Distributed Resource Management to Support Distributed Application-Specific Quality of Service

Gary Molenkamp, Michael Katchabaw Hanan Lutfiyya and Michael Bauer

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

Abstract. This paper presents a policy-based software architecture and implementation for application Quality of Service (QoS) management. The approach is dynamic in that the application is started with an initial resource allocation, and if it does not meet its QoS requirement, a resource manager attempts to allocate more resources to it until its QoS requirement is met. One key problem is that of determining the corrective action(s) to be taken when an application's QoS requirements are not being satisfied. For example, one possible action is to adjust the application's CPU priority when its QoS requirement is not being met. However, if there are several applications on the same host with QoS requirements, this may not be feasible. In other words, there may be constraints on the corrective actions depending on other factors. This paper explores the use of administrative policies to guide decisions on actions to take to meet QoS requirements and presents a prototype and initial experimentation.

Key Words: Dynamic Quality of Service, Policies, Control Actions, Multimedia

1 Introduction

There has been an increase in distributed applications that involve the exchange of data which is time-sensitive. These types of applications arise in a number of areas including video-on-demand, distance education, tele-medicine, tele-conferencing, electronic commerce, and several others [7]. Users of these applications expect them to perform at "acceptable" levels, that is, they expect a high level of quality of service (QoS). Quality of service in this context refers to non-functional, run-time (or operational) requirements, such as the application's performance or availability.

QoS management techniques, such as resource reservation and admission control techniques, can be used to guarantee the QoS requirements, but these techniques usually base the resource reservation on worst-case needs which leads to inefficient resource utilization. In addition, these approaches often require that the developer or user have intricate knowledge of resource needs (e.g., the number of CPU cycles needed). Other QoS management techniques focus on either having applications adapt their behaviour based on reduced resource availability (e.g., change of video resolution) or make adjustments to resource usage in the system. Generally, the adjustments to resources refers to making adjustments to the behaviour of the service providing the resources (e.g., frame dropping will cause a server to back off). Thus, monitored resource usage is used to determine adaptations.

As will be described in detail in the next section, most of the related research is not suitable for environments and applications with the characteristics just described. (i) Often, the QoS requirements are considered to be *soft*, i.e., applications with these requirements are still considered functionally correct if the QoS requirement is not satisfied. This is opposed to *hard* QoS requirements where applications (e.g., flight control systems, chemical process control systems, patient-monitoring systems) must meet the QoS requirement to avoid disastrous consequences. (ii) Many distributed applications are dynamic in nature. For example, the QoS requirements may vary for different users (and different sessions) of the same application, for the same user of the application at different times, or even during single session of the application. Also, during a session the QoS requirements may change. For example, in a multicasting application it is not unusual to have new clients join a

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

multicasting session. (iii) The user or developer will not have intricate knowledge of the hardware architecture and system software in the target environment. This is especially difficult to do for widely heterogeneous environments. (iv) It is also likely that the users of these applications will have multiple applications on their systems, likely running concurrently, and that these applications will also have to co-exist with more traditional applications for transaction processing, data processing, and software development.

This paper describes an approach to QoS management for an environment and applications with the above characteristics.

Section 2 describes related work. Section 3 describes the strategy taken by the our approach to QoS management. Section 4 provides a brief discussion of QoS requirements and policies. A proposed architecture for a system dynamically allocate resources to meet QoS requirements is presented in Section 5 and details of a prototype are presented in Section 6. Some experimental results are presented in Section 7. A conclusion is presented in Section 8.

2 Related Work

There is a large body of work related to QoS management that is briefly described in this section.

2.1 Single Resource Algorithms

Over the years, a number of algorithms have been developed for CPU, disk and network resource allocation (e.g. [20, 3, 22, 16, 15]). A single algorithm is not enough to support QoS management. Users and developers should not interface with or be exposed to these algorithms and approaches directly. Our work addresses this issue by allowing the user and application developers to focus on defining the QoS requirements e.g., "The number of video frames per second displayed to the user must be at least 25", but not having to define low-level resource needs such as the number of CPU cycles. Our QoS management approach hides these details.

2.2 Static QoS Infrastructures for Network Resources

There are a number of approaches for providing network quality of service for multimedia applications. An excellent review can be found in [1]. Most of these approaches are relatively static, although some support some form of renegotiation of the resources allocated to them for satisfaction of their QoS requirements. These approaches have two disadvantages: (i) They require major modifications to the operating systems or (ii) The technique is based on worse-case needs (as the result of using resource reservation) which often leads to inefficient resource utilization. Our work makes use of the existing features provided by the operating system and does not need to use resource reservation which is generally more applicable for systems with hard real-time requirements.

2.3 Dynamic QoS Management Approaches for Application Adaptation

One class of application adaptation mechanisms adapts the resource consumption needs. An example of this can be found in [9]. Another type of application adaptation mechanism alters the topology of the end-to-end processing. Yet another application mechanism allows the user to prioritise services. If there are not enough computing resources then the behaviour of the application changes by changing or eliminating a service. For example, an application may be told to reduce the resolution that it is displaying an image at or eliminate audio and leave video.

Many of the techniques are either used or modified for use in middleware services. For example, the work in [12] introduces a Task Control Model for QoS application adaptations in the Agilos middleware control architecture. The adaptations usually take the form of changing request or delivery rates in applications based on control theory models. The *QuO* project [17] proposes extensions to

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

standard CORBA components and services for supporting application adaptation. Other work [6] includes that which combines resource reservation with application adaptation.

