QoS Management of Distributed Multimedia Applications in Best Effort Environments

Anne Fladenmuller
e-mail: fladenmuller@masi.ibp.fr
Laboratoire MASI, Université Pierre et Marie Curie
4 place Jussieu, 75252 Paris Cedex 05, France

1 Introduction

The requirements of multimedia applications can be defined in terms of required Quality of Service (QoS). Currently there appears to be two approaches to managing the QoS of these applications depending on whether you can reserve resources or not.

The first approach ensures users have a guaranteed service quality. Therefore resources are to be reserved at a system and network level. A number of schemes based on this model have been proposed in the literature [1, 2]. These schemes are appropriate when resources reservation is possible, nevertheless most current operating environments do not provide such facilities. So an second approach of QoS management has been proposed if no global end-to-end QoS guarantees can be provided. Such scheme is necessary to give users the best possible services. This approach considers applications have to be aware of their environment and react dynamically to provide the best quality according to the available resources. This is the solution adopted with adaptive applications.

Proponents of this first scheme [3] argue that adaptive applications are unstable, complex and cannot completely deal with fluctuations in available resources. The proponents of the second schemes [4] argue that adaptive applications provide the highest fidelity, graceful degradation and are robust.

2 Adaptive applications

Adaptive applications basically change their execution pattern to suit the operating environment. So, when changes in the availability of resources occur, adaptive applications detect them and react according to their degradation and up-gradation scripts. Different services are provided to users, depending on the amount of available resources. These different services are characterised by a set of QoS parameters that represent a *service level*. Adaptive applications will tend to give the most appropriate service according to the available resources, and therefore will monitor the quality provided for its service level. If the quality becomes too poor the application degrade to another service level to get a better quality. For example, an adaptive video application might decide to display a movie in black and white (its *average service level*) rather than in colour (its *good service level*) to get a good frame rate, which would define the quality criteria for this application. If the QoS parameter that characterises the quality goes above or below certain thresholds, the applications upgrade or degrade their service level.

The approach adopted with adaptive applications presents 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 the best effort environments. Moreover, the application programmer determines the particular degradation or up-gradation scripts to match with application 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.

3 Need of management for adaptive Applications

Execution of adaptive applications provide satisfying results, nevertheless concurrent execution of several adaptive applications with no global management causes unfair and undeterministic repartition of resources among these applications [5]. For example, in case of a lack of resources, one application can decide to degrade its service. Another adaptive application running concurrently will then detect the improvement caused by this reduction of resources utilisation and will increase its service level. This can be repeated until one application gets all the resources and the other one is forced to terminate.

Also the interaction between adaptive applications can force one or more to oscillate rapidly between service levels. Thus, the feedback control introduced by these adaptive applications can make the environment very unstable and can introduce an unfair repartition of resources among these applications. Therefore, we have proposed in [5] a management scheme to make the environment more stable and allow a fairer repartition of resources. This scheme is composed of two components, the adaptive application (AA) supervisor and the admission control entity.

The *AA supervisor* prevents applications from taking "blind" decisions when modifying their execution. In case the system load increases, the supervisor will tend to make all running adaptive applications converge to a common service level. When an application detects a noticeable change in its QoS, it requests an authorisation to the *AA supervisor* to degrade or upgrade its service level. Whether or not this authority is granted, is determined by the service level of the requesting application relative to all 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 the application doesn't change of service level. Results of our experiments [5] have shown that concurrent executions of adaptive applications run with the *AA Supervisor* causes the applications to converge to the same service level. Thus, applications benefit from a fairer repartition of resources. Moreover such approach also reduces fluctuations in the amount of resources allocated to each application.

The admission control phase will permit to guarantee users a minimum service level. When there are insufficient system resources to run all of the adaptive applications at the desired service level, all applications will run at a degraded level. Actually, the degradation policy of the application might even decide to "auto-kill" itself, rather than continue with an unacceptable level of service. To compensate this lack, before an application can commence, the admission controller determines if it would unacceptably degrade the quality of other existing adaptive applications. The implementation model is described in details in [5].

