A MODEL FOR RESOURCE MANAGEMENT TO SUPPORT END-TO-END APPLICATION-DRIVEN QUALITY OF SERVICE*

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

Because of considerable research effort into quality of service management, 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. Quality of service solutions must handle quality of service specifications as application-level expectations, as opposed to low-level resource reservations. These solutions must be dynamic to meet changing environment conditions and user requirements, and general enough to be applicable in a wide variety of situations. Finally, these solutions must also facilitate the development of quality of service enabled applications, instead of hindering it.

To address these problems, we have developed a dynamic end-to-end application-driven approach to resource management. 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.
[**Keywords:** Application-level end-to-end quality of service, dynamic resource management]

1 INTRODUCTION

It has become evident in recent years that the current approaches to resource management taken in systems software are not suitable to handle the stress induced by demands for high levels of quality of service by users in today's computing environments. The literature in this area provides considerable testimony to this fact [1, 2, 3]. 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:

- Quality of service must be considered from the user's perspective, *i.e.*, based on the user's expectations associated with applications. As stated in [4], we want users to be able to specify "*what* the performance expectation for their work is, *not how* to achieve this expectation".
- The provisioning of quality of service must be dynamic to accommodate changing user needs, resource demands, and deficiencies in translating user expectations to actual resource allocations.
- Quality of service solutions must be able to cope with allocating and scheduling multiple resources with no assumptions made on the applications, environments, and other management solutions in place.
- Support for developing quality of service enabled applications must be provided and made as straightforward as possible for developers.

Our current work takes steps towards addressing the above problems. 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 [5]. 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, 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.

*An extended version of this paper is available as Technical Report # 528 from the Department of Computer Science at the University of Western Ontario.

Appeared in the Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems , Las Vegas, Nevada, October 1998

2.1 RESOURCE SCHEDULING AND ALLOCATION

A variety of novel approaches to CPU scheduling and allocation have been developed, as discussed in [6, 7], including processor capacity reserves, rate-based execution, rate-controlled scheduling, hierarchical scheduling, proportional share resource allocation, scheduling graphs and time constraint satisfaction, and biased virtual finishing times. Several approaches to reserving and scheduling network resources have been developed, including RSVP, ST-II, HeiTS, Tenet and others. Lower level work in IPv6, ATM, and real-time ethernet have also made progress in network resource management. Additional work has been done in terms of memory, disk, and I/O resource allocation and management. All of this work provides improvements over traditional approaches to scheduling, allowing tighter control over how a variety of resources are allocated to applications. Some researchers have used ideas and concepts from the above work in developing new operating systems. The work described in [1, 2] is also significant, providing flexible techniques to introduce improved resource allocation schemes into existing systems, without kernel modifications.

This work represents significant progress in the area of quality of service management. As this work concedes, however, there are still many challenging open problems needing to be addressed, as mentioned in Section 1.

2.2 GOAL ORIENTED RESOURCE MANAGEMENT

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 [4]. 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 some of the outstanding problems discussed in the previous section.

Interesting proposals in the area of goal oriented resource management have been presented in [4, 8, 9]. Unfortunately, much of this work is largely theoretic and algorithmic. Proof of concept has largely been in the form of simulation studies, with few prototype implementations.

3 A MODEL FOR QUALITY OF SERVICE RESOURCE MANAGEMENT

We have developed a model for resource management to support end-to-end application-driven quality of service. This approach is end-to-end in that it supports quality of service on the end-system, in addition to across the interconnecting networks. 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.

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. 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 [4, 8, 9].

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.

