QoS Management Scheme for Multimedia Applications in Best Effort Environments

Anne Fladenmuller[a], Eric Horlait[a], Aruna Seneviratne[b]

[a] *Laboratoire MASI, Université Pierre et Marie Curie*

4 place Jussieu, 75252 Paris Cedex 05, France

[b] *School of Electrical Engineering, University of Technology, Sydney*

P.O.Box 123, Broadway, NSW 2007, Australia

e-mail: Anne.Fladenmuller@masi.ibp.fr

Abstract

Management schemes for multimedia applications typically fall into two categories. Those which rely on central manager to reserve resources for each application and those that use application based adaptation to deliver a reasonable quality of service in best effort environments. The problems associated with these two approaches are discussed in this paper and some experiments have been done to study the behaviour of applications in best effort environments. Therefore we have implemented two sample adaptive applications and given a generic description of how to build them. Our experiments about behaviour of adaptive applications show the undeterministic and unfair repartitioning of resources among these applications. This leads to the necessity of a central management scheme for these applications. We present a model for such a scheme and provide results to show its validity.

Keywords: Multimedia, Quality of Service, Management, Adaptivity.

1 Introduction

The requirements of multimedia applications can be defined in terms of their required Quality of Service (QoS). Currently there appears to be two approaches to managing the QoS of these applications. In the first approach, before an application can run it interacts with a QoS Manager which checks whether there are sufficient resources available to ensure a minimum QoS level. If there are enough resources, the QoS manager reserves these resources and launches the application. Thus, with this scheme the QoS manager needs to track the usage and availability of resources. In addition, it needs to map the application's QoS requirements into the available system resources. A number of schemes based on this model has been proposed in the literature [1, 2, 3].

As the resources are reserved in the above scheme, if changes in their availability occur[1], the QoS manager needs to renegotiate the QoS levels, this can be an involved and time consuming exercise. That can be overcome by incorporating the ability to detect or infer changes in the operating environment, and the changes that need to take place in response to these variations, into the application. A number of implementations based on this principle have also been reported [4, 5, 6]

Proponents of the first scheme [7] argue that adaptive applications are unstable, complex and cannot completely deal with fluctuations in available resources. The proponents of the second schemes [8] argue that adaptive applications provide the highest fidelity, graceful degradation and are robust. We believe that a hybrid architecture which supports centralised management of QoS coupled with application based adaptivity can overcome the limitations of both the above schemes.

[1] this will be common in mobile environments

2 Different approaches for QoS management

2.1 Management with or without resources reservation

Several approaches consider multimedia management either with or without QoS guarantee. QoS guarantee ensures users that applications are started only if a minimum quality can be provided. Quality thresholds are determined by application programmers and transmitted to a QoS manager. A negotiation phase between applications and a QoS manager determines whether the specified quality threshold can be guaranteed. This negotiation phase starts with a mapping of the application's QoS specifications that define applications communication profiles. These profiles determine the amount of system and network resources corresponding to applications desirable or minimum QoS. The negotiation phase ends when it is ensured that there are sufficient resources available to run the application and after these resources have been reserved. So, the application has a guaranteed service quality. In this scheme the QoS management is done by the application for a QoS specification phase, as well as by the underlying system and network. If resource reservation can be done, such a scheme would work very well. However most current networks and operating systems do not provide resource reservation facilities.

Other available schemes attempt to match the applications QoS to their operating environment. These schemes are best suited to situations where resource reservation is not possible or limited. Since no resource reservation is performed, the QoS negotiation phase is simpler. The QoS specification, mapping of parameters, and communication profile determination are no longer necessary. The management of QoS is then performed by the applications themselves. In this approach, the application programmer selects the rules for adaptation of their applications. No further control for QoS management by the

underlying support system seems necessary. As applications programmers are the best people to determine which services are the most appropriate for their applications and what quality will be expected by the users, this approach seems to be a very sensible one.

In this paper we are focusing on QoS management when no resource reservation is possible. Section 3 explains how an application can adapt its execution to match the system load and shows the risks that applications can face when taking blind decisions about their environment. In section 4, we will present a hybrid management framework and describe the implemented prototype.

