On the Duality between Resource Reservation and Proportional Share Resource Allocation

Ion Stoica* Hussein Abdel-Wahab† Kevin Jeffay††

ABSTRACT

We describe a new framework for resource allocation that unifies the well-known proportional share and resource reservation policies. Each client is characterized by two parameters: a *weight* that represents the rate at which the client "pays" for the resource, and a *share* that represents the fraction of the resource that the client should receive. A fixed rate corresponds to a proportional share allocation, while a fixed share corresponds to a reservation. Furthermore, rates and shares are duals of each other. Once one parameter is fixed the other becomes fixed as well. If a client asks for a fixed share then the level of competition for the resource determines the rate at which it has to pay, while if the rate is fixed, the level of competition determines the service time the client should receive.

To implement this framework we use a new proportional share algorithm, called Earliest Eligible Virtual Deadline First, that achieves optimal accuracy in the rates at which process execute. This makes it possible to provide support for highly predictable, real-time services. As a proof of concept we have implemented a prototype of a CPU scheduler under the FreeBSD operating system. The experimental results show that our scheduler achieves the goal of providing integrated support for batch and real-time applications.

Keywords: resource allocation, proportional share, reservation, scheduling, duality

1 INTRODUCTION

One of the most challenging problems in modern operating systems is the design of flexible and accurate schedulers to allocate resources among competing clients. This issue has become more important with the emergence of new types of real-time applications, such as multimedia, which have well defined time constraints. In order to meet these constraints the underlying operating system should allocate resources in a predictable and responsive way. In addition, a general-purpose operating system should seamlessly integrate these new types of applications with conventional interactive and batch applications.

To achieve this, the scheduler has to satisfy contradictory requirements such as flexibility and fairness on the one hand, and strict timeliness guarantees, on the other hand. Two of the most popular paradigms that try to address these requirements are *proportional share resource allocation*[1,7,12,13,18,19] and *reservation-based* algorithms.[10] While proportional share resource allocation achieves flexibility and ensures fairness, resource reservation offers better support for real-time applications. In this paper we propose a new framework which, by exploiting the dualism between reservation and proportional share allocation, retains the advantages of both paradigms.

In proportional share allocation each client is characterized by a parameter that expresses the *relative* share of a resource that the client *should* receive. For example, in lottery scheduling[18,19] a client's share is determined by the number of lottery tickets allocated to that client. Alternatively, in this paper, to express a client's share, we use the notion of *weight*. However, irrespective of the abstraction, in a dynamic system — one in which clients dynamically join and leave the competition — a client's share depends on both the current system state and the current time. It is state-dependent because it depends on all clients that compete for the resource, and it is time-dependent since the level of competition may change any

* Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162 (*stoica@cs.odu.edu*).

† Department of Computer Science, Old Dominion University, Norfolk, VA 23529-0162 (*wahab@cs.odu.edu*). Supported by the National Science Foundation (grant number CCR 95-9313857).

†† Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175 (*jeffay@cs.unc.edu*). Supported by grants from the Intel and IBM corporations, the National Science Foundation (grants CCR-9510156 & IRIS-9508514), and the Advanced Research Projects Agency (grant number 96-06580).

In: *Multimedia Computing and Networking 1997*,
SPIE Proceedings Series, Volume 3020, San Jose, CA, February 1997, pages 207-214.

time as new clients join or leave. These characteristics determine the two main properties of proportional-share resource allocation: *flexibility*, and *fairness*. The flexibility follows from the fact that clients are allowed to join and leave at any time, and no special restrictions are imposed on clients joining the competition. Fairness results naturally from the fact that ideally, the resource is always allocated in *proportion* to the competing clients' weights.

Unfortunately, not all the applications can tolerate the allocation uncertainty which characterizes proportional share algorithms. For example, to achieve a high quality sound from an audio-player application, it is necessary to play audio frames at precise intervals of time. However, this is not possible unless the application is guaranteed to receive *at all times* a minimum share of the CPU. Unfortunately, in a pure proportional-share allocation where the application's share of the CPU can be arbitrarily low, this is not possible.