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

FIGURE 1: APPLICATION-DRIVEN RESOURCE MANAGEMENT STRATEGY

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. This approach has several advantages. No assumptions are made about the metrics used to measure delivered quality of service, so virtually any metric can be used, providing it is measurable within a program, including: frames of media processed per second, application responsiveness, transactions processed per second, data loss, synchronization, jitter, and so on. Since application-level metrics can be easily used, the details of how quality is provided are effectively hidden from the user and application developers by the underlying quality of service solution. The dynamic nature of this approach adjusts resource allocations until only those resources needed to meet user expectations are allocated, resulting in user satisfaction, reduced resource wastage from overallocation, and side-stepping difficult quality of service mapping problems [1]. This dynamic nature also allows quality constraints and expectations to change while the application and quality of service solution are executing. Finally, this approach makes no assumptions of the underlying system architecture, network, operating system, or resource allocation schemes in use, and 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.

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, we have developed the architecture shown in Figure 2. This architecture is derived from our earlier work in application instrumentation [5]. 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.

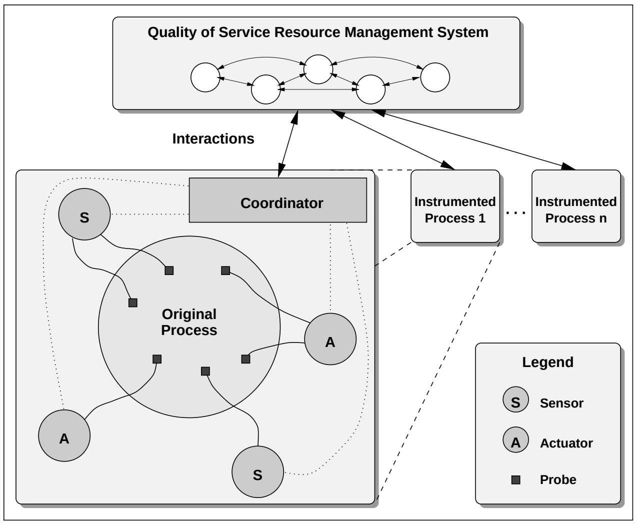

FIGURE 2: APPLICATION-DRIVEN RESOURCE MANAGEMENT ARCHITECTURE

The end-to-end 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.

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.
- **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 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

Appeared in the Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems , Las Vegas, Nevada, October 1998

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.

3.2.3 DETAILED EXAMPLE

In this section, we present an example showing how the general strategy of Section 3.1 is carried out through the components of the architecture described in this section. In this example, we will consider the instrumentation and use of a video application whose pseudocode shown in Figure 3.

```
Given: Video application v.
       QOS policy p.
------------------------------------------------
1.  Perform initialization for v.
2.  Initialize coordinator c.
3.  Initialize frame rate sensor r with p.
4.  Create window w for displaying video.
5.  while (v not done) do:
6.       Retrieve next video frame f.
7.       Decode and display f in w.
8.       Execute r → probe.
9.  endwhile
```

FIGURE 3: INSTRUMENTATION EXAMPLE

Suppose that the user specified a policy p that the application deliver video at a frame rate of 25 frames per second, plus or minus 2 frames. The application is started with its initial resource allocation. At this time, a coordinator c for the process is also created and a frame rate sensor r is initialized. At this point, the quality of service policy p is translated into initial thresholds for the frame rate sensor r. In this example, the policy from above will translate into a threshold specifying a target rate of 25 frames per second, an upper threshold of 27 frames per second, and a lower threshold of 23 frames per second.

Once initialized, the application retrieves video frames, decodes them, and displays them in a window w created by the application. After each frame is processed, a frame rate sensor probe is called, as shown on line 8 of Figure 3. The sensor r 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 policy p is violated, the sensor r 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.

For example, if the delivered frame rate increased to 30 frames per second, the frame rate sensor r will generate a report to have the rate decreased, which would find its way to the resource management system. The resource management system determines a resource allocation reduction or redistribution to move the application's delivered quality of service into line with policy expectations in p, and makes the necessary adjustments. If, on the other hand, the delivered frame rate dropped to 20 frames per second, the frame rate sensor r threshold is violated and the sensor generates a report to have the frame rate increased. The resource management system determines a resource allocation improvement to move the application's quality of service into line with policy expectations p, and makes the necessary adjustments. This improvement could entail either the allocation of additional resources, or a redistribution of previously allocated resources.

