An Economic Approach to Adaptive Resource Management

Neil Stratford and Richard Mortier*

November 1998

Resource management is a fundamental concept in operating system design. In recent years it has become fashionable to consider the problem as an aspect of heterogeneous support for Quality of Service ('QoS'). Several authors have advocated the construction of an "oracle"-like entity, with the effect of abstracting the fundamental problems into oblivion. In this paper we propose a radically different approach that attempts to address the underlying issues in a uniform and fundamentally scalable manner.

1 Motivation

Resource management is the task undertaken by an operating system to provide timely and correct allocation of limited resources to applications, according to some user defined policy. Considering this as an optimisation task, one can see that the operating system has the dual goals of global (system) and local (application) optimisation across multiple resources.

Global optimisation corresponds to running an efficient system, with the aim of maximising the user's utility, where a user's utility is considered to be the sum of the utilities of each of their applications. In the case of a multi-user system, we consider the global optimum to be similarly the sum of the optima of each user. *Local optimisation* is performed per-application, and attempts to enable an application to provide the highest quality output possible whilst fulfilling the user's requirements, and remaining within the constraint of the system's finite resource.

Managing the system's resources in order to achieve global and local optima is clearly a hard problem. The desire to support Quality of Service exacerbates this problem by introducing requirements of timeliness, along with the desire to avoid crosstalk between applications. Further complications arise with the presence of adaptive applications which may respond to their current resource allocation by adjusting their behaviour in order to better fulfil the user's requests.

1.1 Application Adaptation

The key to solving the resource allocation problem is to provide support for both those applications which require *QoS-Assurance*, and those that are able to perform *QoS-Adaptation*. *QoS-Assured* applications are those that require guaranteed levels of a variety of resources, where these levels do not vary over time. The alternative is to develop *QoS-Adaptive* applications that are structured with multiple modes of operation, to be switched between dependent on the current resource allocation. There exist two distinct classes of application adaptation: user-triggered adaptation and cross-resource adaptation. In this paper we consider the class of applications that respond to resource availability in order to meet a fixed user utility.

Currently the majority of adaptive applications only adapt in reaction to variation in a single resource (e.g. network bandwidth). These applications typically attempt to maximise user utility functions by reducing quality in one dimension (e.g. picture quality) to increase a perceptually more important metric (e.g. frame-rate). However, in an end-system where all resources are limited, it becomes advantageous to develop applications that can adapt with respect to multiple resources; this can give rise to complex tradeoffs.

Consider for example a 'Movie Player' application, playing video from a CD-ROM and displaying it on the screen. A traditionally adaptive application–see, for example [3]–might adapt solely with respect to the CPU bandwidth available. However, if the format of the video on the disc is carefully chosen, multiple resource adaptation becomes possible, between disc bandwidth, CPU bandwidth and the buffer memory required to decode the video. This allows for a spectrum of operation modes, each of possibly similar utility to the user.

In this case, resource management is then the task of selecting from these multiple operation modes, with the aim of achieving the user's desired utility, *and* maximising the amount of useful work that other applications may carry out whilst this is occurring.

More generally, the primary goals of a multi-service resource management system are maintaining system efficiency and providing support for differing types of application. Broadly speaking applications can be divided into several classes:

- Batch. For example, compiler jobs.
- Interactive. For example, text editors and shells.
- Assured real-time. These applications require guaranteed access to their resources for their complete lifetime. Examples include CD-ROM writing.
- Adaptive real-time. These applications can adapt to their resource allocation. Examples include video and audio players utilising suitably scalable data coding schemes.

Each of these application classes requires quality of service support of one form or another. Batch applications require guarantees of progress, while interactive applications require bounds on latency. In Section 6 we consider how our architecture may be employed to attempt to meet the demands of these application classes.

* {Neil.Stratford,Richard.Mortier}@cl.cam.ac.uk. University of Cambridge Computer Laboratory, Cambridge CB2 3QG, U.K.

1.2 Conventional Approach

Traditional systems, such as variants of Unix, attempt to solve only the global optimisation problem through policies embedded in the kernel. Recently in the literature the resource allocation problem has been approached by the introduction of a central *Quality of Service Manager* [10, 8, 4, 12]. This entity is made responsible for both the global and local optimisation of resource allocation, requiring accurate, detailed models of application behaviour. Various approximations to the solution are then obtained according to user policies. In general, this approach has several major drawbacks:

- Application model description is complex and it is hard to find a model that is suitable for current applications. Experience suggests that it is *not* possible to predict a model suitable for future applications. In addition, it is desirable to present as simple an interface to the user as possible, ideally through reduction of requirements to a single, simple parameter.
- Centralised QoS management attempts to solve a hard optimisation problem given constraints not only unknown to the system, but also inherently incomplete.
- Centralised management does not scale well across multiple cooperating hosts.
- Co-scheduling artifacts make resource demand hard to predict; an application's demand depends not only on the application itself, but also on inter-resource scheduling artifacts.
- Resource demand depends on inputs that are under external control.

In this paper we present a new resource management architecture that enables the resource allocation and management task to be distributed among applications *and the resources themselves* in a secure and scalable manner. This work is very much in progress and presents many research challenges. We present our current thoughts in the area.

2 Architecture Overview

We believe that there should be a clear distinction between local and global optimisation. Rather than approaching the resource allocation problem by considering the system as a whole, we propose to break the system up into active *applications* and *resource managers*; these are then responsible for their own local optimisation. By correctly setting up these local optimisation problems, it should be possible to achieve an approximation to global optimisation as a side-effect of the local optimisations.

The basic mechanism that we propose to use is that of *resource pricing* [5, 7, 6]. Each resource manager is responsible for maximising its revenue, which is generated by selling *resource contracts* to applications. Applications are responsible for maximising their utility (and thus the user-perceived utility) by purchasing and trading[1] resource contracts. Applications are provided with *credits* from a *User Agent*, renewable over a given time-scale. User agents are responsible for implementing the policy of a particular user of the system. In a multi-user system there may exist system-wide policy which imposes limits on the credits allocated to particular user agents. Figure 1 depicts an overview of the architecture.

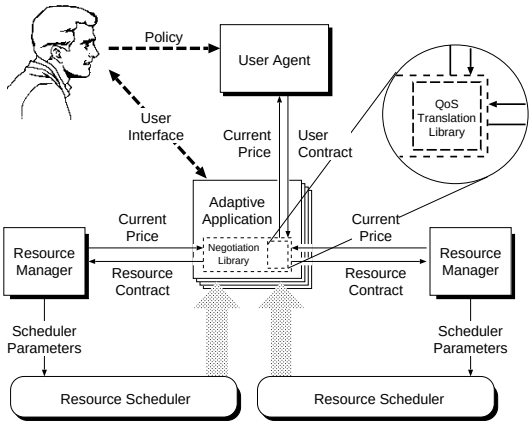

Figure 1: Architecture Overview

The separation between local and global optimisation has several clear advantages over the traditional centralised approach. Each application in this model is responsible for its own utility function, hence there is no requirement to specify a limited form of this function to any external agent (e.g. a central QoS-Manager). This enables applications to enter into application specific forms of adaptation that may be developed on a per-application basis, by the application writer. It is our belief that only application writers are in a position to develop the definition and understanding of user-perceived utility, and how this maps into platform independent requirements. This is definitely not a task that can be carried out by an external agent such as a QoS-Manager. Translation into platform specific requirements is carried out by a QoS-Translation library provided by the system.

Section 3 provides a detailed description of contracts and Section 4 discusses the function of user agents in the architecture. The time-scales over which applications' credits are renewed, and at which they buy and sell contracts are important system control factors and are discussed in the relevant sections. Sections 6.3 and 6.4 give examples of how the system may be used to support the dual goals of QoS-Assurance and QoS-Adaptation.

3 Contracts

The basic unit of negotiation in this system is the *contract*. Contracts come in two flavours:

User Contract These are between the user agent (on behalf of the user), and an application. They are specified in terms of a credit allocation (C_{user}) and a time-period over which that allocation will be renewed (T_{user}).

Resource Contract These are between an application and a resource manager. They are specified in some resource specific manner.

[1] A form of resource trading is used to support adaptation.

3.1 Resource Contract Specification

A *resource contract* specifies a set of parameters in a platform and resource specific manner (following translation from the user contract), and a time span over which that contract will remain valid. An example [9] is use of a 3-tuple contract, (p, s, t), for the CPU resource, where p is a scheduling period measured in milliseconds during which the application is guaranteed s milliseconds (the *slice*) of CPU time, repeated over the length of the contract, t seconds. The slice of time that the application is guaranteed may be allocated at any point within the period, and this time of allocation may change between periods; only the amount of allocation is guaranteed.

The resource manager is responsible for setting the price of the contract to reflect the load that it will inflict on the resource for the full duration of the contract. Longer term contracts (effectively reservations) would be expected to cost more to purchase, due to the problem of unforeseen future demand. This will encourage applications to renegotiate.

The resource manager is then responsible for setting the underlying resource scheduler parameters, based on contracts that it has entered into. We do not impose a parameter set for the resource schedulers since this is highly dependent on the scheduling algorithm chosen, and the the resource in question. We therefore feel that it can be abstracted into a QoS-Translation library.

Contract renewal will occur at various time-scales. We expect user contracts to be renewed at time-scales the order of seconds and resource contracts to be re-negotiated approximately every 100 milliseconds. Scheduling is likely to take place at time-scales of single milliseconds. It is important that we impose a minimum length of resource contracts to avoid wasting resources through excessive re-negotiation.

3.2 Contract Translation

User contracts are translated by the application, from platform independent specification of requirements into the specific demands on the local system by the use of a *QoS-Translation library*. There are many possibilities for development of such a library, with varying degrees of autonomy from application/user involvement. In general a translation library will employ some form of measurement and feedback mechanism. One such possible translation scheme borrows from the field of call admission control in ATM networks and is based on observations of past application behaviour [1].

3.3 Contract Pricing

Resource managers are required to price resource contracts when they are presented with requests from applications. Applications may present requests periodically, either for the renewal of contracts, or as probes for the current price, pending adaptation.

Any suitable resource pricing algorithm therefore must satisfy the following properties:

- Efficiency. It will be activated relatively often and thus should not consume excessive resource.
- Stability. It is highly desirable that the system remain stable with respect to pricing fluctuations. Inflation is a situation that probably should be avoided, although it may prove useful as an indication to the user that it is time to expand their system.
- Variable time-scales. It is essential that the algorithm is able to compute prices for contracts whose length depends on the application and resource involved. This may be done by considering past behaviour observations, or by applying some well known model.

In general, the idea behind resource pricing is to provide the user/application with incentives to act in a correct manner. This is achieved through the feedback mechanism of dynamic pricing—as a resource becomes more congested, its price rises, encouraging users to move to less congested resources (and, potentially, providing a revenue stream for increasing the capacity of the congested resource).

3.4 Contract Trading

In order to support adaptation, applications may be allowed to sell contracts back to the relevant Resource Manager, at a given price, for the remaining part of the contract. Applications that wish to enter into fine-grained adaptation over multiple resources will track the current contract prices and trade their contracts to maximise their utility functions. With a suitable pricing structure this should result in applications moving, where possible, from congested resources to less congested resources, since the less congested resources should be cheaper. To those applications that cannot adapt in any meaningful sense, trading is unlikely to be of any use, and the effect will be of a simple admission control system. This aspect is discussed further in Section 6.

We envisage the use of third-party resource traders for various purposes. The provision of shared disk block caches, shared library code and shared data needs to be charged for in some way. One possibility is to assign ownership of these resources to a third-party, and have that third-party charge for use of those shared resources through applying some *ad hoc* pricing algorithm.

4 User Agent

The *user agent* is an entity that acts on behalf of a user with the aim of implementing policy decisions. A user agent supplies credits periodically to each of the applications that it overlooks. To reflect user policy it may change both the amount of credits allocated to the application and the period over which this allocation is renewed. The user agent can effect application behaviour in a variety of ways.

Reducing the allocation of credits has the effect of reducing the importance of the application in the system. This may result in the application adapting its behaviour, possibly decreasing the quality of its output with respect to other applications. The application is responsible for communicating the cost of its current quality level to the user agent, providing information that can be used in policy decision making.

Adjusting the period (T_{user}) over which new credits (C_{user}) are allocated to an application effects the length and type of contract the application is willing to purchase. If this period is long, an application may purchase long-term contracts and therefore not need to adapt as resource conditions vary (it may however choose to trade contracts to increase its utility). Alternatively if this period is short, the application will be forced to adapt more frequently, reflecting the user's preference concerning the stability of output quality.

These parameters provide mechanisms for the user agent to apply policy, while still enabling applications to respond in application specific ways—ultimate control remains with the application writer, within given bounds. Applications are free to renegotiate individual resource contracts at any period T_{app} and at any level of resource C_{app}, subject to the constraint $T_{app} \leq T_{user}$.

In general a user agent will be extremely simplistic. Many users may prefer to make the policy decisions themselves and act as their own agent. The user agent in our architecture is a placeholder for experimentation with the issues involved. As a simple example, using this architecture it is easy to develop a user-agent that takes into account certain user-preferences when allocating credits. Such preferences could include the application with window-focus receiving more resource, and those that are occluded receiving less. We intend to experiment with some of these user interface issues in future work.

5 Negotiation Library

Highly adaptive applications will obviously wish to enter into negotiation with resource managers in some application specific manner. However, many applications will either not require complex adaptation support, or they will be legacy applications that have been written to some other QoS-Negotiation interface. We provide support for such applications by providing a set of negotiation libraries that will enter into negotiation with resources on the application's behalf.

A negotiation library is responsible for tracking the current price of all resources and reacting in accordance with application supplied parameters. In the case where communication with the resource manager is costly it may be sensible to have the resource manager interrupt those applications with which it currently has contracts, to inform them of the new price, but only when it has undergone a sufficiently large (application specified) change.

As an example, consider an application written for a system with an 'oracle'-like QoS-Manager that accepts quality mode definitions from applications (for instance [12]). A suitable negotiation library will provide the interface traditionally presented by the QoS-Manager to the application, and will be informed of the operating modes. The application may then purchase resources to enable it to achieve the best mode available to it at the present time. The user-agent's credit allocation is effectively mapped onto the old-style modal definition by the QoS-Negotiation library. Using similar techniques we can support many types of legacy applications, including those that have no concept of adaptation or operation modes.

6 Quality of Service Provision

6.1 Batch Applications

Batch applications can be allocated a credit rate according to their importance to the user. The period of credit allocation should be set relatively short. This is to prevent the application reserving resources in times of congestion, when real-time or interactive applications might require immediate access, where the batch application can afford to wait. The application will use the credits from its allocation to buy a contract for the allocation period with as high a resource level as possible. In the case where the application is an unimportant background task, it may be allocated no credits, and therefore only be allowed to run when resource is free (i.e. congestion is non-existent).

6.2 Interactive Applications

Interactive applications require low latency access to resources for short periods of time. This can be achieved by purchasing specialised contracts from resource managers. Resource managers may offer a variety of different contracts, with associated statistical guarantees. These guarantees differ from conventional hard guarantees in that they provide a statistical bound on the likely-hood of violating some parameter, such as delay. Such guarantees enable the system to exploit the gains of statistical multiplexing while providing some predictable level of service. The resource manager will price such contracts according to some measure of the load that they will place on the resource. We also envisage the possibility of co-scheduled contracts/guarantees from multiple-resource managers, possibly through the use of a third-party trader. These contracts will similarly be priced appropriately.

6.3 Assured real-time Applications

Applications that require real-time assurance will purchase long-term contracts for the desired resource level. The user agent should use a very long period of allocation for such applications. This will give the application the (almost) hard guarantee it requires to achieve the desired quality of service.

6.4 Adaptive real-time Applications

Adaptive applications can make the most of this system. They will initially buy contracts for the maximum length of the allocation period, and then trade them as resource prices vary in order to increase their utility. This will allow them to make better use of the resources that the system currently has available, whilst still providing the user with the quality of service they require.

7 Related Work

Resource management for Quality of Service has received a lot of attention in recent years. Examples include AQUA [8], the QoS-Broker [11], the Resource Planner in Rialto [10] and Q-RAM [13]. A good overview of work in this area is given in [2]. Our work differs from the majority of these architectures in that we advocate a distributed solution to the problem, rather than

attempting to fully specify application requirements to a central QoS-Manager.

Various market-based approaches to distributed resource management have been proposed in the literature–a good overview is given in [5]. Our work shares motivation with that presented in [15] but differs through the use of a contract based architecture.

8 Conclusions

This paper has presented a novel approach to the problem of resource management in a modern resource controlled operating system. We believe that the distribution of work between the applications and the individual resources leads to a scalable and functional solution. The application of resource pricing provides incentives for adaptation algorithms by giving feedback on the current levels of congestion for each resource. We are currently developing multimedia applications that can react to such information.

We believe the concept of a timed contract to be central in provision of an experimental test-bed on which to carry out further research. The system can be reduced to a simple rate-based credit allocation system by imposing a global time scale for contracts, whilst still allowing experimentation with more speculative pricing algorithms.

Although initial use of this architecture is in the end-system, we believe that similar ideas could be applied to server systems, such as Xenoservers [14] providing accountable execution of untrusted code in a network, where real customers will be paying real money for real resource. The concept of contracts will be essential in such a system to enable the provision of predictable levels of service. Individual dynamic-pricing of resources should provide incentives for clients to make efficient use of the system and should also maximise the revenue of the owners. In essence we have generalised resource pricing as carried out on multi-user mainframe systems.

This work is far from complete and introduces many new research challenges. The key to a successful system will be in obtaining a sensible balance between efficiency and complexity.

References

[1] Paul Barham, Simon Crosby, Tim Granger, Neil Stratford, Meriel Huggard, and Fergal Toomey. Measurement based resource allocation for multimedia applications. In *Proceedings of the Multimedia Computing and Networking Conference, SPIE Volume 3310*, 1998.

[2] Andrew Campbell, Cristina Aurrecoechea, and Linda Hauw. A review of QoS architectures. *ACM Multimedia Systems Journal*, 1996.

[3] David L. Tennenhouse Charles L. Compton. Collaborative load shedding for media-based applications. In *Proceedings of the Workshop on the Role of Real-Time in Multimedia/Interactive Computing Systems*, November 1993.

[4] S. Chatterjee, J. Sydir, and B. Sabata. Modeling applications for adaptive QoS-based resource management. In *Proceedings of High Assurance Systems Engineering Workshop*, August 1997.

[5] Scott H. Clearwater, editor. *Market-Based Control, A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[6] Costas Courcoubetis and Vasilios A. Siris. An evaluation of pricing schemes that are based on effective usage. Technical Report 214, Institute of Computer Science (ICS), Foundation for Research and Technology, Hellas (FORTH), February 1998.

[7] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[8] K. Laksham and Raj Yavatkar. Integrated CPU and network-I/O QoS management in an endsystem. In *Proceedings of the Fifth International Workshop on Quality of Service*, 1997.

[9] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas In Communications*, 14(7):1280–1297, September 1996. Article describes state in May 1995.

[10] R. Draves J. Barrera M. Jones, P. Leach. Modular real-time resource management in the Rialto operating system. In *Proceedings of HotOS-V*, May 1995.

[11] Klara Nahrstedt and Jonathan M. Smith. The QoS Broker. *IEEE Multimedia*, 1995.

[12] D. Oparah. Adaptive resource management in a multimedia operating system. In *Proceedings of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video*, July 1998.

[13] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Proceedings of The 18th IEEE Real-Time Systems Symposium*, 1997.

[14] Dickon Reed, Ian Pratt, Stephen Early, Paul Menage, and Neil Stratford. Xenoservers: Accountable execution of untrusted programs. Submitted to HotOS'99, November 1999.

[15] Carl A. Waldspurger and William E. Weihl. An object-oriented framework for modular resource management. In *Proceedings of the Fifth Workshop on Object-Orientation in Operating Systems (IWOOOS '96)*, October 1996.