Traditionally, these types of applications were scheduled by employing real-time algorithms, such as rate monotonic or earliest deadline first.[6] As observed in,[10] this is equivalent to *reserving* a *certain* share (percentage) of the resource. Unlike proportional share allocation, in the reservation approach the minimum share of a client is both state and time *independent*, *i.e.*, as long as the a client has reserved a certain share of the resource, it is *guaranteed* to receive at *least* that share independent of the level of competition for the resource. On the other hand, resource reservation sacrifices flexibility and fairness. In order to provide strict share guarantees, reservations impose strict admission policies which make it fairly restrictive. For example, a user can be in the situation in which he cannot run a new application, although he might be willing to accept a degradation in the performance of other applications in order to accommodate the new one.

Since modern operating systems have to provide support for a broad range of applications such as batch, interactive, and real-time applications, their schedulers must balance the often contradictory requirements of these applications. For example, while a proportional share scheduler would be better suited for batch applications, a reservation based scheduler would be more appropriate for real-time applications. In the remainder of this paper we propose a unified framework that integrates both the proportional share and the reservation based approaches to provide the benefits of both simultaneously.

2 COMPUTATION MODEL

We consider a set of clients that compete for a time-shared resource such as a processor or communications bandwidth. We assume that the resource is allocated in time quanta of size at most q. At the beginning of each time quantum, a client is selected to use the resource. Once the client acquires the resource, it may use it either for the entire time quantum, or it may release it before the time quantum expires. Although simple, this model captures the basic mechanisms traditionally used for sharing common resources.

In order to obtain access to a resource, a client must first issue a *request* which specifies the duration of the service time it needs. Once a client's request is fulfilled, it may either issue a new request, or become passive. For uniformity, we assume throughout that the client is the sole initiator of requests. However, in practice this is not necessarily true. For example, in the case of the CPU, the scheduler itself could issue requests on behalf of the client. In this case, the requested duration is either specified by the client, or the scheduler assumes a default duration.

3 THE WEIGHT-SHARE DUALISM

The basic idea behind integrating proportional share and resource reservation policies is very simple: Instead of characterizing a client exclusively by its weight, as in proportional share schemes, or by its share, as in reservation schemes, we use *both* characterizations simultaneously. That is, we associate a pair (w, f) with each client, where w represents a client's weight, and f represents the actual share of the resource the client should receive. Moreover, we have:

$$f = \frac{w}{W} \tag{1}$$

where W represents the total weight of all active clients. Thus if we fix the client's weight, then its share is given by Eq. (1). Alternatively, if the client asks for a fraction f of the resource, then, by using the same equation, the weight of the client can be easily computed as $w = W' f/(1 - f)$, where W' represents the sum of the weights over all the other active clients, *i.e.*, $W' = W - w$ (note that w appears both in the numerator and as a term in the sum W in the denominator in Eq.(1)). In this way, by exploiting the dualism between w and f, we can either achieve proportional share allocation (by fixing w), or resource reservation (by fixing f).

In a competitive environment it is often useful to have an accounting mechanism to evaluate the cost of a client using a resource. At a higher level, this information can be used to bill the client according to its resource usage and the type of service it receives. This information, or statistics computed from it, can also be used by a client to determine the appropriate level of service it should request (*i.e.*, reservation or proportional share allocation).

From an economic perspective, the client's weight can be viewed as being the *rate* at which the client has to "pay" for using the resource. In this way, the weight/share dualism translates naturally into a dualism between predictable cost and predictable service. When a client uses proportional share allocation, it will know exactly how much it has to "pay" over any interval of time — w times the duration of the interval — while the client is active . The client cannot, however, predict how much service time it will actually receive. This is because the fraction of the resource, and therefore the service time the client will receive, may change at any time depending on the level of competition for the resource. Alternatively, if a client decides to reserve a fraction of the resource, then it will know exactly how much service time it will receive over any interval — f times the duration of that interval — but it will not know how much it will be charged. Again, this is because the competition will dictate the rate at which the client must pay in order to maintain its share. Thus, a client has to decide between a predictable *cost* and a predictable *service*. By fixing the rate the client achieves a predictable cost, while by asking for a fixed share it achieves a predictable service. Moreover, in previous work[15] we have shown that this model exhibits the following intuitive properties: (1) it costs more for a client to allocate the same service time under the reservation class than under proportional share class, and (2) it costs more for a client to allocate the same service time over a shorter period of time.