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.

Further implementation details of our instrumentation and code examples can be found in [5].

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. This approach to an initial prototype marks a very sound first step—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.

The approach in this prototype is to develop a user-level CPU resource management system, as was done in [1, 2]. The resource management system is structured as a single resource manager daemon process executing at the highest possible real-time priority on the system. Essentially, this means that the manager will execute whenever it is in a runnable state. This, along with

Appeared in the Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems , Las Vegas, Nevada, October 1998

locking it into memory, provides excellent 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.

Recognizing that a time-sharing approach to quality of service is not a good long-term solution, as the literature has shown time and time again, we have designed the resource manager to support real-time scheduling algorithms proposed by other researchers such as rate monotonic and others in the future, and we are currently working on adding this support to our prototype. With such algorithms, the resource manager would dynamically adjust execution time slices, rates, periods, and so on. The instrumented processes, however, would not be affected at all by such changes.

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 a Quality of Service Management Testbed developed as part of our previous work in this area [10].

This section presents results from experimentation with our application-driven resource manager to date. In particular, we present detailed results from one of the applications we have instrumented thus far, and a summary of other experiments with our solution.

5.1 MPEG VIDEO PLAYBACK EXPERIMENTATION

A considerable portion of the literature in this area uses the Berkeley MPEG video player [11] as a test application in their experimentation, so we have done

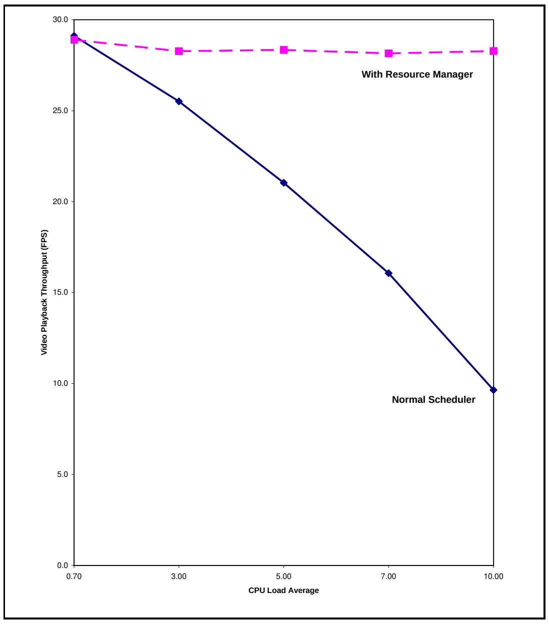

FIGURE 4: VIDEO PLAYBACK THROUGHPUT COMPARISON

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 runqueue 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.

5.2 OTHER EXPERIMENTATION

In addition to the Berkeley video player, we have instrumented a variety of other applications, including several video games, a numerical analysis application, a database server, a web server, a web client, and an electronic commerce application. Results from experimentation with these applications echo the results above.

We also carried out a number of evaluations with our quality of service solution, and found the following:

- The solution incurs little overhead, taking less than 11 microseconds to take a measurement and compare it to thresholds, and less that 474 microseconds to report a quality violation and have resources adjusted appropriately.
- The time for quality of service to converge to expectations is imperceptible and the quality provided is stable after convergence, even under adverse and fluctuating operating conditions.
- The approach integrates well with other management solutions, such as our previous work in [5].
- Developing quality of service enabled applications is quite simple, and can be easily done on complicated, unfamiliar code in less than an hour. We are currently developing tools and techniques to automate instrumentation to further improve the process.

6 CONCLUDING REMARKS

This paper focuses on the development of a model for resource management to support end-to-end 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 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.
- 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 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 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, where potential problems are detected and handled before they actually occur.

REFERENCES

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QoS Architectures. *Multimedia Systems Journal, Special Issue on QoS Architecture*, May 1998.

[7] 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.

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

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

[10] 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.

[11] 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.