applications Under Overload and Indeterministic Conditions. *Proceedings of the Third IEEE Real-Time Technology and Applications Symposium*, Montreal, Canada, June 1997.

[42] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Toward a Predictable Real-Time System. *Proceedings of USENIX Mach Workshop*, October 1990.

[43] C. Topolcic. Experimental Internet Stream Protocol, Version 2 (ST-II). *Internet RFC 1190*, 1990.

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000

[44] C. van den Branden Lambrecht and O. Verscheure. Perceptual Quality Measure using a Spatio-Temporal Model of the Human Visual System. *Proceedings of the 1996 SPIE International Symposium on Visual Communications and Image Processing*, San Jose, California, February 1996.

[45] C. Venkatramani and T. Chiueh. Design, Implementation, and Evaluation of A Software-Driven Real-Time Ethernet Protocol. *Proceedings of the 1995 ACM SIGCOMM Conference*, Cambridge, Massachusetts, August 1995.

[46] 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.

[47] F. Wang, C. Wen, C. Cheng, M. Lee, T. Lin, S. Wang, and Y. Oyang. Memory and Disk Bandwidth Management in Video-on-Demand Servers. *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital and Audio Video*, Zushi, Japan, April 1996.

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

[49] D. Yau and S. Lam. Adaptive Rate-Controlled Scheduling for Multimedia Applications. *Proceedings of the 1996 ACM Multimedia Conference*, Boston, Massachusetts, November 1996.

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

Appeared in Decision Support Systems Journal, Volume 28, Elsevier Science Press, 2000