4 A UNIFIED FRAMEWORK

Here we propose a scheme for exploiting the weight/share dualism for flexible real-time CPU scheduling. As shown in Figure 1, we use a two-level hierarchy to classify processes. At the higher level processes are split between proportional share and reservation classes. W_{prop} and W_{res} represent the sum of the weights over all active clients in the proportional share class and in the reservation class, respectively. Similarly, F_{prop} and F_{res} represent the total share of the clients in the proportional share and in the reservation class, respectively. Since the aggregate fraction of the resource that each class should receive is proportional to its total aggregate weight, we have

$$\frac{W_{prop}}{W_{res}} = \frac{F_{prop}}{F_{res}} \tag{2}$$

We assume a work-conserving system, *i.e.*, as long as there is at least one active client the resource cannot be idle. Thus, as long as there are active clients in the system the resource is fully utilized.

Next, we discuss how the weight and the share of each active client is affected when a new client joins the competition (the cases when a client leaves, or when the client's weight is changed are treated similarly). First, consider the case in which a client with weight w joins the proportional share class. Here W_{prop} increases by w. Since the shares of the clients in the reservation class are fixed (F_{res} and F_{prop} remain constant), from Eq (2) it follows that W_{res} increases by $w\ F_{res}/F_{prop}$. This can be achieved this by increasing the weight of every client in the reservation class by $w\ f'/F_{prop}$, where f' is the

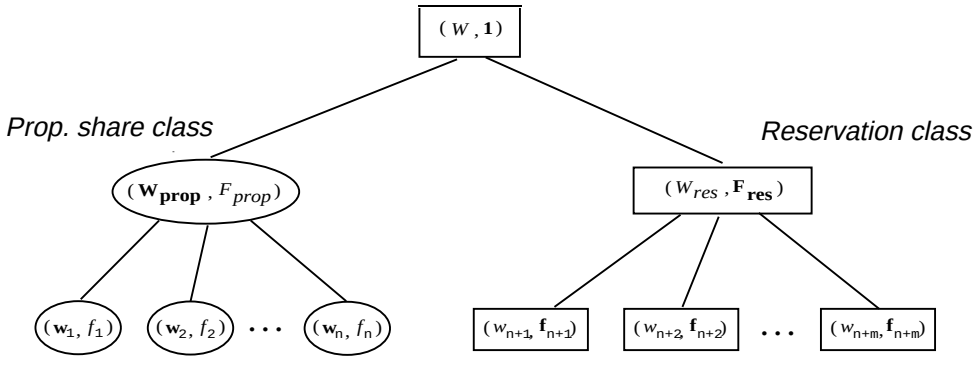


Figure 1: A system with $n + m$ active clients, where n clients belong to the proportional share class, and m belong to the reservation class. W_{res} and W_{prop} represent the aggregate weights of the active clients in the proportional share and reservation class, respectively; F_{res} and F_{prop} represent the aggregate shares in the proportional share, and the reservation class, respectively.

client's reserved share.

For clarity, consider the example in Figure 2. Initially, there are four active clients in the system: two (with weights 15 and 5) belonging to the proportional share class, and the other two (with shares 0.1 and 0.5) belonging to the reservation class. Figure 2(*b*) shows the same system after a client with weight 20 joins the proportional share class. The aggregate weight of the proportional share class W_{prop} increases to 40, and, in order to guarantee the reserved fractions, by Eq. (2) the aggregate weight of the reservation class W_{res} correspondingly increases to 60. In this way, the shares of the clients in the reservation class remain unchanged, while their weights increase from 5 to 10, and from 25 to 50, respectively. Thus, in this case, the price that the clients in the reservation class have to pay for maintaining their share doubles. Again, this expresses the fact that it is more expensive to get the same level of service when the competition increases. Conversely, note that while the weights of the clients in the proportional share class remain constant, their shares halve in order to accommodate the new client. In other words, for the same money you get less in a more competitive environment.

