# An Economic Approach to Adaptive Resource Management

Neil Stratford and Richard Mortier*
*University of Cambridge Computer Laboratory, Cambridge, UK*
{*Neil.Stratford, Richard.Mortier*}*@cl.cam.ac.uk*

## Abstract

*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'). The desire for QoS support leads to the dual management goals of global (system) and local (application) optimisation. In this paper we propose an architecture based on an economic model of resource management using frequently renegotiated timed resource contracts. We use dynamic pricing as a congestion feedback mechanism to enable applications to make system policy controlled adaptation decisions. We argue that this scheme has many advantages over a traditional central resource management entity including scalability and application specific adaptation.*

## 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 a linear combination of the utilities of each of their applications. In the case of a multi-user system, we consider the global optimum similarly to be a linear combination 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 users' requirements, and remaining within the constraint of the system's finite resources.

Managing the system's resources in order to achieve global and local optima is clearly a hard problem. The desire to support QoS exacerbates this problem by introducing requirements of timeliness, along with the desire to avoid cross-talk 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. Other approaches have either concentrated solely on optimizing global performance, without consideration of local performance, or on optimizing local performance, thus achieving poor global performance.

Our approach consists of two phases:

- Low-level, firm or statistical guarantees of access to resources by applications;
- Higher-level optimization across multiple resources, and various time-scales, using feedback provided by dynamic pricing.

### 1.1. Application Adaptation

The key to solving the resource management problem is to provide support for differing types of application, including those which are able to enter into some form of adaptation, based on system and user policy. We refer to such applications as *QoS-Adaptive*. Two distinct classes of application adaptation exist: user-triggered and resource-triggered. In this paper we consider the latter—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 limited resource (e.g. network bandwidth). These applications typically attempt to maximise user utility by reducing quality in one dimension (e.g. picture resolution) to increase a perceptually more important metric (e.g. frame-rate). However, due to the many limited resources in an end-system it becomes advantageous to develop applications that can adapt with respect to multiple resources. This should allow greater overall utilization, but 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.

*Supported by an EPSRC CASE award, in collaboration with BT.

A traditionally adaptive application—see for example [4]—might adapt solely in response to the CPU bandwidth available. However, if the format of the video on the disc is carefully chosen, adaptation across multiple resources becomes possible; for example, between disc bandwidth, CPU bandwidth and buffer memory. This allows for a spectrum of operation modes, with each giving potentially 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 both the user's desired utility, *and* maximising the amount of useful work that other applications may carry out whilst this is occurring.

### 1.2. Conventional Approach

Traditional systems, such as variants of Unix, attempt to solve the global optimisation problem solely through policies embedded in the kernel. Recently the resource allocation problem for *QoS* has been approached by the introduction of a central *QoS Manager* [6, 8, 2]. This entity is made responsible for both 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 all current applications. Experience suggests that it is *not* possible to predict a model suitable for future applications. It is also 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 for a resource depends not only on the application and the other resources it requires, but also on the demands of other applications for any of the system's resources.

- Resource demand depends on inputs that are under control of the user, and therefore external to the system and very hard to model.

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 controlled and scalable manner. This work is very much in progress and involves 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 *applications* that are involved in negotiating their own resource allocations, 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 close to global optimisation as a side-effect of the local optimisations.

The basic mechanism that we propose to use is that of *resource pricing* [3, 7, 5]. 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 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.

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, so there is no requirement to specify a limited form of this function to any external agent (e.g. a central QoS-Manager). This enables application specific forms of adaptation that may be developed by the application's writers. It is our belief that only an application's writers are in a position to understand the specific resource requirements and potential tradeoffs that an application may make. Application resource access requirements should be translated by the application itself into system specific parameters, through the use of a QoS-Translation library. This library may be provided by the system, presenting a standard interface, or supplied by the application itself, allowing for more complex application specific parameters (such as feedback queue lengths) to be taken into consideration.

Section 3 provides a detailed description of contracts, their specification, pricing, trading, and negotiation. Section 4 discusses the function of user agents in the architecture and Section 5 gives a description of how the system may be used to support differing types of application. Finally, Sections 6 and 7 present related work and some con-

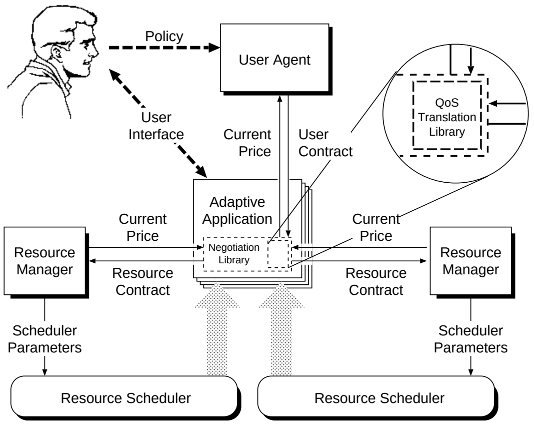


**Figure 1. Architecture Overview.**

clusions.