3 Adaptive applications

3.1 Creation of adaptive applications

Adaptive applications basically change their execution pattern to suit the operating environment. Therefore applications must be capable of detecting the changes in their environments. This can not be done at the system level because the system would need to know every applications needs. With an effectively infinite variety of applications this is not realistic. Classing the needs of each application by the media they use is also inappropriate. The video play-back and video editor example provided in [6] illustrates this clearly. The main requirement of the video player is to keep good synchronisation between voice and video, while the video editor expects precise editing and therefore needs a guarantee that every frame is seen by users. So, applications can use the same media and have different requirements. The expected degradation scheme should then be different depending on applications rather than on media. Our conclusion is that a single degradation policy can not satisfy the requirements of diverse applications.

Each application has its own preferences about which QoS parameters are the most important. It can be the video size, the frame rate, the synchronisation between two media, etc. Because of the fluctuations of resources in best effort environments, application programmers may prefer to privilege their most important QoS parameters to provide the best possible quality to users. For example, they might decide to display a movie in black and white rather than in colour to get a good frame rate. We define a service level as a set of QoS parameters with a fixed value. If we consider the previous example, we would find two service levels: black and white video or colour video, and the frame rate would determine the quality of each service level. The quality can be good or bad. Applications can degrade their service level to get a higher quality or upgrade it if it can afford a good quality at a higher service level.

Applications determine when to change their service level by monitoring their most important QoS parameters. If these parameters are above or below certain thresholds, the applications upgrade or degrade their service levels to provide a better quality to the users. The control of QoS parameters is performed after a fixed cycle in the application. As most adaptive applications use continuous media, it is easy for application programmers to make a control at every application cycle (display of a frame, synchronisation point, etc). Thresholds can be defined either by applications programmers or users, depending on the application interface.

3.2 Advantages of running adaptive applications

The approach adopted with adaptive applications has several advantages. As adaptive applications monitor their own quality, they can react dynamically to environment changes. This is particularly useful in non stable environments like the ones relying on mobile

computers or more generally best effort environments.

Moreover, Modifications of application execution patterns are defined in the application degradation/up-gradation schemes. These schemes are determined by the application programmers to match with applications requirements and users expectations. As application programmers are aware of their application requirements, they are the best people to define the most appropriate services and their minimum and maximum expected QoS.

When an application detects a modification of its quality, it calls a degradation/upgradation script. These scripts define the execution modification policy adopted by an application by specifying the service level that should be chosen to keep a good quality. A decrease of service level reduces the amount of system or network resources used by the application. Thus, applications are aware of their environment and do not try to overload the system by getting the highest service. They adapt their service to match to the system or network load. Such an approach seems appropriate to best effort environments. Nevertheless, as the concept of adaptive applications is fairly new and no real study has been done to study their behaviour, we have decided to run experiments about their execution. Therefore, we have implemented two different samples of adaptive applications. In the next section we are going to describe the execution pattern of these two applications.

3.3 Some examples of adaptive application

The first adaptive application implemented to run our experiments is a simple prototype sensitive to system load modifications. For this program, five service levels have been defined, each corresponding to a program priority. The lowest (respectively highest) service level corresponds to the lowest (respectively highest) program priority. This priority can be modified dynamically using the **nice** unix command, when a significant modifica-

tion of the available system resources is detected.

In order to detect system load modifications the program executes the same function several times and measures the interval of time for each execution. As other programs can run concurrently to ours on the system, the execution time of this function varies. The program compares the *i*-th execution of the function E_{i-1} with the (*i-1*)-th E_{i-1} and detects increases or decreases of execution times. If another program uses more resources, the execution time of our function increases $E_i > E_{i-1}$. Then, the model measures a decrease in available system resources and it decreases to a lower service level by reducing the program priority. For our simulations, we have considered execution time modifications as significant if they reach 20 percent of the previous execution time.

