Adaptive Resource Management in a Multimedia Operating System

Don Oparah
Computer Laboratory,
University of Cambridge,
Cambridge CB2 3QG, UK
Email: doo20@cl.cam.ac.uk

Abstract

This paper highlights the need for adaptive resource management within the endsystem, in addition to the provision of basic QoS guarantees, in order to maximize the users' perceived performance of their applications at any given time within the limits of the physical resources available. A prototype system developed within the Nemesis multimedia operating system [7] is described. The system provides support for the dynamic adaptation of applications according to user-controlled resource allocation policies. The paper also identifies interesting directions for further work in future.

1. Introduction

To support the ever increasing number of multimedia applications being developed, some work has been done on providing the Quality of Service (QoS) required by such applications in both the network and the endsystem. Operating systems work such as [9] and Nemesis [7] (developed locally) provides support for meaningful QoS guarantees. However we make the following key observations

- QoS work in the endsystem has focussed on moving away from the unpredictability of systems like UNIX by providing a predictable level of service to applications via a QoS guarantee. However most multimedia applications are able to adapt in one or more directions (e.g via frame rate, frame size) and operate in a number of different valid modes. Hence the goal should be to provide a predictable level of service *for the current mode of operation*.

- Hardware improvements notwithstanding, application writers design and users inevitably want to run, a mix of applications whose total resource requirements if run in their highest mode of operation would overload the system.

These observations motivate the need for a resource management system integrated with an operating system such as Nemesis to

- Peform a *balancing act* by dynamically reallocating QoS guarantees for resources amongst adaptive applications in such a way that the current combination of applications, operating at different modes, corresponds to the best possible performance *as perceived by the user*.

- Provide a mechanism which allows the user to express resource allocation *policies* or preferences. This relieves the user of the complicated task of manually attempting to degrade other applications in order to support a new application or an application upgrade while still allowing the user to influence the manner in which the reallocations take place.

- Provide support for adaptive applications.

Our view of the progression of research in endsystem QoS is illustrated in Figure 1. We go on to describe the current state of a resource management architecture designed

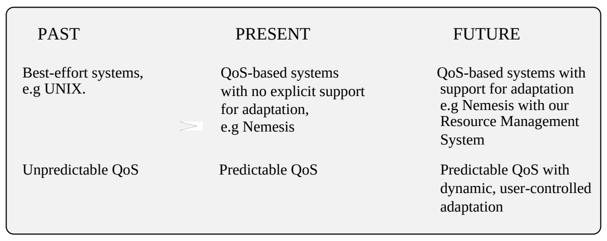
PAST
PRESENT
FUTURE
Best-effort systems, e.g UNIX.
QoS-based systems with no explicit support for adaptation, e.g Nemesis
QoS-based systems with support for adaptation e.g Nemesis with our Resource Management System
Unpredictable QoS
Predictable QoS
Predictable QoS with dynamic, user-controlled adaptation

Figure 1. Endsystem QoS research

to provide such support and focus on a prototype which illustrates the main ideas and provides a basis for further discussion.

2. Architectural overview

A simplified view of our resource management architecture is shown in Figure 2. At the centre of the system is the

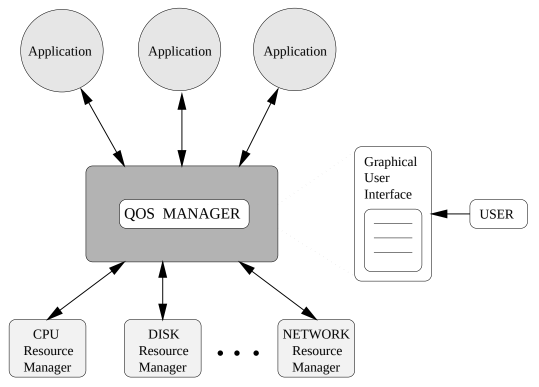

Figure 2. Resource management architecture

QoS Manager (QM) which is a system service responsible for co-ordinating the distribution of shared operating system resources amongst all the domains (Nemesis equivalent of a process) depending on availability and resource allocation policy. In order to get a QoS guarantee for a certain resource or to re-negotiate an old guarantee, an application must make a request to the QM. The QM indicates its allocation decisions to the appropriate Resource Manager (RM) which is the entity that ensures that QoS guarantees are actually enforced via appropriate scheduling.

Adaptive applications can register a number of valid *modes* of operation with the QM at the start of their execution. During situations of resource shortage or unexpected availability (e.g when an application dies and frees up resources), the QM will use this information combined with a knowledge of the user's preferences, expressed via the GUI, to reallocate resources in such a way as to maximize the user's perceived performance or utility. Section 3 describes these mechanisms in more detail via an example.

3. Prototype

The current prototype implementation only fully supports the CPU resource (extensions to support the disk and network resources are being implemented at the timing of writing) and applications only adapt in one direction (i.e via frame rate). However we believe that the system still incorporates enough ideas to show the potential of our work and stimulate discussion. A screen shot from a workstation running our resource management system within the Nemesis operating system is shown in Figure 3.

