Why Using the Request Abstraction in Proportional Share Allocation Systems is Useful ?

Ion Stoica * Hui Zhang † Kevin Jeffay‡

Abstract

In the recent years, the proportional share schedulers have emerged as a viable alternative for integrating the new multimedia applications with conventional interactive and batch applications. Traditionally, these schedulers use the notion of time quanta for characterizing the service time allocated to a client. In this paper we show that by adding a higher level abstraction that quantifies the service time requested by a client, we can improve the average client response time, and reduce the number of context switches.

1 Introduction

One of the most challenging problems in modern operating systems is to design flexible and accurate algorithms 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.

In the past years, the proportional share schedulers have emerged as a viable solution to this problem [1, 2, 4, 9, 7, 11, 14, 15]. Although, in general, proportional share schedulers offer weaker guarantees for applications with timeliness constraints than the traditional real-time based schedulers [6, 7], they tend to be more flexible and ensure a graceful degradation in overload situations. In a proportional share system, each client is characterized by a *weight* that determines the share of the resource that the client should receive. The scheduler tries to allocate the resource among competing clients in proportion to their share. We define the service time that a client should receive, as the service time which the same client would receive in an ideal system in which all the active clients are serviced at *the same time*, in proportion to their weights. We note that this model is equivalent to a simple round-robin scheduler which allocates the resource to the active clients in arbitrarily small time quanta.

Unfortunately, in many practical situations, time quanta cannot be taken arbitrarily small. One of the reasons is the overhead introduced by the scheduling algorithm and the overhead in switching from one client to another: taking time quanta of the same order of magnitude as these overheads could drastically reduce the resource utilization. For example, it would be unacceptable for a CPU to spend more time in scheduling a new process, and context switching between the processes, than doing useful computation. Another reason is that some operations cannot be interrupted, i.e., once started they must complete in the same time quanta. For example, once a communication switch begins to send a packet for one session, it cannot serve any other session until the entire packet is sent. Therefore, similarly to other proportional share allocation algorithms [1, 2, 4, 9, 7, 11, 14, 15], we assume that the resource is allocated in discrete 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 [12].

*School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891 (istoica@cs.cmu.edu).

†School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891 (hzhang@cs.cmu.edu).

‡Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175 (jeffay@cs.unc.edu).

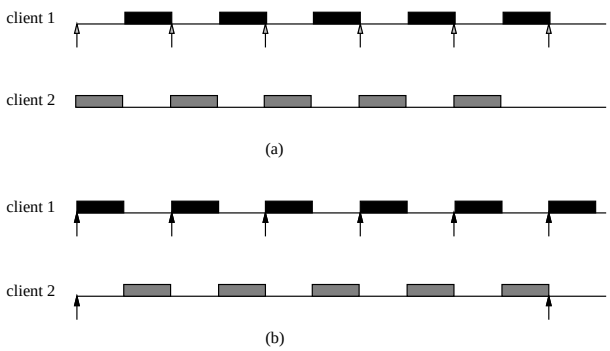

Figure 1: *A proportional share system with two active clients having equal weights. The first client is an event-driven periodic process that requires one time unit to process each event. The second client is an intensive computation job. (a) shows a possible scheduling instance for the two clients, under the assumption that the scheduler does not have any information about the clients' requests. (b) shows a possible scheduling for the same clients, when the scheduler uses information about clients' requests. As a result, the response time of the first client is halved. (The shaded arrows in (a) represents the times when the external events occur; the filled arrows in (b) represents the times when the requests are issued by the two clients.)*

To the best of our knowledge, all the previous proportional share allocation algorithms[1, 2, 4, 9, 7, 14, 15] use the time quanta (or an equivalent abstraction), as the only abstraction for characterizing the service time allocated to a client. In the next section we show that by adding a higher level abstraction that quantifies the service time requested by a client can improve the average client response time, as well as the system utilization.

2 The Request Abstraction

In our model, in order to obtain access to a resource, a client must 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, throughout this paper we assume that the client is the sole initiator of the requests. However, in practice this is not necessarily true. For example, in the case of the CPU, the scheduler itself could be the one to issue the requests on behalf of the client. In this case, the request duration is either specified by the client, or the scheduler assumes a default duration. This allows us to treat all continuous media, interactive, and batch activities in a consistent way.

It is important to note that the size of a request determines indirectly the allocation accuracy: the higher the request duration is, the lower the allocation accuracy is. This is because the semantic of servicing a request of size r issued by a client with share f is to allocate r time units to the client over a $\frac{r}{f}$ time units interval.[1] This definition does not specify how the service time is allocated over that interval; any allocation is acceptable as long as the client receives r time units during the interval. Clearly, this freedom may result in large disparities in the service time received by a client over shorter time intervals, which has direct impact on the allocation accuracy [13]. Next, we show how by using the request abstraction we can reduce the average response time of a client by properly choosing the request durations for the competing clients. Intuitively, this is possible because the request duration gives more information to the scheduler about the client requirements, which in turn gives opportunity for optimizing the allocation.

To make things clear, we begin with a simple example. Consider two clients with equal weights, and assume that the time quanta is of unit size. Further, assume that the first client is a periodic process which receives external events every two time units, and needs one time unit to process each event, while the second client is a computation intensive job. First, consider the case when the two clients are scheduled by a proportional share algorithm which assumes that the clients are characterize *only* by their weights. In this case, since the scheduler

[1] For simplicity, here we assume that there are no clients joining or leaving the competition to the resource while the request is pending. Therefore the share of the client remains unchanged.

