Congestion Prices as Feedback Signals: An Approach to QoS Management

Rolf Neugebauer *
Department of Computing Science
University of Glasgow
Glasgow, G12 8QQ, Scotland, U.K.
neugebar@dcs.gla.ac.uk

Derek McAuley
Microsoft Research Limited
St. George House
Cambridge CB2 3NH, U.K.
dmcauley@microsoft.com

Abstract

Recently there has been a renewed interest in the application of economic models to the management of computational resources. Most of this interest is focused on pricing models for the Internet; in particular, on congestion or shadow prices, that address the phenomenon of what economists call external costs — users are exposed to the costs they impose on other users when causing congestion of a resource.

This paper describes how congestion prices can be applied to resource management in operating systems. Shadow prices are interpreted as feedback signals to applications which can adjust their resource requirements according to an application-specific strategy. This leads to a decentralised approach of resource management where applications are enabled and encouraged to perform resource and quality tradeoffs themselves. We have implemented a simulation environment and a number of strategies to evaluate the usefulness of congestion prices as a feedback signal and demonstrate that this approach can offer different service levels to different tasks. We also discuss how the simulation results can be applied in a real operating system and how this work can be extended to form a generic resource management framework.

1 Motivation

Over the past decade or so there have been numerous proposals for adding support for multi-media and Quality of Service (QoS) guarantees to operating systems. In order to provide predictable service to applications, resource reservations are used, giving hard or soft guarantees for resource allocations. Typically, reservations for temporal resources, in particular the CPU, are either based on real-time scheduling algorithms [18, 13, 20, 8] or on proportional share algorithms [4, 23, 2]. The latter are often used in conjunction with a hierarchical scheduling framework which partitions the available resources between tasks, with reservations and tasks sharing resources in a weighted fair fashion.

Resource reservation schemes in general face two related problems: resource requirements have to be specified in advance; and an admission control system has to be deployed to ensure that granted resource reservations can be met by the system. Simply providing a mechanism for specifying resource requirements is not sufficient – the challenge lies in determining the actual resource needs within the constraints of the system's finite resources *and* user preferences. The primary aim of an admission control system is to take these potentially inaccurate and time-varying resource requirements and provide at least soft guarantees to applications that their resource requirements will be met. However, a simplistic approach to admission control based on a first-come-first-served strategy may lead to an inflexible and unfair system, where tasks arriving later than others may be denied access to a resource even if they are more important [20]. Ultimately, the admission control or resource management system should aim to provide maximum utility to the users of the system.

*Rolf Neugebauer is supported by a Marie Curie Fellowship from the European Union, Contract-No.: ERBFMBICT972363.

To tackle these problems, the use of QoS managers [9, 12, 21] is usually proposed. Applications inform a QoS manager of their resource requirements and users may specify their respective utility values. The manager then attempts to maximise the overall resource utilisation and system utility. In [24] it is argued that this potentially NP-hard problem can be tackled by distributing the resource management task among the applications and the resources themselves. The proposed architecture is based on an economic model whereby resource managers responsible for individual resources sell resource contracts to applications. Applications, provided with credits by the users or a user agent, negotiate and purchase these contracts and attempt to locally maximise their, and therefore the user's, utility.

In this paper we build on this general model and examine in more detail a pricing scheme for CPU resources. In particular we investigate the application of a congestion pricing model developed in the context of congestion avoidance in communication networks. In the next section we briefly describe this congestion pricing model. In section 3 we discuss how this model can be used for CPU resource management and section 4 presents a number of sample strategies showing how applications can react to the feedback provided by shadow prices. In section 5 we describe the issues which need to be addressed when applying congestion pricing in the context of a real operating system. Section 6 presents related work and in section 7 we present our conclusions.

2 Congestion Pricing

The basic task of a resource management system aiming to provide resource reservation and timeliness guarantees is to avoid overload of the resource while still offering fairness and flexibility to users and their tasks. This task can be expressed in economic terms and has been done for congestion control for the Internet by economists [16], mathematicians [10], and computer scientists [11].