In the example, there are 4 video display applications each configured to run in one of 3 possible modes 0 to 2 corresponding to 25, 15 and 5 frames per second. The video sources are Fore Systems AVA devices [4] and are connected to a Pentium PC via an ATM newtork. The applications are receiving video via separate ATM VCs (the duplication in pictures was due to a shortage of video cameras at the time the screen shot was taken). The user can request a change in mode by clicking the appropriate mouse button in the bottom control bar of the video window.

The GUI provided by the QM to the user displays the domains currently running in order of *user importance*. A domain's importance can be adjusted by highlighting it and using the `UP/DOWN` buttons. The current CPU allocation for each domain, expressed as a guaranteed slice of CPU time within a given period as well as as a percentage, is also shown. The user can dynamically control the resource allocation policies used for resource revocation and redistribution via the `Edit Policy` pulldown menu.

We assume that the current situation is as shown in Figure 3 with domains RV1 and RV2 in mode 0 (i.e 25fps), RV4 in mode 2 and RV5 in mode 1. The CPU requirements for modes 0 to 2 are 5%, 4% and 2% respectively. These values were obtained by giving the video domain a large CPU allocation and measuring its requirements at different modes (we are currently working on a better method for deriving application requirements). The total CPU allocation is 94% which is the system set limit for the CPU. Hence to support any further requests, the QM would have to consider revoking previously allocated resources.

If the user then requests a mode increase in RV4 up to mode 1 (i.e 15fps), an extra 2% of the CPU must be found to satisfy this request. The resource allocation process then proceeds as follows.

Step 1 Obtain a candidate set, S, of domains for resource revocation. This set would correspond to domains which are less important than RV4. In this case the set is RV2, RV1 and RV5 (see Figure 3).

Step 2 Derive a set of all the possible combinations of mode reductions to the domains in S which would yield the required 2%. In this case there are 4 possibilities as summarized in Table 1. We are assuming that the number of domains involved is small enough to make this calculation feasible in real time. We believe this is a reasonable assumption for a typical workstation environment.

Step 3 Choose one option based on the currently defined user policy. The currently implemented revocation

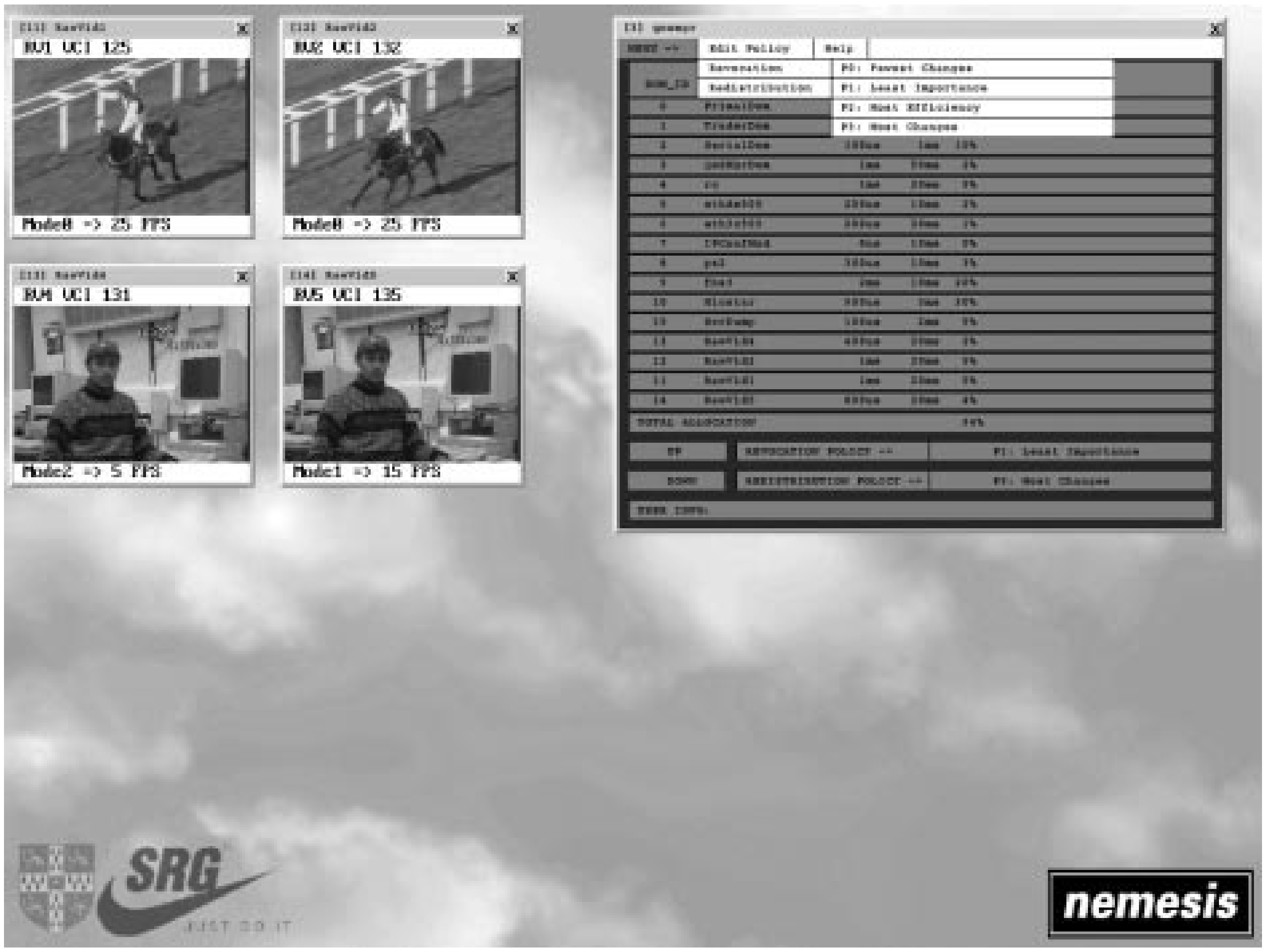

Figure 3. Screen shot of the prototype system

Table 1. Resource revocation options

Option	Domain	Fps change	CPU %
A	RV1	25 > 5fps	3%
B	RV2	25 > 5fps	3%
	RV1	25 > 15fps	1%
	RV2	25 > 15fps	1%
C		Total	2%
D	RV5	15 > 5fps	2%

policies are

- **P0: Fewest Changes**. Pick the option involving changes to as few domains as possible, leading to as little disruption as possible. Options A, B and D would equally qualify in this example and D would then picked because it involves the least important domain.
- **P1: Least Importance**. Pick the option involving the least important domains. This would correspond to option D.
- **P2: Most Efficiency**. Pick the option whose total CPU value is closest to the target value. Options C & D qualify and D would be picked.
- **P3: Most Changes**. Pick the option involving changes to as many domains as possible, leading to a more even distribution of the revocations. Option C would be picked.

The current revocation policy is P1 and hence option D is chosen. A similiar algorithm is used to redistribute resources amongst domains when additional resources become available.

Thus the system gives the user dynamic control over both the relative importance between applications (via the importance list) and the *manner* in which resources are allocated (via the policy options) instead of constraining adaptation to occur based on some built-in system policy which may not always lead to desirable results as far as the user is concerned.

4. Future work

Future work will address the following

- Extension of the implementation to support multiple resources.
- A generic model for adaptive applications to support adaptation in any number of directions, e.g frame size, compression factor and colour depth.
- A generic way of expressing resource allocation policies. We believe that the simple examples of policy described here leave a lot of scope for further investigation. We intend to develop more precise measures of the user's utility, building on work such as [2] and [3], and we feel that due to the user-oriented nature of the work, having a running prototype will assist greatly with the derivation of realistic solutions.
- The integration of this work into a distributed environment, which may in turn lead to new possibilities.

5. Related work

[8] describes a system similiar to ours where *QoS states* are equivalent to our modes. However the kinds of policies defined are different. The idea of *QoS levels* used in [1] is also similiar to the modes concept. The AQUA system [6] supports adaptation within an application via an adaptation function and [5] identifies some of the problems described here but neither of them address policies for allocating resources between applications.

Acknowledgements

The author would like to thank the members of the SRG group, both past and present, for their support during this work. Special thanks to Dr. Ian Leslie for commenting on an earlier draft of this paper.

References

[1] T. Abdelzaher & K. Shin. *End-host Architecture for QoS-Adaptive Communication.* To appear in IEEE Real-Time Technology and Applications Symposium, June 1998.

[2] S. Chatterjee, J. Sydir, B. Sabata & T. Lawerence. *Modelling Applications for Adaptive QoS-based Resource Management.* The 2nd IEEE High Assurance Systems Engineering Workshop, August 1997.

[3] K. Fukuda, N. Wakamiya, M. Murata and H. Miyahara. *QoS Mapping between User's Preference and Bandwidth Control for Video Transport.* 5th International Workshop on Quality of Service, May 1997.

[4] Fore Systems Research. *AVA Owner's Manual* April 1996.

[5] M. Jones, P. Leach, R. Draves & J. Barrera. *Modular Real-Time Resource Management in the Rialto Operating System.* 5th Workshop on Hot Topics in Operating Systems, 1995.

[6] K. Lakshman, R. Yavatkar & R. Finkel *Integrated CPU and Network-I/O QoS Management in an Endsystem.* 5th International Workshop on Quality of Service, 1997.

[7] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns & E. Hyden. *The design and implementation of an operating system to support distributed multimedia applications.* IEEE Journal on Selected Areas In Communications 14, 7, 1996.

[8] N. Nishio & H. Tokuda. *Simplified Method for Session Coordination Using Multi-level QOS Specification and Translation.* 5th International Workshop on Quality of Service, 1997.

[9] R. Rajkumar, K. Juvva, A. Molano & S. Oikawa. *Resource Kernels: A Resource-Centric Approach to Real-Time and Multimedia Systems.* The SPIE/ACM Conference on Multimedia Computing and Networking, 1998.