Next, assume that a client requesting a share f joins the reservation class. The aggregate share of the reservation class increases by f, while the aggregate share of the proportional share class decreases by f (recall that $F_{prop} + F_{res} = 1$). Since the weights of the clients in the proportional share class are fixed (and therefore W_{prop} remains constant), in order to guarantee the reserved shares, by Eq. (2), the aggregate weight of the reservation class W_{res} should increase by $f\ W_{prop}\ /F_{prop}$. Figure 2(*c*) shows the case when a client asking for a fraction equal to 0.2 of the resource joins the competition. Here the aggregate weight of the reservation class increases from 30 to 80. Similarly, the weights of all clients in the reservation class increase, while the shares of the clients in the proportional share class decrease accordingly.

Finally, note that the resulting shares and weights of all active clients are the same in both cases (see Figures 2(*b*) and 2(*c*)); in the first case, the new client joining the competition with weight 20 receives a share equal to 0.2, while in the second case a new client asking for a fraction equal to 0.2 of the resource corresponds to a weight equal to 20. This underlines again the weight/share dualism, in which each of the two parameters uniquely determines the other. Over time, the two scenarios may diverge as other clients leave or join the competition, however, in the first case the rate (w = 20) at which the new client has to pay for the resource does not change with the level of competition for the resource. In the second case the fraction (f = 0.2) of the resource allocated to the new client remains unchanged.

To implement this framework we have developed a new proportional share algorithm, called Earliest Eligible Virtual Deadline First (EEVDF).[16] The main advantage of this algorithm over other previous proportional share algorithms[7,13,18,19] is that it allocates resources more accurately. Specifically, we have shown that in a system in which no client request is larger than one time quantum, the difference between the service time that a client *should* ideally receive, and the service time it actually receives is bounded by the maximum size q of the time quantum. Moreover, this bound remains true in a system in which the clients are allowed to join and leave the competition dynamically. These bounds are optimal and, to the

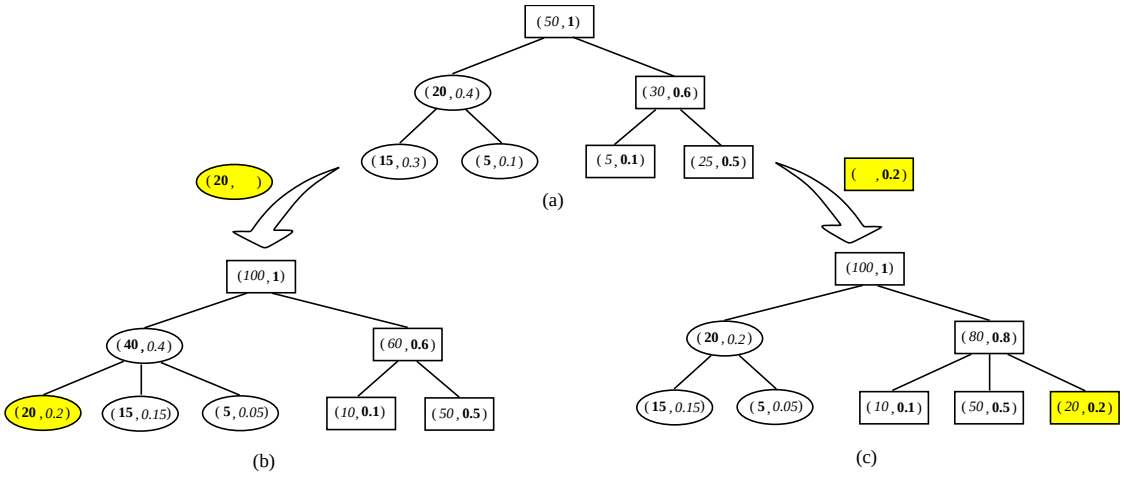


Figure 2: (*a*) shows a system with four clients: two in the proportional share class (with weights 15 and 5), and two in the reservation class (with shares 0.1 and 0.5). (*b*) shows the same system after a client with weight 20 joins the proportional share class. (*c*) shows the system after a client requesting a share of 0.2 joins the reservation class.

best of our knowledge, EEVDF is the first algorithm to achieve them in a dynamic system that provides support for both fractional and non-uniform quanta. (The difference between fractional and non-uniform quanta is that while in the first case the fraction of the time quantum that the client will actually use is assumed to be known in advance, in the non-uniform quanta it is not.)

This makes it possible to provide support for highly predictable services. For example, in a system with a time quantum of 10 msec, a client reserving 20% of the CPU is guaranteed to receive no *less* than 190 msec service time over any 1 second period. Moreover, it can be shown that in a fully preemptive system we are able to guarantee that a client in the reservation class will meet its deadline as long as its requests does not exceed its share. Although a similar result can be obtained by using classic real-time algorithms, such as Rate Monotonic and Earliest Deadline First,[6] we note that these algorithms do not provide any support for the proportional share class. Moreover, they are inflexible and perform poorly in overload situations.[14] In contrast, the EEVDF algorithm provides optimal performances for both proportional and reservation classes.[16]

5 EXPERIMENTS

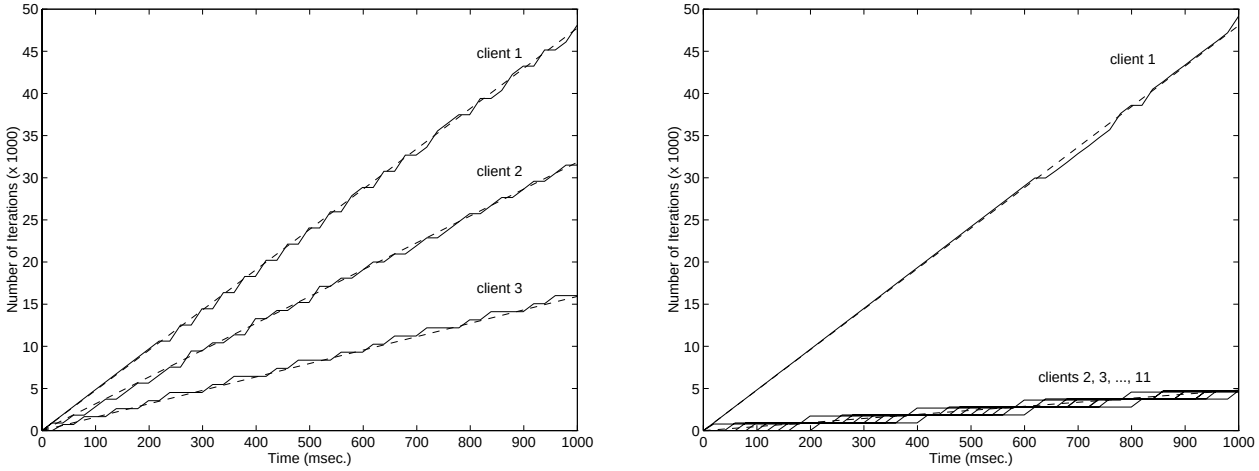
As a proof of concept we have implemented a prototype of a CPU scheduler under FreeBSD v. 2.0.1. All the experiments were run on a PC-compatible with a 75 MhZ Pentium processor and 16 MB of RAM.

Our scheduler coexists with the original FreeBSD scheduler. All the processes that request proportional share or reservation services are assigned a reserved (user-level) priority, and are handled by our scheduler. All the other processes are scheduled by the regular FreeBSD scheduler. Because of this, kernel processes are scheduled before any process in the proportional share or the reservation class. As this affects the accuracy of our allocation, in designing our experiments we have tried to eliminate these interferences as much as possible. Moreover, to achieve a better allocation accuracy we have reduced the time slice from 100 to 10 msec. Also, the size of all requests was chosen to be 10 msec. A client can interact with the scheduler via four system calls: **`setclientallinfo`**, **`setclientinfo`**, **`getclientallinfo`**, and **`getclientinfo`**. The first two calls are used to set/change the client parameters such as: weight/share, request size, and the amount of funds available. Similarly, the last two calls are used to obtain the values of client parameters.