The second application is an adaptive Mpeg-play. This application modifies the video display to keep a reasonable frame rate when dynamic system load modifications occur. A significant modification of the environment changes the service level of the application. For adaptive Mpeg-play, each of the 3 defined service levels corresponds to a given video size and colour. The highest service level corresponds to a big format colour video, the middle one to a small format of colour video and the lowest one refers to a small format of black & white video, as shown in Figure 1.

The program averages the display time of n mpeg-play pictures A_n to cope with the difference of time needed for decoding the different types of frames. To each frame i (with $i > n-1$) corresponds an average display time A_{ni}. This average display time is expected to stay in a range $A_{MAX} > A_n > A_{MIN}$. When A_{ni} becomes greater than A_{MAX} the program detects some new resources are available and the service level is increased when it is not the maximum one already. If A_{ni} becomes lower than A_{MIN} the program considers the quality is not sufficient, so the service level is decreased when it is not the lower one.

For the experiments we set n at 10, the maximum averaged frame rate at 15 images per second and the minimum averaged frame rate to 5.

3.4 Experiments

We were interested in studying the behaviour of adaptive applications running concurrently on the same system. Therefore we have run concurrently our two adaptive applications and compare the service level of both [14] applications. The experiments show it is not possible to predict the evolution of adaptive applications. Moreover when there is a lack of resources, it is possible for one adaptive application to decrease its service level whereas another running concurrently would improve the service given to its users. Thus one application could benefit from the degradation of another.

As a conclusion of these experiments (cf Figure 2), it can be seen that without a system level management of some form, adaptive applications can unfairly distribute the available resources. To avoid these problems, we present in the next section a hybrid management model.

4 A hybrid model for applications management

The model has two components, the adaptive applications supervisor and the admission control entity.

The supervisor prevents applications from taking "blind" decisions about the modification of their execution pattern. Its role consists only in authorising or forbidding adaptives applications from changing of service levels. It answers the degradation or upgradation request sent by the applications. Thus, the supervisor will tend to make all running adaptive applications converge to a common service level. We want to demonstrate in the next

section that the creation of a supervisor reduces problems caused by non-deterministic behaviour of such applications and unfair distribution of resources among adaptive applications.

Nevertheless, the supervisor does not guarantee that once an application is started, it can be run at its minimum service with reasonable quality. When there is not enough resources to run all of the adaptive applications at the desired service level, all applications will converge to a degraded service level. When the amount of resources is not sufficient to continue working correctly at its lowest service level, the application degradation policy might decide to "auto-kill" itself or will continue running with a very poor quality. If several adaptive applications are running concurrently, it seems unfair to have one chosen arbitrarily to exit from its execution, or have all applications running in low quality with degraded mode. Such a situation can easily arise on current multi-user systems, since users can start new applications whenever they want, in an environment with no resource reservation and QoS guarantee are provided (hypothesis for the framework).

To compensate this unfair situation, we have also defined an admission control mechanism to monitor the number of adaptive applications running on the system. So, before an adaptive application can commence, the admission controller has to determine if it will unacceptably degrade the quality of currently working adaptive applications. The admission control mechanism is described in section 4.2.

An overview of the structure of this system is given in Figure 3.

4.1 Adaptive application supervisor

The concept of adaptive applications seems well suited to multimedia environments as these applications are very demanding in terms of resources. Nevertheless, as explained

in the section 3.4, our experiments showed that adaptive applications are not capable of managing their resources in isolation. There needs to be some sort of global control to prevent applications from taking "blind" decisions. This can be performed by a *Adaptive Application (AA) Supervisor*. The role of this supervisor is to make applications converge to the same service level.

Therefore a control is introduced within the *AA supervisor*. Each application, when it detects a notable changes of their QoS, requests the *AA supervisor* an authorisation to degrade or upgrade its service level.

The *AA supervisor* keeps track of the quality level of every adaptive application and determines which ones can have their service levels modified. Whether an application can have its quality level improved or degraded is determined by its current quality relative to the other applications. To be fair, only applications with the higher service level can be degraded, and only applications with the lower service level can be upgraded. In other cases, the request is refused by the *AA supervisor* and no modification is done by applications.

The *AA supervisor* interface provides a range of five service levels (*best, very good, good, average, tolerable*). Each application programmer can choose between this range of service levels and associate to some or all of these levels the applications characteristics for this service level.