These proposals are motivated by the observation that if the network is not saturated, the marginal cost of sending an extra packet is close to zero. If, however, the network is congested, the marginal cost of sending an extra packet may incur an unreasonably high cost in the form of increased congestion and packet loss to other users. In economic terms congestion is an "externality", a phenomenon closely related to the "tragedy of the commons" [5] and typically *shadow prices* are used to cover these external costs. In other words, shadow prices make users aware of the external cost they impose on others.

This basic concept leads to a simple decentralised model for resource management, with resource managers providing feedback in the form of shadow prices and applications reacting accordingly. This approach has a number of appealing properties. Firstly, shadow prices form both a very simple and generic feedback signal. Secondly, applications can take full advantage of application-specific knowledge when adapting to varying resource availability, taking into account application-specific user preferences. Thirdly, an appropriate pricing scheme with sensible consumer behaviour can achieve a (near) optimal resource allocation with all consumers maximising their benefits from limited resources (*social optimum*). And finally, congestion pricing is an elegant and simple mechanism for resource revocation, a problem typically not addressed in reservation based schemes.

To formalise the notion of congestion prices we follow the model presented in [10] and [11]. In general, a user i of a resource can be represented as $U_i(x_i) = u_i(x_i) - C$ where u_i is the user's utility function depending on the amount of a resource x_i he receives and C denotes some non constant cost associated with the resource consumption. Naturally, users seek to maximise U_i.

A social planner, on the other hand, would attempt to maximise the sum of all user's utility minus the cost of the overall system load (externalities) to achieve an efficient resource utilisation.

$$\max \sum_{i=1}^{n} u_i(x_i) - C(\sum_{i=1}^{n} x_i) \tag{1}$$

While this optimisation problem is mathematically tractable, its solution relies on knowledge of the users' utility functions $u_i(x_i)$, which are typically not known to the system. However, in [10], it is demonstrated that the problem can be decomposed into an optimisation problem for each user and an optimisation problem, not involving the user's utility functions, for the resource.

Suppose a user is charged a rate t_i proportional to the amount of the resource x_i he receives. Then the user faces the optimisation problem:

$$\max U_i(x_i) = u_i(x_i) - t_i x_i \tag{2}$$

For a monotonically increasing, concave, and continously differentiable utility function, the unique solution is:

$$u_i'(x_i) = t_i$$

If the resource manager seeks to achieve a socially optimal resource allocation according to equation 1 it will set the charge t_i to the shadow price $p(y)$ depending on the load y of the resource giving:

$$t_i = p(y) = \frac{d}{dy}C(y) \tag{3}$$

with $C(y)$ being the rate at which cost is incurred at overall load y. Thus, the feedback signal in form of the charge $x_i p(y)$ is both proportional to the user's resource allocation and the congestion cost it incurs.

Equation 2 can be presented in an alternative form. Suppose the user sets an amount w_i and in return receives a share of the resource x_i proportional to w_i such that $w_i = t_i x_i$. The user optimisation problem becomes:

$$\max U_i(x_i) = u_i(\frac{w_i}{t_i}) - w_i \tag{4}$$

Analogous to the solution to equation 2, the condition: $u_i'(\frac{w_i}{t_i}) = t_i$ identifies the optimal choice of w_i under a given price t_i.

3 Application to CPU scheduling

The pricing scheme described in section 2 has been proposed as a congestion avoidance mechanism for IP networks, where a clear indication of congestion exists in the form of dropped packets. In a soft real-time system, without strict admission control, the analogous congestion indicator are missed deadlines.

To illustrate this, consider a simple Earliest Deadline First (EDF) based scheduler in which tasks specify their resource demands as a tuple $[p, s]$ to receive a share of s nanoseconds every period of p nanoseconds. It has been shown [14] that, under certain assumptions, EDF generates a feasible schedule if the overall reservation is below 100%. If the overall system utilisation exceeds 100% and all tasks consume their entire share each period then, inevitably, deadlines will be missed. Since we are considering a soft real-time system this can be tolerated *occasionally*.