Our approach can support application adaptation (although we do not extensively focus on this in this paper). Our approach does not rely on existing middlewares like CORBA before it can be applied. In fact, our approach can be used to manage multiple and concurrent applications that use CORBA, Java or sockets. As a result, our work can be more easily extended to QoS management across multiple administrative domains. This is not the case with using a system like QuO which assumes that the CORBA services are in all administrative domains.

2.4 Dynamic QoS Management Approaches for Dynamically Changing Resource Allocation

There is relatively little work on dynamic QoS management approaches for changing resource allocations of an application. The work in [2] proposes a quality of service management scheme that dynamically adjusts resource allocations (e.g. the operating system adjusts its computing resource allocations). The applications make most of their own resource management decisions, instead of the operating system or other services. While this may get better results for individual applications, it places additional burden on application developers to develop this logic, and can be dangerous since it relies upon all applications and users to behave in a fair and equitable manner when resources are being allocated. Unfortunately, the implementation and validation of this approach has been self-described as ad hoc and incomplete; further refinements of this work have not been done. Our work is different in that it will allow for the potential to optimise performance across all applications.

The work in [4, 11] describe dynamic approaches to QoS management where the environment adapts as much as possible to application needs. The user is central to this process, as it is up to the user the decide when quality is unacceptable, and what to do to correct the problem, with the assistance of feedback from the system. This degree of user involvement places a great deal of burden on the user.

3 Strategy

Our approach to QoS management does not require user or developer knowledge of their application's computing resource needs. Computing resources are allocated as needed (if available), if the application does not satisfy its QoS requirements. For example, the CPU priority of an application may be adjusted if that application is not satisfying its QoS requirements. This is different than having the application re-adjust its behaviour. Rather, the computing resource allocation is dynamically changed.

This approach entails several problems: (1) The application needs to be monitored in order to determine if the QoS requirements are being met. If they are not, then this must be detected (*violation detection*). (2) Once a QoS requirement has been detected by the monitoring system, the cause of the degradation must be determined (*location*). For example, a violation of the QoS requirement would occur if a video application was expecting to receive at least 23 frames per second, but was not. This could be caused by several situations, e.g., the video application might not be getting enough local processor cycles, the server process might not be getting enough cycles, a process failed, or there is an unexpected load on a network link. Locating the cause is an important step in determining the appropriate resource allocation. For example, if the problem is that there is an unexpected load on the network, then there is no need to adjust resources on the video application process's host. (3) The resource allocation should be adjusted. Examples include providing more processor cycles or restarting a failed process. The actions to be taken depend not only on the cause of the violation, but also depend on the constraints imposed on how to achieve the QoS requirement. For example, one possible corrective action is to adjust the CPU priority of the video application receiving the video stream. However, if there are several multimedia applications on the same host,

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

then perhaps attaining the desired level of service for all is not possible and a different action is necessary, e.g. adjust the priority based on who the user of the video application is.

These constraints constitute a second category of requirements. These are, however, not QoS requirements on the expected behaviour of an application. Rather, they are administrative or organizational requirements. These requirements will vary among different administrative domains and will vary over time.

4 Types of Requirements and Policies

We have identified two different types of requirements: requirements on the run-time behaviour of an application (hence referred to as *application QoS requirements*) and requirements that express constraints on the set of possible corrective run-time actions (hence referred to as *administrative requirements*). Clearly, for both types of requirements to be useful in the management of distributed applications, and given the run-time nature of both sets of requirements, it must be possible to represent both within the management system and it must be possible to determine the specific components that they apply to.

Application QoS requirements for a particular application will change. For example, the requirements of an application may depend on the user who has invoked the application. Thus, different sessions of the same application may have different QoS requirements. Administrative requirements will also change, since the constraints on the possible adaptations will also change during the lifetime of the system.

As described earlier, an application QoS requirement may be violated. In these cases, it must be possible to specify an action. In many cases we will want this action to include sending a notification of the violation to another entity that is doing the diagnosis and determining the adaptation to be taken.

The implication of these two observations is that it must be possible to store both application QoS and administrative requirements. The application QoS requirements should be accessible by an application when that application is started up. It must also be possible to specify the action(s) to be taken if the application QoS requirement is not satisfied.

This suggests that we allow the requirements to be expressed as *policies*. A *policy* can be defined [19] as a rule that describes the action(s) to occur when specific conditions occur. There has always been some use of some form of policies to manage networking resources in most enterprises. For example, system administrators often define policies regarding what types of traffic a firewall can reject. The Internet Engineering Task Force (IETF) is currently developing a set of standards (current drafts found in [14, 18, 21]) associated with policies.

A policy for an application QoS requirement has as its condition the negation of the QoS requirement which means that the specified action(s) is to take place when it has been detected that the QoS requirement has been violated. We call these *expectation* policies. The action(s) will include where to send the notification of a violation of the QoS requirement.

Policies can also be used for administrative requirements. These are called *enforcement* policies. A more detailed description of how these polices can be formalised and supplied to the application upon start-up is described in [13].

5 Architecture

This section describes the key architectural components (depicted in Figure 1). For illustrary purposes, we shall use the following QoS requirement in examples in the remainder of the paper: "The application deliver video at a frame rate of between 23 and 27 frames per second." (lower and upper bounds are used to indicate when the frame rate is too slow or too fast).

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

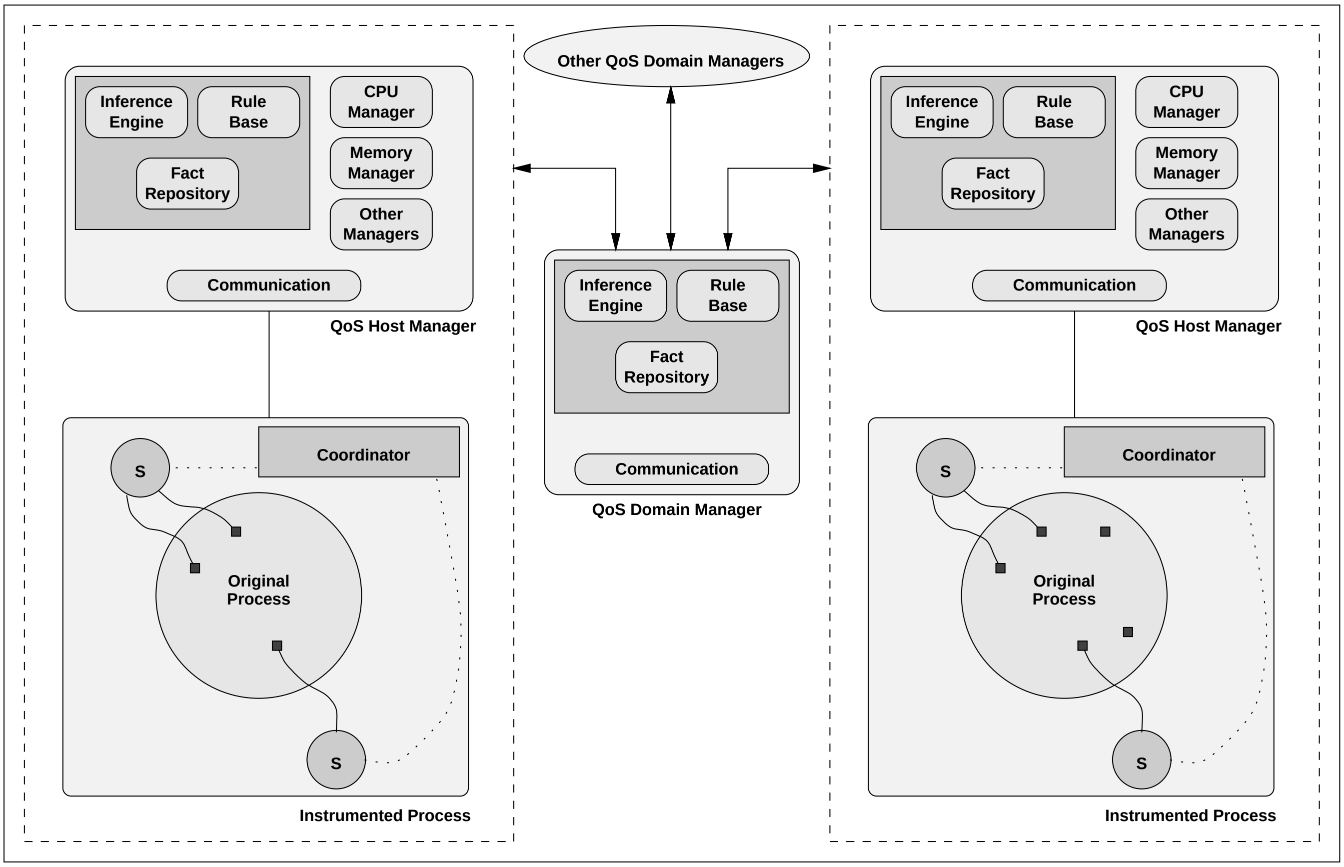

Fig. 1. Management Architecture

5.1 Instrumented Process

Expectation policies express application QoS requirements which are characterised by conditions that must be satisfied about attributes of the application. For example, the frame rate of a video application is an attribute and the statement "a frame rate of between 23 and 27 frames per second" is a condition about that attribute that should be satisfied by the application at run-time.

Instrumentation code is code inserted into an application at strategic locations to collect the values of attributes. While some measurements of attributes 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.

An instrumented process is an application process with embedded instrumentation code. It is this instrumentation code that enables the management of the application process. This instrumentation is provided by several components.

Sensors. Sensors (represented as S in Figure 1) are used to collect, maintain, and (perhaps) process a wide variety of attribute information within the instrumented processes. Each attribute is part of a policy (or policies) being applied to the application; policies specify constraints on process attributes. For simplicity, we associate a sensor with one attribute. Sensors get their input data from probes inserted at strategic points in the application code or by reading other sensors. During run-time, sensors can be enabled or disabled, reporting intervals can be adjusted and thresholds can

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

be modified.

Probes. Probes are embedded in process code to facilitate interactions with sensors. Probes allow sensors to collect metric information. Each probe is specific to a particular sensor. 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.

Coordinators and sensors are defined as classes. Probes are methods of the sensors.

Example 1.
As an example, consider the sample pseudo-code for a video playback application in Figure 2 that has the QoS requirement stated at the beginning of this section. This QoS requirement is translated into initial thresholds for sensor s_1, capable of measuring the frame rate. The target threshold is 25 frames per second, the upper threshold is 27 frames per second and the lower threshold is 23 frames per second.

Sensor s_1 includes at least the following two probes which are also methods of s_1: (1) An initialisation probe that takes as a parameter the default threshold target value. This probe gives a value to the target, as well as the upper and lower thresholds. (2) A probe that (i) determines the elapsed time between frames. (ii) checks to see if this time falls within a particular range defined by the lower and upper acceptable thresholds, and (iii) informs the coordinator if the constraints on the values of the frame rate are not satisfied c. □

Given: Video application v.
QOS expectations e.