4 Lacks of the implemented model

The notion of a fair repartition of resources adopted in the previous model, becomes inadequate when one adaptive application requires more resources for a lower service level than another one. For example, a adaptive video application would require 80 percent of the CPU time to provide a good service level, whereas an audio one would need only 20 percent of the CPU time to provide a very good service level. If the system load becomes important, the fair repartition scheduler will fairly share CPU time between the applications. In this case, the video application would run poorly with its less than its 80 percent CPU time whereas the audio application would run with good quality at its best service level.

According to the *AA supervisor* policy described previously the degradation request sent by the video application would be rejected, because another application would be running at a higher service level. And the video would run with a very poor quality (for example a very low frame rate) at its current service level. To prevent this situation, it is necessary to degrade the service level of the video application and then provide a better quality (for example decrease the pictures size to increase the frame rate).

Another problem appears, if users have preferences within their applications and would like to specify a higher priority for their applications. An application interface has therefore to be implemented by the application programmer. So, we consider the management of applications with different needs, both with and without priority associated with the applications.

4.1 Adaptive applications with no priority

As explained previously, problems can occur when one application wants to degrade its service level while another with a higher service level is running concurrently. To allow the degradation the *AA supervisor* will then have to keep track of the number of degradation requests sent by the application. When this number of requests exceeds a threshold, the *AA Supervisor* will authorise the request. This threshold depends on the service level of the application relative to the other applications. The lower the service level is, the more requests will be required to cause the *AA Supervisor* to issue an authorisation. When the *AA Supervisor* authorises any modification of service level, all the request counters are reset. A similar operation is done for up-gradation scripts.

Another problem appears when considering distributed adaptive applications. In this case, the cause of the application running with a poor service level could be due to CPU or Bandwidth contention with the adopted scheme. The application can not be aware of which of these contention has caused the reduction in its quality.

Thus, the application of degradation script concerning local resources (for example, decoding pictures in black and white rather than in colour) when the problem comes from the network seems pointless. When application programmers design the service levels for their applications, their degradation and up-gradation scripts should take into account alternatively network and system resources.

Such scripts would also permit a partial reservation of resources schemes to be implemented. If the network guarantee the bandwidth allocated for the application, the degradation or up-gradation script will concern only local resources.

4.2 Management of priority applications

More and more users want to have some way of controlling their applications. They usually have preferences on which application they consider to be the most important and they expect to be able to attribute them a higher priority. In the management scheme given previously, equity among applications is the main objective. The introduction of priority between applications does not modify this model, but this issue needs to be handle differently. Problems occur when resources are lacking for applications with the higher priority. In an environment with no possible resource reservation, it is necessary to make the applications with lower priorities degrade their QoS to provide the higher priority application with a better quality.

We have considered two alternatives to address this problem. The first approach consists of forcing the service level degradation of lower priority applications. Thus, in order to make them use less resources and free resources the higher priority applications. The second alternative, is to somehow cut the resources used by the lower priority applications (e.g. with the UNIX nice command), making the freed resources available for the higher priority application. This may cause these applications to request a degradation of service level.

Nevertheless, deallocation of resources is difficult to achieve. First, the management entity would have to be aware of which resources are critical and second it would have to locate the bottleneck (network or system) as well as knowing which application is using these resources. Therefore, we have decided to study the first approach, opting for a simpler solution for the implementation. We intend to adopt the choice of suspending the lower priority applications when resources are lacking and resume their execution when more resources becomes available.

5 Conclusion

Most of current QoS management scheme are not appropriate for best effort environments. We consider here the approach of using adaptive applications for managing QoS in such environments. These applications change dynamically there execution to take into account changes in the availability of resources.

Nevertheless, running multiple adaptive applications leads to unfair resource repartition. An implementation for the management of adaptive application has given good results, but some issues had not been considered yet. So we provide a methodology to implement degradation and up-gradation script for distributed applications. The changes to our original model [5] do not add a significant complexity to our management scheme, and should make the environment more stable.

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

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

[5] A. Fladenmuller, E. Horlait, A. Seneviratne, "An hybrid QoS management Scheme", Sydney, Australia, 11-12 december 1995. Redirected as "QoS Management Scheme for Multimedia Applications in Best Effort Environments" for the Journal of Electrical and Electronics Engineering, Australia.