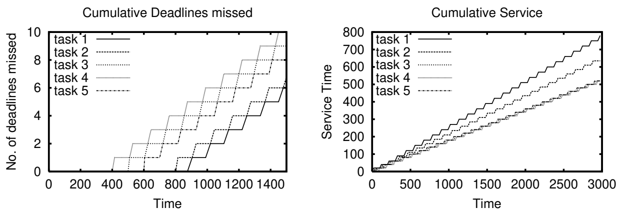


Figure 1: Cumulative missed deadlines and service time in 15% overload

In figure 1 the results of a simulation of a set of five tasks, each initially having a reservation of $[100, 20]$ equivalent to 20% of the CPU, are given. During the initial phase no deadlines are missed and the tasks receive the same service time each period. At time $t = 300$ the reservation of two tasks is increased to $[100, 30]$ and $[100, 25]$ respectively, resulting in a 15% overload of the system. As one would expect with EDF, not all deadlines can be met. Due to the relatively high overload, all tasks start missing deadlines after a short period. Thus, the two tasks that increased their share have an external effect on the other tasks. Since our implementation of EDF[1] does not discard requests whose deadlines cannot be met, all tasks continue to receive service time, with the three unchanged tasks making slightly less progress and the tasks with the increased resource allocation making more progress. Essentially, when overloaded the scheduling algorithm degrades to weighted fair sharing. However, the longer the CPU is overloaded the more deadlines are missed; quickly converging to the point where all deadlines are missed. Clearly, this situation should be avoided for longer periods of time and situations of overload should be transient. From this simple experiment we conclude that the metric of missed deadlines is a suitable indicator for resource congestion.

If missed deadlines represent the external cost of overload, the next question is how to translate this into shadow prices. Consider a task A whose deadline d_n was missed while its previous deadline d_{n-1} was met. Assume also, that the deadlines of all other active tasks are met. Essentially, all tasks which ran between task A's previous deadline d_{n-1} and its missed deadline d_n contributed to the missed deadline in proportion to the share of the resource they consumed during that period. More generally, all users of a resource should be charged proportionally to their use of the resource from the start of a busy period leading to an overload until the overload situation ends. However, this seems impractical to implement directly since it requires the system to be able to predict the future resource usage.

An alternative has been described in [11]. Instead of charging from the start of a busy period, one could start charging tasks

[1] Which is based on the EDF variant deployed in the Nemesis operating system [13], known as Atropos.

when the first deadline has been missed but continue charging after the congestion phase has stopped. This may lead to a minor unfairness, where tasks which did not contribute to the congestion might get charged. However, we anticipate that charging periods are only of short duration while changes to the task set occur on larger timescales.

This strategy could be implemented using the periodic timer interrupt most operating systems use to maintain the system software clock and/or to perform scheduling related functions such as updating usage statistics or priority re-computation. If the scheduler detected a missed deadline (i.e., an overload) the interrupt service routine would decrement the account of the currently running task by a fixed value on every execution until the scheduler detects that the system is idle. On standard PC style hardware this periodic timer can be set to periods as small as $122\mu s$ offering a high resolution feedback signal.[2] A variation of this scheme could implement a strategy akin to Random Early Detection (RED) [3], where tasks would be charged small amounts at random before actual overload occurs and charged as described above in overload. This may prevent situations of heavy overload by reducing the overall system utilisation marginally. We plan to investigate different charging schemes in more detail in the future.

4 Sample Strategies

The charging mechanism introduced in the previous section provides a feedback signal to individual tasks in the form of congestion charges. A user can express the relative importance of tasks by assigning different "budgets" to different tasks. Tasks can then use this feedback to implement a variety of different strategies to adapt to these feedback signals. We have implemented a simulation environment and a number of strategies to evaluate the usefulness of congestion prices as feedback signals. In this section we present some initial results of this evaluation. A detailed description of the simulator and the implementation of the strategies is beyond the scope of this paper. We hope to report on it in detail in the future.

The sample strategies include a simplified TCP-like algorithm implementing an additive increase/exponential decrease strategy: If a task incurs no congestion charges it requests a small, fixed-sized increase of its resource allocation; if the rate at which a task incurs congestion charges is above a set threshold it reduces its resource reservation by a fixed proportion.

A second strategy implements the Willingness To Pay (WTP) algorithm described in [11]:

$$\Delta x_i(t) = \kappa_i(w_i(t) - x_i(t) \times p(y(t))) \quad (5)$$

with x_i denoting the current service rate, w_i the willingness to pay factor, i.e., the rate at which the task is willing to pay for congestion charges, p the rate at which congestion charges are incurred, and κ_i a constant influencing the rate of convergence.

A third strategy implements a Proportional-Integral Derivative (PID) controller similar to the one used in [22, 15]:

$$\Delta x_i(t) = C_{iP} e_i(t) + C_{iI} \int e_i(t) + C_{iD} \frac{de_i(t)}{dt} \quad (6)$$

with C_{iP}, C_{iI}, C_{iD} being constants and $e_i(t)$ representing the error between a configurable willingness to pay rate and the rate at which congestion charges are incurred, i.e. $e_i(t) = w_i(t) - p(y(t))$.

In a first experiment we illustrate the basic behaviour of each of the strategies. For each experiment five tasks were started with a period of $10ms$ and an initial slice of $1ms$. The algorithms for the different strategies are performed once every $100ms$ by each task and alter the slice of the task to change its execution rate x_i accordingly. If the system is overloaded, the tasks get charged 1 unit every $100\mu s$ (modelling a $122\mu s$ periodic timer interrupt) thus the maximum charging rate is 10000 $units/s$. For the experiments with the WTP and PID strategy, the willingness to pay factor is set to 100 $units/s$. The results are shown in figure 2 as the service rate each task received, averaged over $200ms$. Table 1 summarises the three experiments by giving the mean ($\bar{x}$) and the variance of the achieved service rate (σ^2), together with the total number of missed deadlines for each task (d).

T	TCP-like			WTP			PID		
	$\bar{x}$	σ^2	d	$\bar{x}$	σ^2	d	$\bar{x}$	σ^2	d
1	16.13	7.99	0	18.73	2.55	0	19.04	2.40	0
2	18.88	9.97	386	19.01	2.79	0	19.05	2.40	0
3	16.86	12.29	0	19.05	2.82	0	19.46	2.73	511
4	17.16	8.47	2	19.02	2.80	0	18.83	2.23	0
5	15.52	10.42	0	19.39	3.18	487	19.03	2.37	0

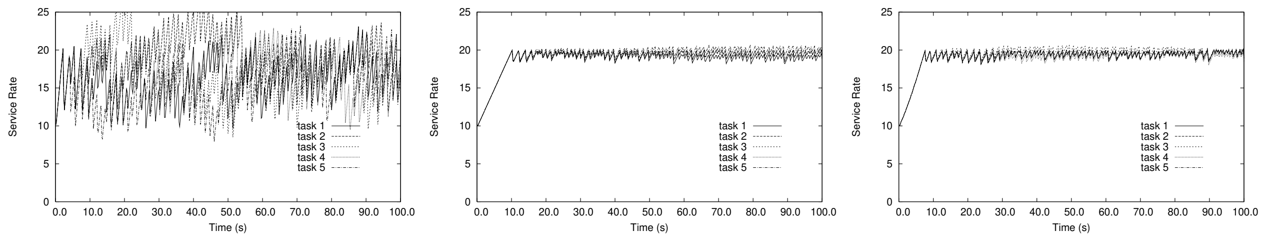
Table 1: Summary of the three experiments

For the TCP-like algorithm the average service rate is comparably low with a large variance due to its aggressive back-off. Tasks sometimes back-off for consecutive periods because the system may need a longer time to catch up after missing deadlines. Furthermore, the system does not converge to a stable allocation. In contrast, using WTP achieves close to optimal average allocation for each task after about 10 seconds. However, the tasks' service rates oscillate slightly just below the optimal allocation. The PID algorithm converges slightly quicker towards a stable state where the tasks allocation oscillates similar to the tasks using the WTP strategy. It is worth noting, however, that it is fairly difficult to tune the parameters of the PID algorithm. In fact the configuration used for all experiments in this paper ($C_P = .1, C_I = .001, C_D = .1$) were found through a trial-and-error process, with variations of the parameters often leading to unstable behaviour. The convergence constant κ_i used by the WTP algorithm has a far less dramatic effect and simply influences the rate of convergence (we use $\kappa_i = .1$ for all experiments).

The uneven distribution of missed deadlines for each of the strategies and the uneven distribution of service rates for the TCP-like strategy can be attributed to the implementation of the scheduler. Tasks on the run queue are ordered by their deadlines and are inserted into the run queue in the same order after each period. In the experiment, all tasks have the same deadlines and start at exactly the same time, thus, under transient overload, the same tasks are likely to miss deadlines. However, we anticipate that in a more realistic environment, where a larger number of tasks with more diverse deadlines are active and where tasks block or yield occasionally, this effect will be far less dominant.

For both WTP and PID the user can specify a willingness to pay factor $w_i(t)$ for each task which determines the rate at which the task is able to pay for congestion prices. Thus, tasks with different $w_i(t)$ factors should receive different levels of service. Essentially, the proportion of $\sum w_i$ to the maximum charging rate determines which level of overload is acceptable to the system.

[2]Naturally, care must be taken so that this does not impose an intolerable overhead. On the version of Nemesis for Alpha processors the periodic timer interrupt is *only* used for maintaining wall clock time and the service routine is entirely implemented in PAL code. Changing the timer period to $122\mu s$ and adding code for implementing the charging scheme does not impact the overall performance of the system.



TCP-like WTP PID

Figure 2: Sample strategies

For example, if w_i is set to 50 $units/s$ instead of 100 $units/s$ in the previous experiment the numbers of missed deadlines are about half as big as the ones shown in table 1.

Next, we evaluate the ability of the WTP and PID strategies to provide different service levels to different tasks and the ability of the strategies to adapt in a changing environment. For each of the strategies we start 9 tasks with different initial allocations (10%, 5%, 1%, .5%), different periods ($5ms$,$10ms$,$20ms$), and different values for w_i (100, 200, 300, 400). After $50s$ a tenth task joins, after $100s$ the value of w_i for one task is changed from 100 to 400 while after 150s the value of w_i for another task is reduced from 400 to 200, and finally, after 200s a task terminates.

The results of this experiment for both strategies are shown in figure 3 which illustrates the tasks' received service rates. In both experiments one can clearly see the 4 different service levels corresponding to the different values of w_i. For the PID strategy the task set converges quicker to a stable allocation and the service rates adjust quicker when the task set changes or the w_i parameters are changed. In general, both algorithms yield comparable results. However, the implementation of the PID algorithm is more complex than the very simple implementation of the WTP algorithm.

It is important to note that the service levels achieved by the tasks are solely depending on w_i and are *independent* of their periods *and* their initial allocation. Thus, the decentralised system, in which tasks choose their resource allocation themselves based on feedback signals by the scheduler, can lead to a weighted fair resource allocation, while still providing soft real-time guarantees over short periods of time.

These three different strategies can be used for different types of applications. For example, a low priority background task such as a background document indexer or client for a distributed computing challenge such as rc5des[3] or SETI@Home[4] may use a TCP-like strategy to rapidly decrease their resource allocation when the system becomes overloaded. A PID or WTP like strategy might be used by more important batch processing jobs or multi-media applications. Furthermore, with the w_i parameter the user is provided with a single, easily understood "knob" to adjust the importance of a task.

5 Issues and Future Work

We have described a general model for congestion pricing and demonstrated through a number of experiments that the feedback provided through shadow prices can be used by applications to change their resource allocation. In this section we discuss the issues involved in applying this model to the management of resources in a real operating system.

In the above experiments, tasks voluntarily adjust their resource allocation during overload. However, in a real system, tasks may not be that cooperative. To promote cooperation, tasks should be given credits and the amount of credits a task may spend to cover congestion costs should be limited. If a task runs out of credits it may only execute best-effort until the transient overload has ended. Limiting the available credits provides the right incentives to tasks to only use the necessary resources and also limits the maximum tolerable overload. Furthermore, the user can express relative importance of tasks by allocating different amounts of credits to different tasks.

An important issue in this context is that for the system to be stable, the total amount of credits available in the system needs to be limited and a task may not be allowed to accumulate arbitrary amounts of credit nor create credits. This could be achieved by allocating new credits to tasks in intervals at a user defined rate and decaying unused credits over time. Users will only have a limited amount of credit available. This model could be extended to support abstractions such as user-defined currencies and inflation of these currencies similar to the abstractions used in lottery scheduling [25].

The three different strategies, and others, require to be evaluated exhaustively with more realistic workloads in a more dynamic environment. We intend to do this both through simulations and by implementing the model in a real operating system, namely Nemesis [13]. In particular, we are interested in multimedia applications which can adapt their resource requirements by offering different levels of quality to users. Such applications can use the feedback provided by the system through congestion prices and the feedback provided by the user through credit allocation to make informed quality and resource tradeoffs based on more general, application-specific user preferences. We have implemented a small toolkit which can assist developers with this task [19]. Furthermore, we are interested in the interaction between different adaption strategies, in particular the relationship between w_i, charging mechanisms, and the tasks' adaption strategies. In this context we also plan to investigate if the system can be "sabotaged" by deliberately misbehaving tasks.

The overall aim of the pricing scheme should be to prevent the system from being overloaded in the stable state and provide maximum user utility. This allows the system to give short term resource guarantees to applications which have strict timeliness requirements. We anticipate that applications are prepared to adapt to changing resource availability over longer periods. We currently do not have enough experience with the system to make predictions about the timescales at which these changes will occur. However, we currently consider two techniques which can influence the timescale at which resource reservations change. Firstly, an admission control system could be deployed which only allows new tasks to enter the system if the system is in a stable

[3] http://www.distributed.net/

[4] http://setiathome.ssl.berkeley.edu/

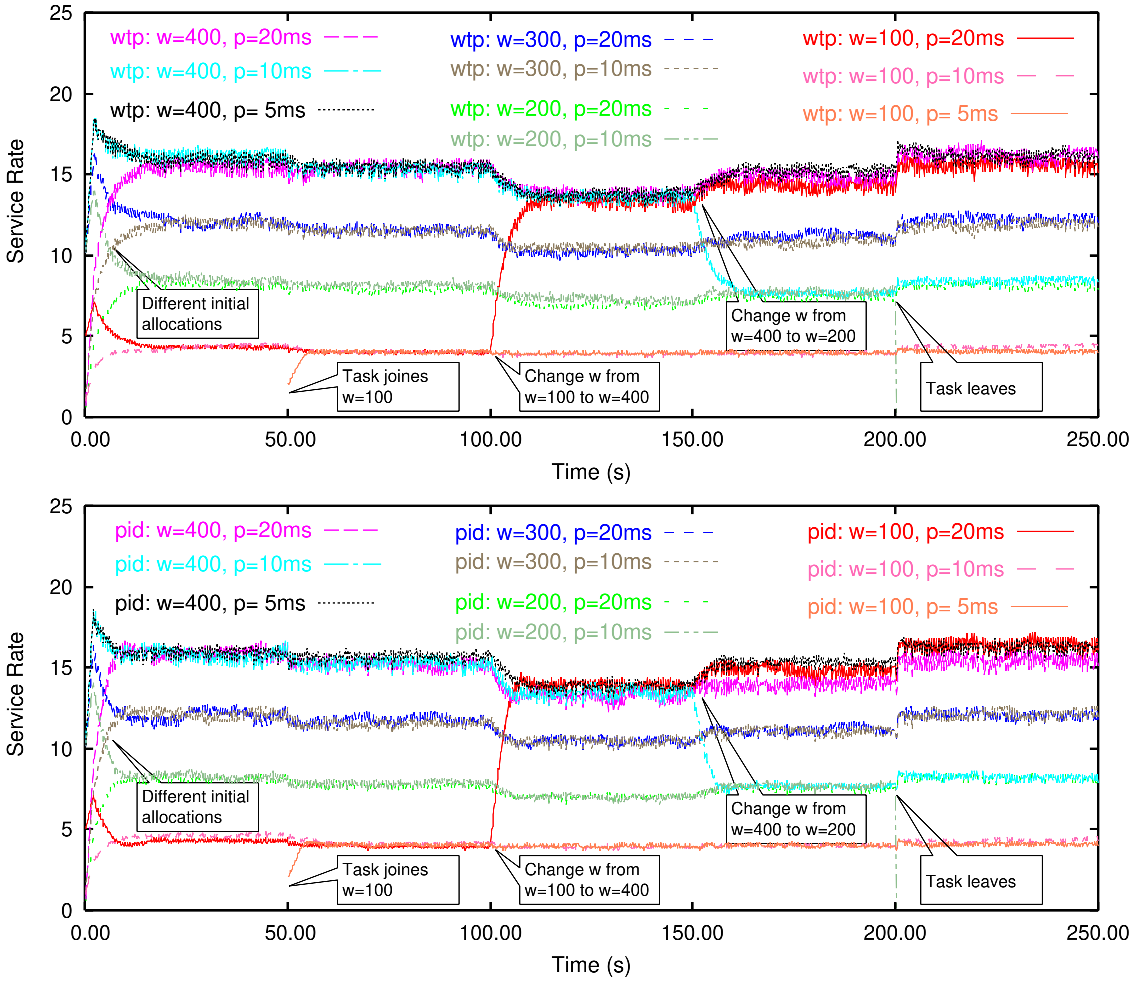


Figure 3: WTP (top) and PID (bottom): Received Service Rate (Averaged over $200ms$)

state or if the new tasks' initial allocation will not push the overall resource reservation above a threshold not much greater than 100%. Secondly, users could deploy a user agent which dynamically alters the credit allocations of the users' tasks based on some user-defined preferences.

So far we have only considered a single resource, CPU resource allocations. However, we believe that the general principal of congestion pricing is applicable to other resources. For example, for virtual memory management, the number of page faults can be used as an indication of congestion of physical memory. In general, managing resource reservations for multiple resources is very difficult. However, we believe that decentralising resource management makes this more tractable and permits the exploitation of application domain specific information and knowledge.

6 Related Work

Using economic models for resource management is certainly not a novel approach — [7] and [1] provide excellent overviews over past research in this area. Most of the systems described are auction-based and intended for more coarse-grained resource allocations. In [17] a proportional share scheduling algorithm is described in which tasks are charged for their resource usage and are refunded periodically. In [6] a system where objects are charged for backing store usage is presented. Charges are based on an analytical cost function of the storage utilisation. It is interesting to note that none of these systems base their charges on shadow prices, which have been demonstrated to be a useful mechanism to avoid congestion in communication networks.

Recently, two scheduling algorithms have been described, which proportion CPU resources based on feedback [22, 15]. Both schemes are based on real-time scheduling algorithms and deploy a PID controller to calculate the changes to the current resource allocation. In [22] applications provide feedback on their progress to the controller via *symbiotic interfaces* and the controller is used to dynamically adjust their resource reservations accordingly. The system described in [15] uses the deadline miss ratio of an EDF scheduler as the input to its feedback loop. Resulting adjustments to tasks' resource allocations are based on the tasks' discrete utility functions assumed to be known to the system. With the approach presented in this paper, the feedback is provided in the other direction – from the system to the applications, explicitly notifying applications that an adjustment is required and allowing them to adapt in an applications specific way. Applications may use utility functions to accomplish this task but it is not necessary to make utility functions explicit in the system.

QoS managers are usually deployed to manage the distribution of resources amongst competing tasks. The resource planner in Rialto [9] and the Q-RAM architecture [21] follow a centralised approach where tasks specify their resource requirements to a central entity responsible for resource management. Rialto deploys a simple negotiation protocol where tasks request a reservation and the planner either grants or rejects the request. In Q-RAM, tasks' requirements are based on a comprehensive description of all modes of operation of an application with associated user specified utility values. With complete knowledge of resource requirements and user preferences, the QoS manager then maximises the overall system utility. With our approach this exhaustive description can still be used by the applications to perform resource tradeoffs, but no central resource manager is required and the resource requirements do not need to be known in advance. We believe that this approach can yield similar optimal results. In a sense our approach is similar to AQUA [12] where applications are expected to collaborate through application-level QoS management libraries. Applications are given simple hints by the system on whether they can increase their resource usage or are *expected* to decrease it. AQUA expects applications to collaborate but does not enforce a reduction of resource usage in overload. Shadow prices, as described in this paper, provide a more expressive feedback signal to applications and the charging mechanism provides applications with an incentive to adapt.

7 Conclusions

In this paper we evaluated the application of a congestion pricing scheme proposed for congestion avoidance in communication networks to the problem of resource management in operating systems. Analogous to the networking environment, we treat missed deadlines as the external cost of congestion and provide, through shadow prices, an explicit feedback signal to applications causing the congestion. Through a credit system, applications are given the incentive to adapt to these signals. We have argued that this approach leads to a decentralised form of QoS management which does not require a central QoS manager.

We have demonstrated, through simulations, that the simple feedback provided by shadow prices can be used to implement a range of different adaption strategies and that different credit allocations by the users allow tasks to obtain different service levels. We feel encouraged by these results and will evaluate this approach more extensively, both in a more realistic simulated environment and in the context of a real system.

We have identified and discussed the issues which need to be addressed to extend this model to a generic resource management framework for real operating systems. In particular, future work will include an investigation of the interaction between user preferences and application adaption strategies in dynamic environments and the application of congestion pricing to more than one resource.

References

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

[2] K. Duda and D. Cheriton. Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *Proc. of the 17th Symposium on Operating Systems Principles (SOSP'99)*, pages 261–276, Kiawah Island Resort, SC, USA, December 1999.

[3] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. 1(4):397–413, August 1993.

[4] P. Goyal, X. Guo, and H. Vin. A Hierachichal CPU Scheduler for Multimedia Operating Systems. In *Proc. of the 2nd Symposium on Operating Systems Design and Implementation (OSDI'96)*, pages 107–121, Seattle, WA, USA, October 1996.

[5] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, December 1968.

[6] G. Heiser, F. Lam, and S. Russell. Resource Management in the Mungi Single-Address-Space Operating System. In *Proc. of the 21st Australasian Computer Science Conference*, Perth, Australia, February 1998.

[7] B. A. Huberman, editor. *The Ecology of Computation*. North-Holland, Amsterdam, Netherlands, 1988.

[8] M. Jones, D. Rosu, and M. Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proc. of the 16th Symposium on Operating Systems Principles (SOSP'97)*, pages 198–211, Saint-Malo, France, October 1997.

[9] M. B. Jones, P. J. Leach, R. Draves, and J. S. Barrera. Modular Real-Time Resource Management in the Rialto Operating System. In *Proc. of the 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, May 1995.

[10] F. Kelly, A. Maulloo, and D. Tan. Rate control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, 49(3):237–252, March 1998.

[11] P. Key, D. McAuley, P. Barham, and K. Laevens. Congestion pricing for congestion avoidance. Technical Report MSR-TR-99-15, Microsoft Research, Cambridge, U.K., February 1999.

[12] K. Lakshman, R. Yavatkar, and R. Finkel. Integrated CPU and Network-I/O QoS Management in an Endsystem. In *Proc. of the IFIP Fifth Int. Workshop on Quality of Service (IWQoS '97)*, 1997.

[13] 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.

[14] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, January 1973.

[15] C. Lu, J. Stankovic, G. Tao, and S. Son. Design and Evaluation of a Feedback Control EDF Scheduling Algorithm. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, Phoenix, Arizona, USA, December 1999.

[16] J. K. MacKie-Mason and H. R. Varian. Pricing the Internet. In B. Kahin and J. Keller, editors, *Public Access to the Internet*, pages 269–314. MIT Press, Cambridge, MA, USA, 1995.

[17] U. Maheshwari. Charge-Based Proportional Scheduling. Technical Memorandum MIT/LCS/TM-529, January 1995. MIT Laboratory for Computer Science.

[18] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. In *Proc. of the IEEE Int. Conference on Multimedia Computing and Systems*, May 1994.

[19] R. Neugebauer. How Elastic are Real Applications? In *Proc. of the 9th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'99)*, pages 197–200, Basking Ridge, NJ, USA, June 1999.

[20] J. Nieh and M. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. In *Proc. of the 16th Symposium on Operating Systems Principles (SOSP'97)*, pages 184–197, Saint-Malo, France, October 1997.

[21] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, San Francisco, USA, December 1997.

[22] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole. A Feedback-driven Proportion Allocator for Real-Rate Scheduling. In *Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99)*, pages 145–158, New Orleans, LA, USA, February 1999.

[23] I. Stoica, H. Abdel-Wahab, and K. Jeffay. On the Duality between Resource Reservation and Proportional Share Resource Allocation. In *Proc. of the Conference on Multimedia Computing and Networking*, pages 207–214, San Jose, CA, USA, February 1997.

[24] N. Stratford and R. Mortier. An Economic Approach to Adaptive Resource Management. In *Proc. of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, AZ, USA, March 1999.

[25] C. Waldspurger and W. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Mangement. In *Proc. of the 1st Symposium on Operating Systems Design and Implementation (OSDI'94)*, pages 1–11, Monterey, CA, USA, November 1994.