---

1. Perform initialisation for v.
2. Initialise coordinator c.
3. Initialise sensor s_1 with e.
4. **while** (v **not done**) **do:**
5. Retrieve next video frame f.
6. Decode and display f.
7. Execute $s_1 \rightarrow$ *probe*
8. **endwhile**

Fig. 2. Instrumentation Example

How does a sensor relate to policies that express application QoS requirements? A sensor collects values for an attribute of the process. This attribute is part of a policy (or policies) being applied to the application. Application policies specify constraints on process attributes. We assume that a sensor is responsible for monitoring a specific attribute. If the sensor finds that a constraint involving the frame rate attribute is violated, it reports this to the coordinator as an *alarm report*. For our example QoS requirement, it is assumed that one sensor is needed for the frame rate. We note that a sensor may provide values to be used in more than one policy. This means that there is a many to many relationship between policies and sensors. An application may also have more than one policy and it is possible that these policies share attributes. We note that not all sensors measure attributes that are directly used in the specification of a QoS requirement of a policy (this will be illustrated later).

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

Coordinator. A coordinator maintains information about each policy including the following: (i) The policy identifier. (ii) A condition list where a condition is represented by the attribute, comparison operator and value that the attribute is to be compared to using the comparison operator. (iii) An action list. Since a policy may be specified using more than one condition on attributes, when the coordinator receives an alarm report from a sensor it needs to check to see if a policy has been violated. If so, it takes the actions specified in the action list which most likely include a message notifying the QoS Host Manager. It also disables the policy – this means that violations of the policy will not send out any more messages. This prevents a flood of messages at the QoS Host Manager.

Example 2.
There is a policy that has the negation of QoS requirement stated at the beginning of this section. Thus, when this condition is true (meaning that the QoS requirement is not being satisfied), the associated action list (maintained by the coordinator) specifies that the following actions take place: (i) The size of the communication buffer is requested from the buffer sensor (ii) The communication buffer size, the current frame rate, the identifier of the policy are among information sent to the QoS Host Manager for further analysis.

5.2 Quality of Service Host Manager and Domain Manager

The QoS Host Manager receives notifications from a process (through the process's coordinator) when a policy has been violated. The QoS Host Manager has a set of rules that are used to determine the corrective action(s). This involves determining the cause of the policy violation and then determining a corrective action(s). The process of determining the rules to be applied is called inferencing. Inferencing is used to formulate other facts or a hypothesis. Inferencing is performed by the Inference Engine component of the QoS Host Manager. The inferencer chooses which rules can be applied based on the fact repository. The inferencing that can take place can either be as complex as backward chaining (working backwards from a goal to start), forward chaining (vice-versa) or as relatively simple as a lookup. In this work we used forward chaining.

Consider the QoS requirement stated at the beginning of Section 3. A possible corrective action is to increase the CPU priority of the process receiving the frames so that it can process more frames in shorter time. However, what if the problem is not local e.g., the server process feeding the frames is on a host machine that is overloaded or the network connections between the server and the client are congested. If the problem is not local, then changing the CPU priority of the client process will not improve the frame rate. Worse yet, it will have a negative impact on the performance of other processes.

One rule for the QoS Host Manager is informally stated as follows: *If the communication buffer size is above some threshold (implying that the process is not able to process frames fast enough), then the CPU Manager is invoked to adjust the CPU priority of the violated process.* Additional rules are used to determine how much to increase CPU priority based on how close the priority is to being satisfied.

Another rule for the QoS Host Manager can be informally stated as follows: if the communication buffer size is below some threshold then send a notification to the QoS Host Manager, which can then locate the source of the problem, perhaps by interacting with the QoS Host Managers. The implication of a small buffer size is that the video client is able to process the received frames fast enough and that the problem is either a network problem or a server problem. The QoS Domain Manager also has a rule set that is used to drive the location process and guide the formulation of corrective actions. One such rule for the QoS Domain Manager is informally stated as follows: Upon receiving an alarm report from the client-side QoS Host Manager, ask the corresponding server-side QoS Host Manager for CPU load and memory usage. Another QoS Domain Manager rule states that if the CPU load exceeds some predefined threshold or the memory usage exceeds some threshold then an alarm report is sent to the server-side QoS Host Manager.

This is a relatively simple set of rules A more complex set of rules would include rules that reflect administrative requirements. We will describe some of these in section on experimental results.

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

6 Details of the Prototype Implementation

To evaluate the approach, we have developed a prototype system based on the architecture presented in the previous section. This prototype has been implemented for Solaris 2.6.

6.1 Instrumentation

We have built a C++ instrumentation library that implements a hierarchy of sensor classes. The base of the hierarchy provides a registration method that provides the ability to have all sensors register with the coordinator in a uniform manner. Other base methods are for enabling/disabling and read/report.

A coordinator retrieves policies from a file. The coordinator creates a random port number with which to listen for messages from the QoS Host Manager. The port number is registered with the QoS Host Manager. Policy violation reports are issued by the coordinator using an IPC message queue that the QoS Host Manager creates.

6.2 QoS Host Manager

The QoS Host Manager was implemented to permit multiple violation reports to be examined concurrently by maintaining a violation queue with a policy identifier, server host, client host, application type and application mode (e.g., is it in client mode or server mode).

The inference engine, rule set and fact repository are implemented using CLIPS [8]. CLIPs rules are used to do the following: (i) Distinguish between local and non-local causes of the policy violation and (ii) Implement the enforcement policies. These rules represent the enforcement policies.