Communication between the adaptive applications and the *AA supervisor* is done using messages. The types of messages used are:

- *initialisation* from the application to the server,
- *modification* to specify either a modification requested by the application, or the result of this request by the *AA supervisor*,

- *termination* to notify when an application has finished its execution.

Results of simulations (cf figure 4) run on the same Unix system with the *AA supervisor* gives very good results. In all our experiments, with several (from 2 to 5) adaptive applications, we obtained a convergence of the adaptive applications service levels, and so a fairer repartitioning of resources.

Another problem arising with adaptive applications is the feedback control mechanism introduced when changing dynamically from one service level to another. Each application programmer has to make the detection of modification sensitive enough to obtain a good quality, but not too sensitive to avoid too frequent changes and provoke an oscillation in the upgradation and degradation process. The use of this *AA supervisor* reduces such risks by permitting only some of them to degrade and upgrade their quality.

Nevertheless, if too many applications are run concurrently, it is possible that all of these may have a poor quality. There might not be enough resources to support all the applications at the lowest service level. In the next section we present a way of reducing this problem.

4.2 Admission control mechanism

We present the admission control model we are currently implementing. It is composed of four different entities the *statistics module*, *admission controller*, *AA supervisor* and the *resources manager*.

To determine whether an application can start, the admission controller has to be aware of resources needed by this application and resources used by other applications. As our model is based on the assumption that resources reservation is not possible in our environment, no specification of application needs is required by the system. In this model,

we retain the QoS management in the applications themselves, and so avoid having QoS specification phase followed with a QoS mapping to determine needs in term of system and network resources for these applications. Instead we keep track of applications needs, by making statistics about their resources usage.

The usage of a supervisor and admission controller system has to be portable on every different computer, and can be used for a remote negotiation. In such context, the admission controller has to be aware of the needs of adaptive application on a remote and sometimes different system. Therefore, the interface to access data about resources usage has to be transparent on every system. We have chosen to use management information bases (MIBs) to read and write such information. An SNMP agent is used to control and update management information bases about current system resources usage and adaptive applications needs at their lowest service level.

The protocol of the admission control mechanism is as follows. An application sends a message to the *AA supervisor* and waits for a positive acknowledgement from it to start running. The supervisor warns the admission controller of which application wants to start, then the admission controller checks if some other adaptive applications are currently running at their lowest service level. If so, it considers the system is overloaded and refuses to permit the new adaptive application to run. Otherwise it looks in the application MIB and looks at the minimum resources required by the application (R_n^{MIN}), where n specifies this adaptive application would be the n^{th} running on the system. Then it checks the amount of resources used by all currently running applications ($\sum_{i=1}^{n-1} R_i$) by looking in the resource MIB. This MIB is updated by the resources manager. The admission controller looks also in the application MIB to get the sum of minimum of resources of all currently running adaptive applications ($\sum_{i=1}^{n-1} R_i^{MIN}$) . If the difference between the

amount of resources used by the currently running adaptive applications and the resources they would use at their lower service level is greater than the minimum of resources required by the new application (cf Equation 1), this application is started.

$$\sum_{i=1}^{n-1} R_i - \sum_{i=1}^{n-1} R_i^{MIN} > R_n^{MIN} \quad (1)$$

When an application is not referenced yet in the application MIB, it is accepted and statistics are run to get information about its requirements. As soon as it is started, information about its requirements is collected in case another application would like to start. This is done by starting each adaptive application at their lowest service level.

We have implemented this model with management of CPU load. Some further work should be done to enlarge the scope of the resources monitored. We can not yet provide results of this implementation but we intend to do it very soon.

This model is portable and the overhead of the establishment should not be very large.

5 Conclusion

In some environments where no resources reservation can be done, a good scheme consists of giving applications the possibility of modifying their quality to make the best use of the available resources. Therefore we have explained in this paper the necessity of these adaptive applications for such environments.