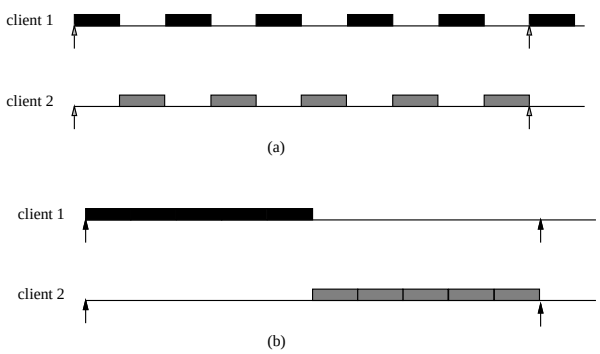

Figure 2: *A proportional share system with two active clients having equal weights. Each client represents a computation intensive job. (a) shows a possible scheduling instance for the two clients, under the assumption that the scheduler does not have any information about the clients' requests. (b) shows a possible scheduling for the same clients, when the scheduler uses information about clients' requests (here, each client issues request with duration of five time units.) As a result, the number of context switches reduces five times.*

has no information about the degree of fairness required by each application, the only thing it can do is to ensure the *same* degree of fairness to *both* clients. Figure 1.(a) shows a possible optimal scheduling, in which, at any given time, the difference between the service time that a client should receive in the ideal system, and the service time it actually receives in the real system is no larger than one time quanta. Observe that from the first client point of view, the response time for its requests is two time units. Our basic approach is to improve the response time of the first client, at the expense of possibly decreasing the allocation accuracy of the second one. In our example, this approach makes perfect sense, since usually an intensive computation job does not needs a very fined grained allocation accuracy.

Next, we show how the concept of request helps us in achieving this goal. We consider that the first client issues requests with duration of one time unit (to mirror the external events received by the client), while the second client issues requests with duration of five time units. Recall that the request duration determines the degree of fairness in the service time allocated to that client. Specifically, in the second client case, this is equivalent to ask for five time units of service time over a ten time units interval, without making any assumption of how the service time is actually allocated during this interval. Taking into account the requests' durations, Figure 1.(b) shows a possible scheduling for the two clients.[2] Thus, in this case, the first client achieves a response time of just one time unit; two times better than in the previous case.

The reason we are able to gain by using the request abstraction is more clear if we consider a specific algorithm, i.e., the Earliest Eligible Virtual Deadline First (EEVDF). In EEVDF, to each request is associated a virtual eligible time, and a virtual deadline which are the corresponding starting and finishing times of servicing the request in an ideal model. EEVDF simply allocates a time quanta to the client that has the eligible request with the earliest virtual deadline. Consequently, in EEVDF, a request is fulfilled only after all the other requests with smaller virtual deadlines have been fulfilled. Thus, there is a high chance that a short request to be fulfilled earlier in a system in which the other clients' requests are larger. This is because in the case of larger requests, there are fewer virtual deadlines over the same time interval, and therefore a shorter request will have to wait, on the average, after fewer requests to be fulfilled. It is important to note that this improvement refers only to the average case; in the worst case – when all the requests have the same virtual deadline – the response time is *independent* of the requests' durations.

Finally, to quantify the improvement achieved by using the concept of request in EEVDF, consider a system that runs an audio application that receives audio packets at a rate of 20 packets/sec and plays an audio packet for 10 msec, and four computation intensive jobs. Further, assume that all the applications have equal weights. Similarly to the previous example, we associate to the audio application a request of 10 msec, while to each computation application we associate a request of one second. Then, it can be shown that in this case, on the average, an audio

[2]In fact, this is the exact schedule generated by Earliest Eligible Virtual Deadline First (EEVDF) algorithm, when the ties are broken in the favor of the smallest request.

request has to wait for 0.04 requests to be fulfilled. On the other hand, if we ignore the request size, and ask that all the clients to get the same treatment,[3] then, on the average, an audio request will have to wait for *two* requests to be fulfilled.

Besides improving the response time, using the request abstraction may significantly reduce the scheduling overhead, as well. This is mainly because by using the request duration as the main indicator of the allocation accuracy, the scheduler may reduce the number of context switches. As an example, consider a system in which the resource is allocated in unit time quanta, and two active clients with equal weights that issue requests with the duration of five time units. Then, again if we do not use the concept of request at the scheduler level, the scheduler has to ensure the best possible allocation accuracy for both clients. As shown in Figure 2.(a) this results in a context switch every time unit. On the other hand, by taking into account the durations of the request, the scheduler needs only to perform one context switch per request, which is five times less than in the previous case (see Figure 2.(b)).

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. 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.

[4] P. Goyal, X. Guo and H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems", to appear in *Proc. of the 2nd OSDI Symp.*, October 1996.

[5] 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.

[6] 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

[7] 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.

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

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

[10] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach To Flow Control in Integrated Services Networks-The Single Node Case", *Proc. of IEEE INFOCOM'92*, 1992, pp. 915–924.

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

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

[13] I. Stoica, H. Abdel-Wahab, and K. Jeffay, "A Proportional Share Resource Allocation Algorithm For Real-Time, Tim-Shared Systems", *Real-Time Systems Symposium*, to appear., Dec. 4-6 1996.

[14] 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.

[15] C. A. Waldspurger, "Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management," *PhD Thesis*, Laboratory for CS, MIT, Sep. 1995.

[3] This is equivalent to the situation in which all the computation jobs issue 10 msec requests.