In the first set of experiments (Figure 3) we consider the allocation accuracy in the context of the proportional share class. As a test program we use a modified version of the Dhrystone benchmark. In order to reduce interference with the kernel we removed all system calls from the program. In each experiment we ran several processes (clients) with different weights. To measure the application's progress we use a high priority process which, at every 20 msec, reads (via shared memory) the current number of iterations from each application. Figure 3(*a*) shows the number of iterations over one second for three clients with weights 3, 2, and 1, respectively. The dashed lines represent the ideal number of iterations for each client. The maximum measured difference between the ideal and the actual number of iterations over all clients was 738 iterations, while the minimum was -805 iterations. We note that since on the average an iteration takes roughly 10.4 μsec on our system, either difference accounts for less than one time slice (*i.e.*, 10 msec), which is consistent to our bounds.

In the next experiment we consider 11 clients (Figure 3(*b*)); one client having weight 10, and all the other having unit weights. We choose this setting because, as reported in literature[2,14,19] it proved to be a worst case setting for previous proportional share allocation algorithms.[13,19] In this case, the measured difference (between the ideal and the actual number of iterations over all clients) was between -1036 and 1044, which is within 10% of our theoretical bounds. The difference is due to inaccuracies in time measurements at both the system and user levels, as well as interference between the process which performs the actual measurements and the kernel. Moreover, we note that the above worst case values were the only ones to exceed the expected bounds during our experiment. In fact, the standard deviation is only 370 for the first client (with weight 10), and ranges between 267 and 290 for the clients with unit weights.

In the final experiment we consider a more complex scenario, in which we run three mpeg players, each displaying 530 frames from the same MPEG file. The player records the times when the frames are played. Figure 4 shows the cumulative number of frames displayed for each client over a 200 second period. Clients 1 and 3 belong to the proportional share class and have the weights 2 and 1, respectively; client 2 belongs to the reservation class and has a fixed share equal to 0.5. At time $t = 0$ client 1 joins the competition. Being the only active client it uses the entire CPU achieving a rate of 7.8 frames/second. After 18 seconds we start the second client. Since this client reserves 50% of the CPU, the remaining half is allocated to the first client. This can be visualized in Figure 4. Whenever clients 1 and 2 are the only active clients their



(*a*) (*b*)

Figure 3: (*a*) plots the cumulative number of iterations over a one second period for three processes executing the Dhrystone benchmark with weights 3, 2, and 1, respectively; (*b*) plots the cumulative number of iterations for 11 Dhrystone processes: one having weight 10, the others having unit weight.

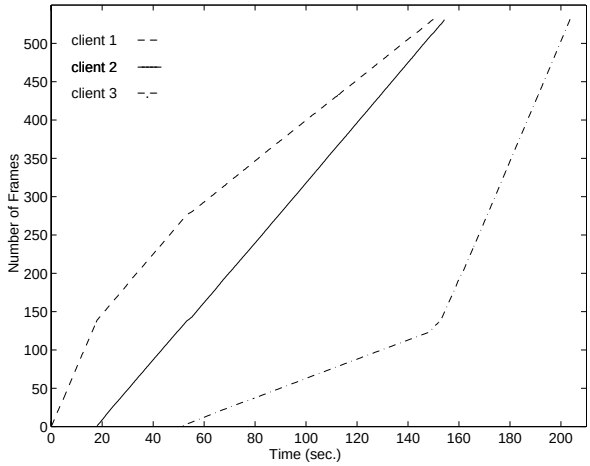


Figure 4: The cumulative number of frames displayed for three MPEG players. Clients 1 and 3 belong to the proportional share class and have the weights 2 and 1, respectively. Client 2 belongs to the reservation class and has a 0.5 share. Each client displays 530 frames after which it leaves the competition.

rates are identical. Next, at time $t = 52$ seconds we start the third client with weight 1. Note that since client 2 belongs to the reservation class, its share does not change when client 3 becomes active. The remaining share is redistributed in proportion 2:1 among the other two clients. At time $t = 153$ seconds the first client finishes playing the entire content of the file and leaves the competition. Since the share of the second client does not change, the entire share of client 1 will be inherited by client 3. Finally, after another 3 seconds client 2 leaves the competition, leaving client 3 as the only one active. As it can be seen in Figure 4 the rate of the second client remains steady over the entire period. This shows that employing a reservation mechanism is effective even in a more "real" setting in which the interactions with other UNIX sub-systems, such the X-server, are considered.

For simplicity, in the current prototype we use a list data structure to represent the state of resource allocation in the system. To quantify the overhead we ran ten Dhrystone benchmarks under both the modified and original FreeBSD scheduler. Each client was assigned the same weight and ran for one million iterations. Under these conditions, our scheduler was only 0.7% slower on average. In addition, we have simulated the effect of using a more sophisticated tree based data structure.[14] To gauge its performances we randomly constructed trees containing as many as 2^{16} requests. For a tree containing 1000 nodes, a search operation (the most common one) took on average 2.2 μsec; searching a tree containing 2^{16}

nodes took 4.8 μsec on average. On the other hand, insertion and deletion were consistently slower (but within a factor of 2). This is due to the additional overhead incurred by re-balancing the tree.

6 SUMMARY & RELATED WORK

There has been a great deal of research into the problems of providing integrated support for batch, interactive, and real-time and multimedia activities. The dominant approach has been to use a real-time scheduling algorithm, such as rate-monotonic or earliest deadline first, for scheduling real-time activities, and a general propose scheduler, such as decay-usage scheduling, for the other activities.[3,4,10] Although this approach guarantees strong timeliness properties for real-time activities, the level of control over other activities is poor. Moreover, the real-time algorithms impose admission policies which make them restrictive and inflexible in a dynamic system.

In contrast, proportional share schedulers[1,7,11-13,18,19] are flexible and ensure fairness among competing clients, however, they cannot accommodate real-time applications that require precise guarantees in order to meet their deadlines. One notable exception is lottery scheduling which, by using the currency abstraction,[18,19] makes it possible to integrate resource reservation and proportional share allocation in the lottery scheduling framework.[20] However, our approach differs in several key aspects. First, while our scheme uses an exhaustive accounting mechanism — an application is charged for both the share it receives and the time it uses the resource — lottery scheduling accounts only for the instantaneous share (case in which an additional layer is eventually needed for accounting purposes). Second, our framework separates explicitly the concepts of relative and fixed shares, and exposes the basic tradeoffs between the service quality and its cost, which we believe makes it easier for an application to choose the appropriate level of service.

By integrating resource reservation and proportional share allocation in a common framework, we inherit the key benefits of both paradigms. We achieve fairness by charging each client in proportion to the percentage of the resource it uses. Since there is no admission policy for the proportional share clients, and since any client may join or leave any time, our approach is also flexible. Moreover, the resource guarantees provided to the reservation class are strong enough to accommodate a broad range of real-time applications. This is made possible by using a new proportional share algorithm (EEVDF) that achieves optimal accuracy in a dynamic system, while providing support for both fractional and non-uniform quanta.[16] Finally, we note that our reservation mechanism exhibits better behavior in overload situations than previous reservation mechanisms based on traditional real-time scheduling algorithms.[10]

Another approach for resource allocation is that of *microeconomic* scheduling.[8,9,17] A microeconomic scheduler uses an "auction" mechanism to allocate resources among competing clients. At the beginning of every time-slice, the scheduler initiates an auction. The client offering the highest bid acquires the resource for the next time-slice. Although this scheduling scheme successfully solves the resource allocation problem in distributed environments, it is too complex to efficiently implement fine-grained resource control.

Fong and Squillante have proposed a new scheduling discipline called Time-Function Scheduling (TFS),[5] where a client's priority is defined by a time-dependent function. TFS provides effective and flexible control over resource allocation and it can be used to achieve general scheduling objectives such as relative per-class throughputs and low waiting time variance. Although TFS can also achieve proportional share allocation by assigning an equal share to each client in the same class, the algorithm's accuracy depends on the frequency at which priorities are updated. Since the update operation is rather expensive this may limit the allocation accuracy that can be achieved.

7 CONCLUSIONS

Two of the most popular classes of allocation policies for shared resources are proportional share and resource reservation. In this paper we have described a new framework that unifies these two classes. Instead of characterizing a client solely by its *weight*, as in proportional share allocation, or by its *share*, as in reservation, we characterize it by *both* parameters. Our scheduling framework builds on the *duality* of these two parameters, in which each one of them uniquely determines the other one. The key observation is that we can obtain any instance of the two allocation policies by simply setting one parameter or the other: by fixing the weight we achieve proportional share allocation, while by fixing the share we obtain resource reservation. Finally, by employing the Earliest Eligible Virtual Deadline First algorithm,[16] we provide support for a large class of real-time applications, which was not possible with the previous proportional share algorithms.[7,13,18,19]

8 REFERENCES

[1] S. K. Baruah, J. E. Gehrke and C. G. Plaxton, "Fast Scheduling of Periodic Tasks on Multiple Resources," *Proc. of the 9th International Parallel Processing Symposium*, April 1995, pp. 280-288.

[2] J. C. R. Bennett and H. Zhang, "WF^2Q: Worst-case Fair Queueing," INFOCOM '96, San-Francisco, March 1996.

[3] G. Bollela and K. Jeffay, "Support for Real-Time Computing Within General Purpose Operating Systems," *Proc. of the IEEE Real-Time Technology and Applications Symposium*, Chicago, May 1995.

[4] G. Coulson A. Campbell, P. Robin, G. Blair, M. Papathomas and D. Hutchinson, "The Design of a QoS Controlled ATM Based Communication System in Chorus," *Internal Report* MPG-94-05, Department of Computer Science, Lancaster University, 1994.

[5] L. L. Fong and M. S. Squillante, "Time-Function Scheduling: A General Approach for Controllable Resource Management," *Technical Report* RC 20155, IBM Research Center, 1995.

[6] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, January 1973, pp. 46-61.

[7] U. Maheshwari, "Charged-based Proportional Scheduling," Technical Memorandum, MIT/LCS/TM-529, Laboratory for CS, MIT, July 1995.

[8] T. W. Malone, R. E. Fikes, K. R. Grant and M. T. Howard. "Enterprise: A Market-Like Task Scheduler for Distributed Computing Environments," *The Ecology of Computation*, B. Huberman (ed.), North-Holland, 1988, pp. 177-205.

[9] M. S. Miller and K. E. Drexler. "Markets and Computation: Agoric Open System," *The Ecology of Computation*, B. Huberman (ed.), North-Holland, 1988, pp. 133-176.

[10] C. W. Mercer, S. Savage, and H. Tokuda, "Processor Capacity Reserves: Operating System Support for Multimedia Applications," *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[11] J. Nieh and M. S. Lam "SMART: A Processor Scheduler for Multimedia Applications," *Proc. of SOSP-15*, Dec. 1995, pp. 233.

[12] J. Nieh and M. S. Lam, "Integrated Processor Scheduling for Multimedia," *Proc. of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, N.H., April, 1995.

[13] I. Stoica and H. Abdel-Wahab, "A new approach to implement proportional share resource allocation," *Technical Report* TR-95-05, CS Dept., ODU, April 1995.

[14] I. Stoica and H. Abdel-Wahab, "Earliest Eligible Request First: A Flexible and Accurate Mechanism for Proportional Share Resource Allocation," *Technical Report* TR-95-22, CS Dpt., ODU, 1995.

[15] I. Stoica and H. Abdel-Wahab, "On the Duality between Resource Reservation and Proportional Share Resource Allocation," TR-96-19 *Technical Report*, CS Dept., ODU, 1996.

[16] I. Stoica, H. Abdel-Wahab, and K. Jeffay, "A Proportional Share Resource Allocation Algorithm For Real-Time, Time-Shared Systems," *IEEE Real-Time Systems Symposium*, Dec. 1996, to appear.

[17] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart and W. S. Stornetta. "Spawn: A Distributed Computational Economy," *IEEE Transactions on Software Engineering*, Vol. 18, No. 2, Feb. 1992, pp. 103-117.

[18] C. A. Waldspurger and W. E. Weihl, "Lottery Scheduling: Flexible Proportional-Share Resource Management," *Proc. of the First Symposium on Operating System Design and Implementation*, Nov. 1994, pp. 1-12.

[19] C. A. Waldspurger, "Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management," *Ph.D. Thesis*, Laboratory for CS, MIT, September 1995.

[20] C. A. Waldspurger and W. E. Weihl, *personal communication*, May 1996.