Nevertheless, running such adaptive applications leads to unfair resources repartitioning. Our observations shows that the absence of a global *AA supervisor* for adaptive applications makes the environment even more unstable. It becomes more difficult to predict the load of the system and a fair repartitioning of the resources between applications will not occur. We have written a *AA supervisor* and have had some good results for the

management of different adaptive applications.

To provide applications with a minimum quality, it seems reasonable to limit the number of demanding applications. Therefore we have added in our model an admission control mechanism. We have also described in this paper the implementation model for the admission control and we intend to study the cost of adding such a component to our future experiments.

References

[1] A. Campbell, G. Coulson, D. Hutchinson "A Quality of Service Architecture", ACM Computer Communication Review, Vol. 24, No. 2, April 1994.

[2] A. Seneviratne, M. Fry, V. Withana, V. Saparamadu, A. Richards, and E. Horlait, "Quality of service management for distributed multi media applications", in Proc. IEEE Communication Society Phoenix Conference on Computers and Communications, Phoenix, Arizona, 1994.

[3] L. Besse, L. Dairaine, L. Fedaoui, W. Tawbi, K. Thai, "Towards an Architecture for Distributed Multimedia Applications Support", in Proc. International Conference on Multimedia Computing Systems, Boston, MA, May 1994.

[4] S. Deering, "Reservations or No Reservations ?", INFOCOM'95 Panel Session, Boston, MA, April 1995.

[5] N. Davies, G. S. Blair, K. Cheverst, and B. Friday, "Supporting Adaptive Services in a Heterogeneous Mobile Environment", in Proc. Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December 1994.

[6] B. D. Noble, M. Price, and M. Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing", in Proc. 2nd USENIX Symposium on Mobile and Location Independent Computing, Michigan, April 1995.

[7] D. Ferrari, "To Share Rather Than to Pay", INFOCOM'95 Panel Discussion, Boston, MA, April, 1995.

[8] C. Huitema, "Application Adaptivity", in Proc. INFOCOM'95 Panel Session, Boston, MA, April 1995.

[9] L. Fedaoui, A. Seneviratne, A. Fladenmuller, and E. Horlait, "Implementation of a End-to-End Management Scheme", in Proc. High Performance Computing Conference, Singapore, September 1994.

[10] A. Seneviratne and H. S. Cho, "Quality of Service Mapping in Distributed Multimedia Systems" in Proc. IEEE MMNet'95, Aizu, Japan, September 1995.

[11] J-C Bolot, T. Turletti, "A rate control mechanism for packet video in the Internet", in Proc. IEEE INFOCOM'94, June 1994.

[12] A. Campbell, G. Coulson, F. Garcia, D. Hutchinson, and H. Leopold, "Integrated Quality of Service for Multimedia Communications", in Proc. IEEE INFOCOM'93, 1993.

[13] K. Patel, B.C. Smith, and L. A. Rowe, "Performance of a Software MPEG Video Coder", Computer Science Division, University of California, Berkeley, http://www.cs.berkeley.edu.

[14] A. Fladenmuller, A.Seneviratne, E. Horlait, "A Hybrid QoS Management Scheme for Distributed Multimedia Applications", in Proc. Second Workshop on PROMS, Salzburg, 9-12 October 1995.

[15] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Structure of Management Information for version 2 of the Simple Network Management Protocol (SNMPv2)", International Telecommunication Union, RFC 1442, April 1993.

[16] M. Rose, "SNMP over OSI", International Telecommunication Union, RFC 1418, March 1993.

Figure Captions

Figure 1 Service levels of adaptive Mpeg_play

Figure 2 Two adaptive applications running concurrently

Figure 3 Implementation scheme for an end-to-end management

Figure 4 Role of the AA supervisor with 2 adaptive applications running concurrently