One rule has as its condition that the complaining process is registered (if it hadn't then this is not a QoS-enabled process), the frame rate is low and the size of the buffer is high or the trend is that the buffer size is increasing relatively quickly. This indciates that the frame rate is low and that the problem is local since a full buffer or a buffer whose size is increasing indicates that the process is not processing the frames fast enough. If this condition is true, then the action to be taken is briefly described as follows: Determine if there are enough CPU resources to be allocated to the complaining process. If so, then allocate and if not then assert a serv_diff fact. This causes the condition of another rule to become true which in turn has the associated action that the complaining process scales back its threshold for a specified period of time.

6.3 CPU Manager

The CPU Manager is organised as follows. There is a communication component that deals with communication with the inference engine. Requests from the inference engine like are received using message queues. The results are available through shared memory. Requests include the registration and deregistration of processes, allocation adjustment requests, enabling/disabling requests, and service differentiation requests. These requests are dispatched to the scheduler manager component whose functionality is explained shortly.

The scheduler manager component is an application running at the superuser level and has the highest priority that maintains information about the number of processes running at each of the real-time priority levels as well as the maximum allowed (which can be changed). By default, all user initiated processes start out as time sharing. Resource allocation is done by having the scheduler manager component change the priority of a process using the priocntl system call. This has the effect of moving a process to the end of a queue at a different level of the array. Different classes at the user level are mapped to a set of priority levels associated the real time class. If the maximum is reached then they remain as a time-sharing process.

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

6.4 QoS Domain Manager

There is one QoS Domain Manager in the local network that receives reports from QoS Host Managers. It has rules that can distinguish between a remote machine problem and a network problem (this usually requires a query of other QoS Host Managers) and can tell a QoS Host Manager on a remote machine to adjust CPU and memory allocations where appropriate. Communication between QoS Host Managers and the QoS Domain Manager is through TCP.

7 Experiments

We have implemented a prototype system based on the architecture described in Section 3. The prototype is implemented in a Sun environment running Solaris 2.6.

To examine the effectiveness of this approach, we evaluated the prototype in a number of experiments involving three different applications - a real time video conferencing application (rtvc), a video player application (mpeg), and a multi-person game with video (doom). The applications were chosen because a) they involve multiple systems, b) they have significant demands for system resources and c) they have soft QoS requirements for the display of video and images. Each application was instrumented with a sensor (fps) that measures frames per second.

The experiments presented in this section show results when the problem causing the QoS violation is the client CPU load. The applications were run under different scenarios and the delivered frames per second on a client system were collected during execution. We graphically present several experiments and briefly discuss each.

Experiment 1: No Management

As a baseline, we ran all the application concurrently, as a single user might do. Figure 3 depicts the frames per second as measured over several minutes. The quality of each, as measured in terms of frames per second varies widely for each application. Given that all are competing for system resources, this is not surprising.

Experiment 2: Managed Environment

In a second experiment, we introduced our management prototype and defined policies. The QoS requirements for the real time video conferencing application and the video player application was a defined frame rate of 25 frames per second, plus or minus 2 frames per second, i.e., it was acceptable if the frame rate was between 23 and 27 frames per second. The game (doom) was assigned no specific policy – in effect, it was designated as "best effort".

We introduced an administrative policy which favoured the video conferencing application over the video player application.

The results are presented in Figure 4. Doom is started first, followed by mpeg and thenrtvc. Within the first minute of the start of rtvc, the system quickly begins to differentiate among the applications, allocating more resources to the video conferencing and video player applications. It is able to manage system resources to enable rtvc to meet its QoS target of 25 frames per second and does a fair job of ensuring that mpeg meets its QoS requirements.

Once rtvc terminates, resources are freed. The system continues to manage the video player application, but now the game can grab available resources. The variability in the delivered frames per second for doom is understandable - it is a best effort approach for getting resources. The reason for the brief decrease in the frames per second for the mpeg application after rtvc terminates and for the variability of the frames per second subsequently is not clear. It could be a side effect of our algorithm, its implementation or unexpected behaviour arising because of the interplay between the real-time scheduler and the time-sharing scheduler in Solaris. We are currently exploring this in greater detail.

Experiment 3: Loaded Processor

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

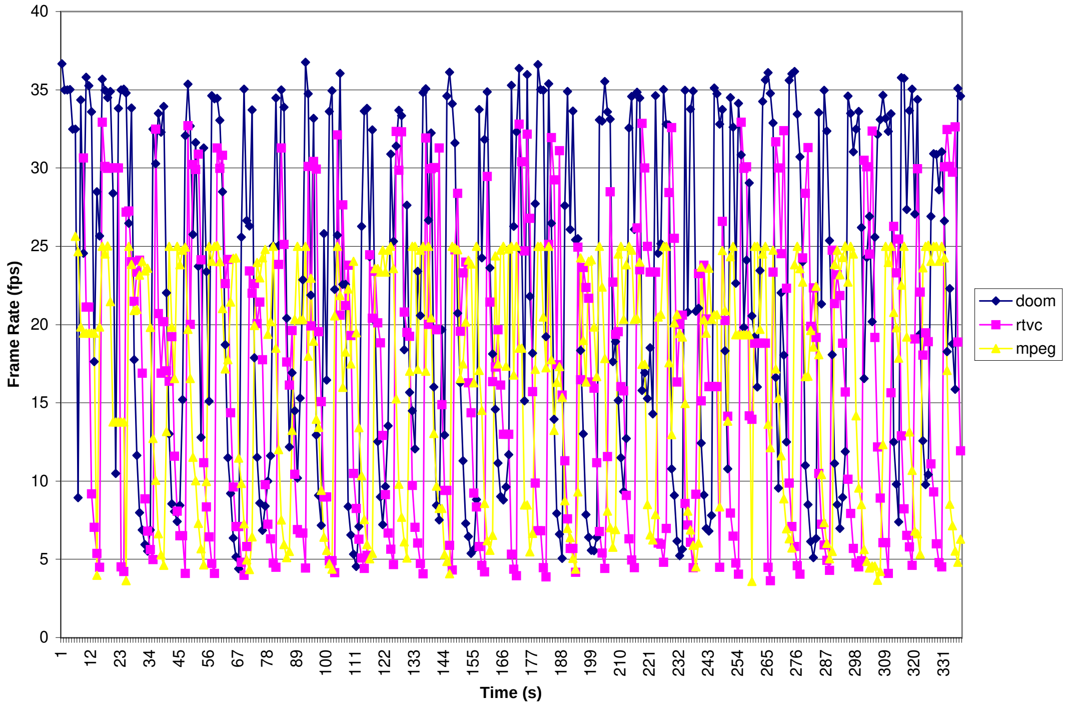

Fig. 3. All Applications with No Management

In the third experiment, we introduced a load on the local processor. A locally developed load generator was started before the applications which created and maintained a run queue of 6. The "load" measured used the systems load monitor and represents an average of the run-time queue length.

Even with the load, the prototype manages to allocate resources to rtvc and mpeg. The results are shown in Figure 5.

Once rtvc terminates, doom is able to grab more available resources. Interestingly, there is less variability with load on the system than without (Experiment 2). We speculate that this is because the prototype is forced to continue to control resources with a load in place than without, where it "lets go".

Experiment 4: Multiple QoS Demands

As a final experiment, we considered the situation in which more than two applications were executing and which demanded QoS. In this case we created somewhat artificial situation in which two "users" would try to run the video player application and the game on the same system. The QoS associated with each mpeg application was the same, namely, a frame rate of between 23 and 27 frames per second. Each game was treated as "best effort". One user (user a) also ran the video conferencing application. As with the previous experiments, the prototype had an administrative policy which favoured the video conferencing application. The results are depicted in Figure 6.

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

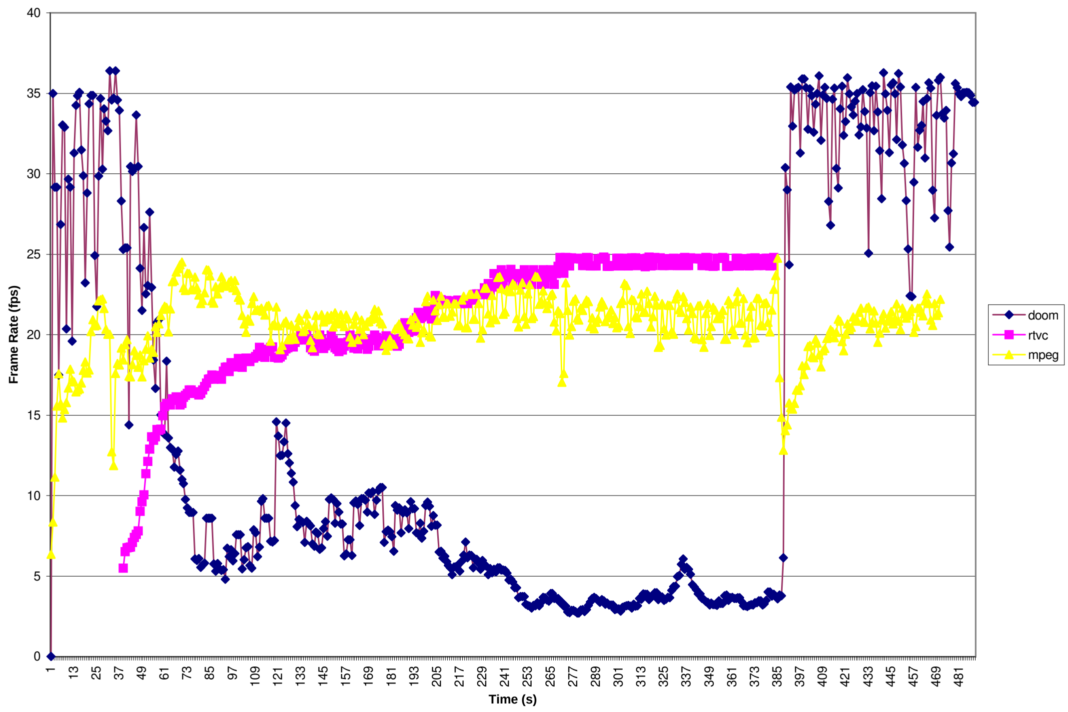

Fig. 4. All Applications in Managed Environment

The prototype was able to allocate resources to meet the QoS requirements of rtvc. This meant that both mpeg applications and both doom applications were given less resources. As the graph illustrates, the frames per second for both mpeg applications was around 10-15 frames per second. Since doom is "best effort", the prototype also allocated more of the resources to both mpeg applications, leaving doom to cope with less computing resources.

Once the video conferencing application terminated, the system allocated more resources to both video player applications bringing to about 23-24 frames per second.

8 Conclusion and Future Work

The results of the experiments with the prototype are promising. First, the prototype was able to manage resources to meeting the QoS requirements of multiple applications on a single host, even under loaded conditions. Enforcement policies were successfully used for differentiating between applications. This is significant in that it shows that it is possible to manage resources on behalf of all applications running on a host, server or across a network. This allows administrators to determine their goals in optimising resource allocation to applications. These goals can be mapped to administration policies. As seen in the CLIPS rules, if the service differentiation rule is applied, the thresholds of the policy attribute are scaled back. This requires that the QoS Manager send a message to the coordinator which causes the coordinator to invoke a method in the sensor that causes

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

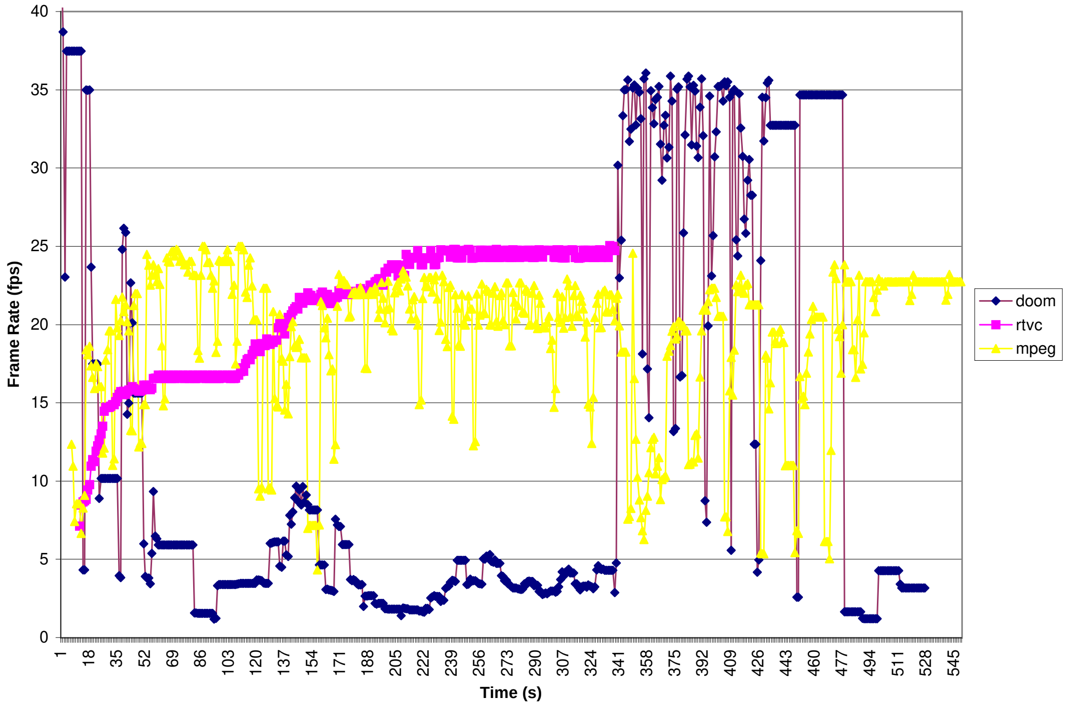

Fig. 5. Managed Environment with Load

a scale back of the threshold. This has the effect of reducing the resource needs of the application which is a form of application adaptation. Other application adaptations are possible and more discussion of how our work supports other adaptations can be found in [10]. Although not shown in this paper, we are able to diagnose between client and server machine overloads and affect the CPU allocation on the server machine.

Second, our work supports satisfying QoS requirements in a dynamic environment as follows: (i) QoS requirements for an application can be retrieved by the application when the application starts up. When the application starts executing it registers with a *policy agent* process. The application process passes information about that process that is relevant in determining the expectation policies that are applicable to that process. This includes a process identifier, an application identifier, an executable identifier and a role identifier. The policy agent uses this information and maps it to the appropriate policies (which is retrieved from an LDAP directory). This is sent to the coordinator component that creates a list of policy objects that it maintains. More information can be found in [13].

Third, in our work neither the user nor developer needs to have intricate knowledge of the hardware architecture or system software. We have instrumented a number of third party applications besides those described in the experimental part of the paper. This includes the Apache Web Server. The only knowledge needed was the name of the probes and knowing which libraries to link in. It was not necessary for the instrumentor to have any knowledge of what sort of QoS management was

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

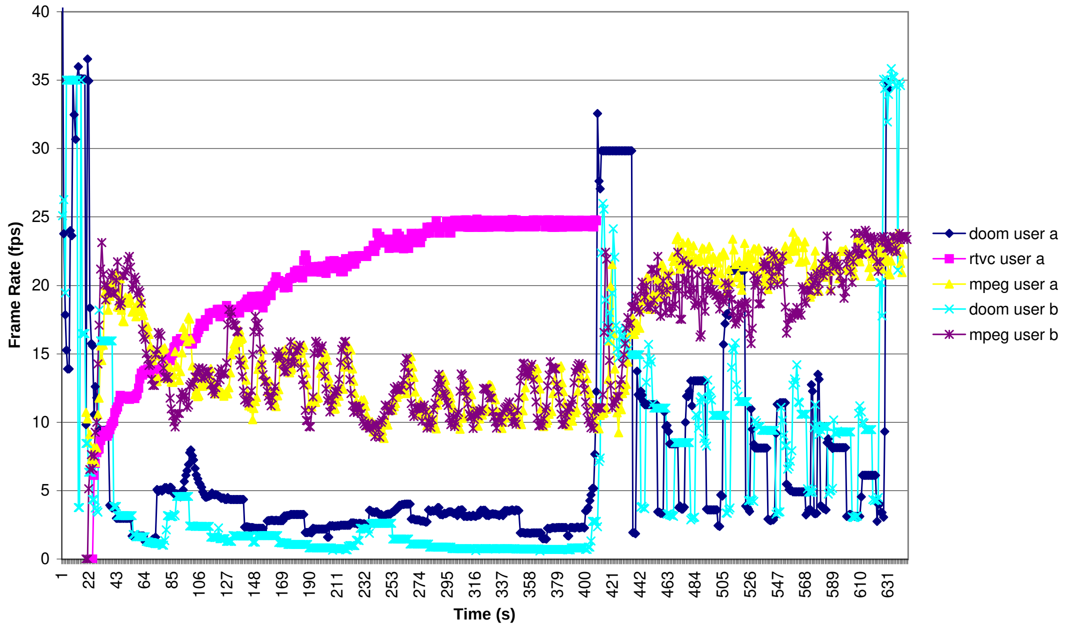

Fig. 6. Multiple Applications with Same QoS Requirements

taking place.

Fourth, our work can be extended to support other languages. We are currently developing an instrumentation library for Java software. The instrumented Java programs will be able to communicate with the QoS Host Manager.

Future work includes the following:

- The prototype focussed on one resource. We will consider other resources, such as memory, network bandwidth and input-output throughput. We currently have implemented a resource manager for memory and for networking. These will be integrated into the current prototype. This raises many questions in terms of how different resource managers would interact.
- We will further study the prototype across multiple systems. There are two types of studies: When the multiple systems are in the same administrative domain and when they are in different administrative domains.
- The CLIPS rules are relatively difficult to specify. We have used a formalism based on [5] for specifying expectation policies (for more detail see [13]) and we are currently writing specifications for enforcement policies and a translator that takes the enforcement policies and translates them to CLIPs rules. This should make it easier for administrators to specify enforcement policies.
- In our work, we found that enforcement policies across different hosts to be difficult to define. We will study the informal policies used at our university to give help better formulate reasonable enforcement policies.

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001

References

1. C. Aurrecoechea, A. Campbell, and L. Hauw. A Survey of QoS Architectures. *Multimedia Systems Journal, Special Issue on QoS Architecture*, May 1998.
2. G. Beaton. A Feedback-Based Quality of Service Management Scheme. *Proceedings of the Third International Workshop on High Performance Protocol Architectures*, Uppsala, Sweden, June 1997.
3. J. Bruno, J. Brustoloni, E. Gabber, M. McShea, M. Ozden, and A. Silberschatz. Disk Scheduling with Quality of Service Guarantees. *Proceedings of the 1999 IEEE International Conference Multimedia Computing and Systems*, Florence, Italy, June 1999.
4. H. Cho and A. Seneviratne. Dynamic QoS Control without the Knowledge of Resource Requirements. *Submitted to IEEE Trans. Computing.*
5. N. Damianou, N. Dalay, E. Lupu, and M. Sloman. Ponder: A language for specifying security and management policies for distributed systems: The language specification (version 2.1). Technical Report Imperial College Research Report DOC 2000/01, Imperial College of Science, Technology and Medicine, London, England, April 2000.
6. I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation. *Proceedings of the Eight International Workshop on Quality of Service (IWQOS 2000)*, pages 181–188, June 2000.
7. N. D. Georganas. Multimedia Applications Development: Experiences. *Journal of Multimedia Tools and Applications, Volume 4, Number 3*, May 1997.
8. J. Giarratano and G. Riley. *Expert Systems: Principles and Programming.* PWS Publishing Company, 1998.
9. A. Goel, D. Steere, C. Pu, and J. Walpole. Adaptive Resource Management via Modular Feedback Control. *Tech. Report. 99-03, Oregon Graduate Institute, Computer Science and Engineering*, Jan 1999.
10. M. Katchabaw, H. Lutfiyya, and M. Bauer. Using User Hints to Guide Resource Management for Quality of Service. *The 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, July 1999.
11. B. Landfeldt, A. Seneviratne, and C. Diot. User Services Assistant: An End-to-End Reactive QoS Architecture. *Proceedings of the Sixth International Workshop on Quality of Service*, Napa, California, May 1998.
12. B. Li and K. Nahrstedt. QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications. *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000)*, 2000.
13. H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. Bauer. Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework. *2nd Workshop on Policies in Distributed Systems and Networks*, pages 461–468, January 2001.
14. B. Moore, J. Strassmer, and E. Elleson. Policy core information model – version 1 specification. Technical report, IETF, May 2000.
15. S. Rampal, D. Reeves, and I. Viniotis. Dynamic Resource Management Based on Measured QoS. *Technical Report TR 96-2, North Carolia State University*, 1996.
16. H. Sariowan, R. Cruz, and G. Polyzos. Scheduling for Quality of Service Guarantees via Service Curves. *Proceedings of the 1995 International Conference on Computer Communications and Networks*, Las Vegas, Nevada, September 1995.
17. R. Schantz, J. Zinksy, J. Loyall, R. Shapiro, and J. Megquier. Adaptable Binding for Quality of Service in Highly Networked Applications. *SSGRR-2000, International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet*, August 2000.
18. Y. Snir, Y. Ramberg, J. Strassner, and R. Cohen. Policy framework qos information model. Technical report, IETF, April 2000.
19. Startdust.com. Introduction to qos policies. Technical report, Stardust.com, Inc., July 1999.
20. 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.
21. J. Strassner, E. Ellesson, B. Moore, and Ryan Moats. Policy framework ldap core schema. Technical report, IETF, November 1999.
22. H. Zhang and S. Keshav. Comparison of Rate-Based Service Disciplines. *Proceedings of SIGCOMM'91*, Zurich, Switzerland, September 1991.

Appeared in the Proceedings of the Fourth IFIP/IEEE International Conference on Management of Multimedia Networks and Services, Chicago, Illinois, October 2001