## 3. Contracts

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

**User Agent-Application Contract** These are between the user agent, implementing the user's policy, and an application. They are specified in terms of an application specific QoS specification, a credit allocation ($C_{user}$) and a time-period over which that allocation will be renewed ($T_{user}$).

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

### 3.1. Application-Resource Contract Specification

An *application-resource contract* is specified in a platform and resource specific manner and is obtained from the application's higher level QoS requirements by translation through a *QoS-Translation Library*. In addition the application-resource contract is specified with a given time span over which it remains valid. An example of this style of platform dependent application-resource contract is a 3-tuple for the CPU resource, $(p, s, t)$ [9]. Here $p$ is a scheduling *period* measured in milliseconds during which the application is guaranteed a *slice*, $s$ milliseconds, of CPU time, repeated over the length of the contract, $t$ seconds. The slice 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 proportion allocated is guaranteed. It is platform specific in that the required slice and period parameters for a given application QoS will vary over different platforms. Note that this is only one instance of an application-resource contract for the CPU; we envisage others.

There are many possibilities for development of QoS-Translation libraries having varying degrees of independence from application and user involvement. In general a translation library will need to employ a 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]. Another scheme based on control theory, using application level parameters such as queue lengths, is presented in [15]. It is the application writers' decision to select or develop a suitable QoS-Translation library for their specific purpose. We expect however that many applications will utilise the interfaces presented by the system translation libraries provided to hide the scheduling parameters of specific resource managers.

The resource manager is responsible for setting the price of the contract to reflect the load that it will impose on the resource for the full duration of the contract. Longer term contracts would be expected to cost more to purchase, due to the problem of unforeseen future demand. This will encourage applications to renegotiate, thus increasing global utility; see Section 3.3 for further discussion of contract trading. The resource manager is also responsible for setting the underlying resource scheduler parameters, based on the contracts into which it has entered. 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.

Contract renewal will occur at various time-scales. We expect user agent-application contracts to be renewed at time-scales of the order of seconds and application-resource contracts to be re-negotiated of the order of tens or hundreds of milliseconds. Scheduling is likely to take place at time-scales of single milliseconds.

### 3.2. 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 must therefore satisfy the following properties:

- Efficiency. It will be activated relatively often and thus should not consume significant 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 owner that it is time to expand the system.

- Independence of time-scale. It is desirable that the contracts in such a system are not fixed to a particular duration. Applications should be able to specify contracts of varying lengths, dependent on their requirements, and resource managers should be able to specify maximum contract lengths, which may change as the system evolves.

In general, the idea behind resource pricing is to provide the user and application with incentives to act in a manner compatible with other users and applications getting work done whilst maintaining high resource utilization. This is achieved by the feedback mechanism of 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, identifiable by its higher price.

### 3.3. Contract Trading

In order to support adaptation, applications may be allowed to sell resource associated with the remaining part of contracts back to the relevant Resource Manager, at a "fair" price. 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 demand shifting 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 5.

We envisage the use of third-party resource traders for various purposes. The provision of shared library code, shared data, and shared caches needs to be charged for in some way. One possibility is to assign ownership of these shared resources to a third-party, and have that third-party charge for use by application of some pricing algorithm. It should also be noted that, as contract trading is likely to use resources itself, the pricing structure used should make this explicit. This should help to damp the system, preventing resource being wasted by continually trading contracts, without applications ever being able to use the resource associated with them for real work.

### 3.4. Contract Negotiation Library

Highly adaptive applications will wish to enter into negotiation with resource managers in some application specific manner using specially developed algorithms. These algorithms will be able to use application specific knowledge, such as progress indicators and traffic parameter estimation, in making negotiation decisions. However, many applications will either not require complex adaptation support, or 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 for 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. This is most sensibly implemented as a call-back notification mechanism that is invoked when the price leaves a certain, specified range. Note that it is only the price that will fluctuate through demand, and not the level of received resource access throughout the length of a contract.

As an example, consider an application written for a system with a centralised 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 application's 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 different modes of operation.

## 4. User Agent

The *user agent* is an application that acts on behalf of a user with the aim of implementing policy decisions. It periodically supplies credits to each of the applications that it oversees, including itself—it is an application, and must therefore also purchase application-resource contracts. To reflect user policy it may change both the allocation of credits to a given application and the period over which this allocation is renewed. The user agent can affect 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 able 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 of an application's behaviour in the face of scarce resource remains with the application's writers. Applications are free to renegotiate individual resource contracts at any period $T_{app}$ and for any level of resource, subject to the constraint $T_{app} \leq T_{user}$.

In general a user agent will be extremely simplistic. Many users prefer to make policy decisions themselves and would act as their own agent. The user agent in our architecture is a placeholder to allow experimentation with the issues involved. As a simple example, using this architecture it is easy to develop a user agent that takes into account 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, in addition to the more basic user agent and resource manager issues.

## 5. Application Support

Traditional batch style applications (such as compilers) may be allocated credits at a rate according to their importance to the user. 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.

Interactive applications that require low latency access to resources for short periods of time may be supported by purchasing contracts with associated statistical guarantees. These provide only a statistical bound on the likelihood of violating a contract parameter. 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 interdependent contracts between multiple resource managers, possibly through the use of a third-party trader. We intend to use such contracts to experiment with resource co-scheduling.

Soft real-time applications that require some form of assurance are expected to 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 firm guarantees it requires to achieve the desired QoS.

Finally, modern highly adaptive applications will be able to make full use 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, while still providing the user with the QoS they require.

## 6. Related Work

Resource management is a fundamental task of an operating system and has received much attention in the literature. Adaptive resource management has been addressed by both the QoS and the Mobile Application community in detail, with different goals in mind. Much of the recent work in the QoS community has advocated the construction of a centralised resource management entity. This work includes AQUA [8], the QoS-Broker [10], the Resource Planner in Rialto [6] and Q-RAM [13]. In general each application specifies a set of valid operating modes and a utility value for each mode. It is then the task of the resource manager to perform global and local optimisation of the system. We argue that it is possible to construct a distributed resource management system based on simple economic models to provide congestion feedback and policy control.

Applying economic models to resource management is not a new idea in itself, having been sporadically attempted over the past thirty years. An excellent overview of recent work in the area is presented in [3]. Of particular relevance is [17] which shares motivation with our work, but differs through our development of a contract based architecture. We believe that frequent firm light-weight contract renegotiation is the correct way to provide for application level adaptation stability. Similar ideas have also been applied to the Mariposa database system at Berkeley [16] for query resource allocation.

Work in the field of mobility has also addressed the adaptive resource management problem. An example is Odyssey [11] which presents a scheme that is complementary to our work. Odyssey advocates the use of *Application-Aware Adaptation*, which is similar to our separation of local and global concerns, and makes the applications themselves responsible for adaptation decisions, based on resource availability feedback from the system. However, Odyssey chooses to abandon resource reservations and guarantees completely, whereas we believe them to be important for stability.

## 7. Conclusions

This paper has presented a novel approach to the problem of resource management in modern resource controlled operating systems. 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 at each resource to the applications causing the congestion. We are currently developing multimedia applications that can react to such information and adapt their resource usage behaviour away from congested resources to lightly loaded resources. We believe that there is interesting research to be done to enable support for cross-resource adaptation, in both the system support and application development arenas.

Although initial use of this architecture is in the end-system, we believe that similar ideas could be applied to server systems where real customers will be paying real money for real resource; for example, in a Xenoservers system [14]. The concept of contracts is essential in such a system to enable the provision of predictable levels of service. Dynamic pricing of individual resources should provide incentives for clients to make efficient use of the system. In addition it should also maximise the revenue of the owners. In essence we have generalised resource pricing as carried out on multi-user mainframe systems, and are using it on a shorter time-scale to influence the per-resource scheduling decisions.

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] P. Barham, S. Crosby, T. Granger, N. Stratford, M. Huggard, and F. Toomey. Measurement based resource allocation for multimedia applications. In *Proceedings of the Multimedia Computing and Networking Conference, SPIE Volume 3310*, 1998.

[2] 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.

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

[4] C. L. Compton and D. L. Tennenhouse. 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.

[5] C. Courcoubetis and V. 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.

[6] M. Jones, P. Leach, R. Draves, and J. Barrera. Modular real-time resource management in the Rialto operating system. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, May 1995.

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

[8] K. Laksham and R. Yavatkar. Integrated CPU and network-I/O QoS management in an endsystem. In *Proceedings of the Fifth International Workshop on Quality of Service (IWQOS'97)*, 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] K. Nahrstedt and J. M. Smith. The QoS Broker. *IEEE Multimedia*, 1995.

[11] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. R. Walker. Agile application aware adaptation for mobility. In *Proccesings of the 16th ACM Symposium on Operating System Principles*, October 1997.

[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] D. Reed, I. Pratt, S. Early, P. Menage, and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, March 1999.

[15] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI'99)*, February 1999.

[16] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. Technical Report 49, University of California, Berkley, April 1994.

[17] C. A. Waldspurger and W. 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.