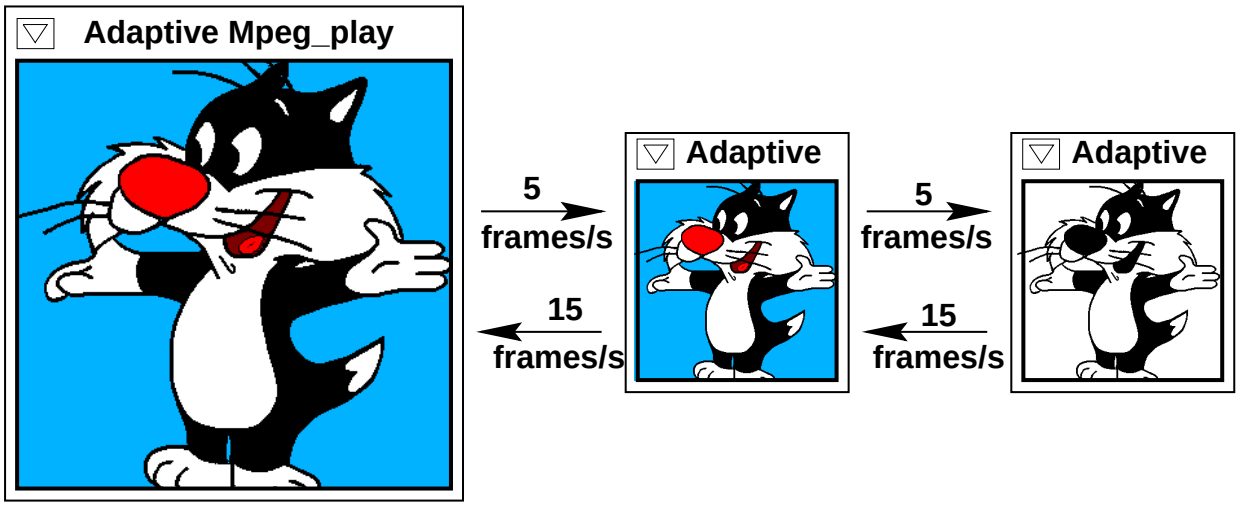

Figure 1:

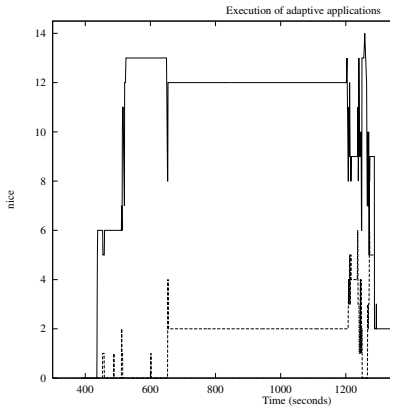

Figure 2:

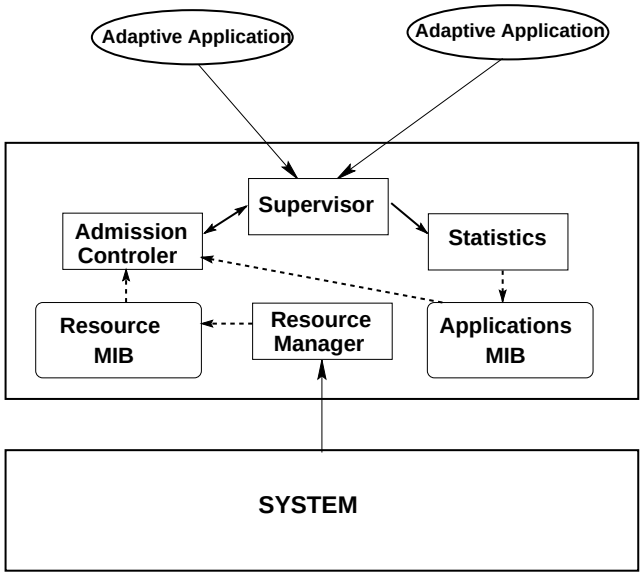

Figure 3:

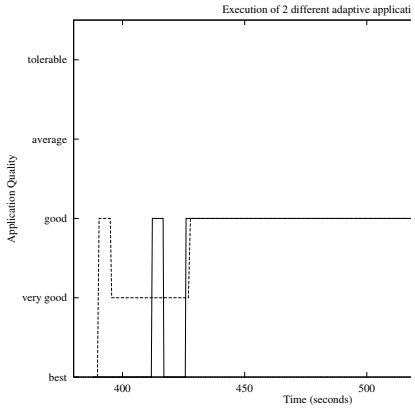

Figure 4: