Portable and adaptive specification of disk bandwidth quality of service

Stephen Childs *
University of Cambridge Computer Laboratory
New Museums Site, Pembroke St.,
Cambridge CB2 3QG UK.

E-mail: Stephen.Childs@cl.cam.ac.uk

Abstract

Continuous media (CM) applications require Quality of Service (QoS) guarantees from the disk, as well as from the CPU and network. Their requirements can often be expressed in terms of data rates, which are portable descriptions of resource needs. QoS-aware disk schedulers need an accurate predictor of their clients' resource usage in order to perform admission control and limit the effect of misbehaving clients. We present a scheme which allows the client to directly specify its desired data rate to the scheduler. A per-client time limit bounds the amount of time spent trying to achieve this rate. By using feedback from the scheduler, the client can renegotiate its time limit. The result is a simple and adaptive method of determining resource requirements.

1 Introduction

As PCs start to be used for disk intensive applications such as video editing and audio recording, the limitations of conventional disk scheduling become apparent. Multimedia operating systems incorporate QoS-aware scheduling algorithms which can honour QoS guarantees. However, the form in which they specify resources may not suit applications very well.

Many applications processing continuous media streams require guaranteed bandwidth from the disk. They need to know that they can exchange data with the disk at a certain rate for the duration of their execution. This bandwidth requirement can be expressed as (p, b), where p is the period, and b the number of bytes to be transferred during that period.

A resource specification of this kind provides portability. Applications need know nothing about the capabilities of the underlying hardware. It is the system's responsibility to translate the client's request into the appropriate low-level parameters.

However, from the system's point of view, this is not as straightforward as just ensuring that the number of disk blocks transferred each period remains constant. The cost in real terms (i.e. time) of getting each block of data from disk will vary hugely, depending on the size of the request, the time taken to move the disk head to satisfy the request, the speed of the disk and processor, the disk interface, etc.

There is no simple way for the system to tell from a bandwidth guarantee how much time it will need per period to fulfil that guarantee. It might be possible to calculate service times in advance using knowledge about the disk, but constructing an accurate model of a disk drive is not an easy task, as even seemingly trivial assumptions can result in large inaccuracies [6].

2 Existing QoS scheduler

Disk-scheduling has been an active area of research for decades. Historically, the focus was mainly on increasing throughput by minimising seek overheads. Recently the need to provide guaranteed service times for CM streams has become a factor.

The disk scheduling algorithm used in the Nemesis operating system [2] aims to enforce QoS guarantees. Clients of the disk specify their QoS requirements as a tuple of the form (p, s, x, l).[1] The p and s parameters specify the period and time-slice for a client. A guarantee of this kind means that a client will be allowed to spend a maximum of s ms performing disk transactions during each period of p ms.

Clients are periodically allocated s ms and a deadline of $now + p$ ms, and are scheduled using a modified EDF algorithm, known as *Atropos* [1]. Because of the difficulty of estimating the cost of a disk transaction in advance, the system accounts the time spent by a client in arrears.

*The author is funded by an ICL Studentship.

[1] The x and l parameters are not used in this work, and will not be referred to again.

This means that the transfer of one particular data unit may take longer than the allotted time, but this will cause the client to be descheduled until it receives a new allocation of time at the start of its next period.

Because the scheduler is accounting time, the total amount of resource is known, and so admission control can be simply performed by calculating what percentage of the total time is still uncontracted.

This time-slice/period notation is very familiar from the field of real-time systems, where the values are usually determined in advance by human experts. This is not feasible in the case we are interested in here, where applications will have to run on many different machines.

3 Accounting blocks *and* time

We have established the need for two descriptions of the disk resource requirements of clients. Firstly, a data rate specified by the client and secondly, a measure of the time consumed per period, used to limit clients and perform admission control. We also need a method of tying these descriptions together.

A first attempt at providing support for specifying data rates might be to use QoS specifications describing the number of blocks to be transferred in a given period. The scheduling algorithm would then be modified to account blocks rather than time, and otherwise left unchanged.

The problem with this approach, as described earlier, is the difficulty of establishing where the overload point of the system is. Two guarantees for the same amount of blocks may in fact cause the system to do hugely different amounts of work. It is clear that this needs to be taken into account.

We have made this possible by accounting *both* the time used and number of blocks transferred per period. When a client starts, it specifies a bandwidth aspiration indicating the data rate it would like to achieve, and also an initial time limit which specifies the maximum amount of time it is allowed to work for in each period.

The disk scheduler starts attempting to satisfy the client's requests. It performs disk transactions and accounts the time taken and number of blocks transferred. For periods when the time limit is sufficient, it is the bandwidth specification that limits the amount of work done. As before, if the client's remaining time (or block) allocation falls below zero, it cannot run again until it is next allocated time.

At the end of each period, the scheduler informs the client of how much time it has used by invoking a callback function (provided by the client when it opens its connection). This gives the client an opportunity to respond, dealing with the situation as it sees fit. For example, in the case where the client has overrun its limit the policy currently implemented renegotiates the QoS contract to obtain a larger time limit. Figure 1 illustrates the feedback process. As the amount of work done by each stream is still bounded

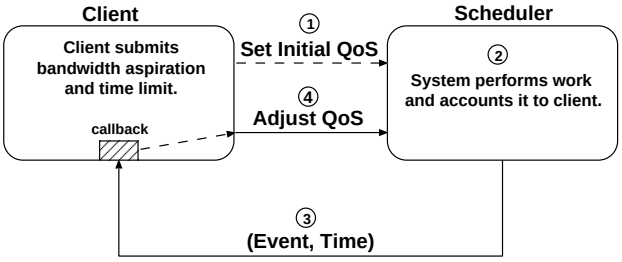

Figure 1. Interaction between client and disk scheduler

by its time limit, the system can determine what percentage of its resources are still available and so perform admission control.

The problem of providing isolation remains, as applications making many seeks will cause others to need more time as well. We need to make sure that the data rate of an application whose stream has reached a stable state is not compromised by new streams. One method might be to assign a higher importance [3] to established streams. This will be used to protect them by ensuring that in a conflict for time, they will always have priority over a newly-entered stream.

4 Example

To illustrate how this scheme works in practice, we will use the example of a variable rate video application which plays video at different resolutions (and hence bandwidths). For the first phase of its operation it plays at 1.5 Mbit/s, doubling to 3 Mbit/s after 100s and to 6 Mbit/s after another 100s. This is simulated by a simple program that reads a block from disk, pauses (to simulate processing of the data), then reads again.

The application specifies a bandwidth requirement of 1.5 Mbit/s (384 disk blocks per second). It doesn't provide an estimate for the time slice, which it initialises to 0. When the client is scheduled, the first unit of data is transferred. The time taken to do this is obviously greater than the client's current time limit (0 ms/period). The scheduler invokes the callback function in the client, notifying it that an `OutOfTime` event has occurred.

This client's policy on receipt of an `OutOfTime` event is to increase its time limit by 5% of the period. However, this limit is still not sufficient, and the client is called back again after the next transfer. After a number of iterations, the client's time limit allows it to complete the amount of work specified, and becomes stable. Figure 2 shows this process.

At $t = 100s$, the client doubles its bandwidth specification and begins to request data from the disk at a higher rate. Because the system is now trying to do twice as much work per period, the old time limit is no longer sufficient,

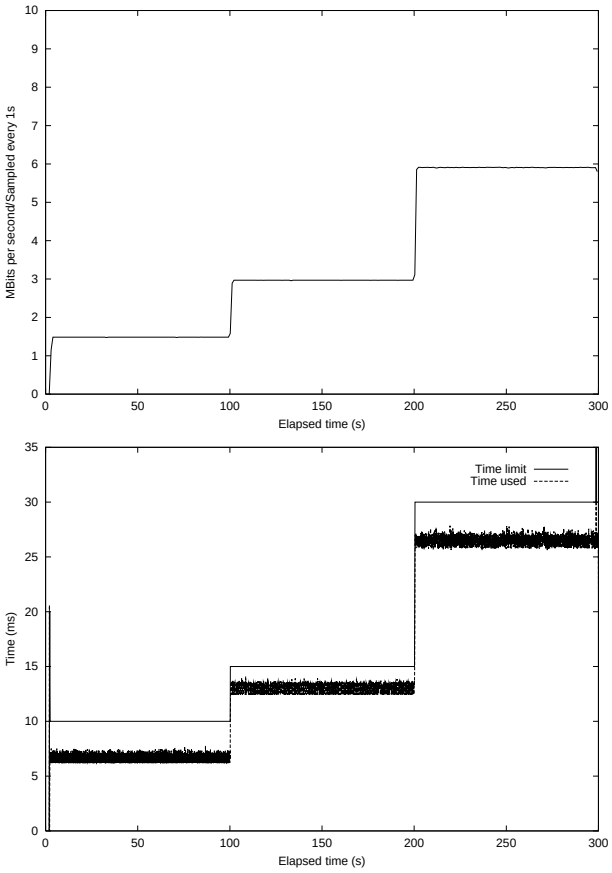

Figure 2. Throughput (*top*) and time used (*bottom*) for an adapting client

and a feedback and renegotiation process similar to that performed on start-up takes place. This occurs again at $t = 200s$.

Also note that at the beginning of playback, the application plays slowly, and then there is a spike in its time allocation as it tries to compensate for this. The scheduler invokes the callback function with a `TooMuchTime` event, which causes the client to reduce its guarantee.

5 Related work

The real-time filesystem in RT-Mach [5] uses similar QoS specifications. Their paper mentions the possibility of specifying bandwidth portably in disk blocks, but assumes contiguous file layout.

The Cello [7] scheduler has been designed as part of a multi-service filesystem and provides separate scheduling classes for different file types.

Adaptive schemes for resource allocation are not new to the literature. Work such as that done by Walpole [3] and Jeffay [4] has established the usefulness of rate-based execution models and callback-based adaption for multimedia scheduling.

6 Further work and conclusion

At the moment, the scheduler returns an event and the amount of time remaining to the client. We need to determine whether this is sufficient for the client to make an informed decision about what action to take.

The adaption policies defined by clients determine the usefulness of the scheme. Obviously the simple policy presented here is limited, and better ones could be developed which benefit from the body of work already done on adaptive systems. Modules can be built which provide predefined policies for standard file types.

Our aim in this work was to make it easier for applications to specify their resource requirements in a portable way. The combination of block- and time-based accounting used here makes this possible while preserving the ability to do effective admission control and limit the resource consumed per client.

References

[1] Paul Barham. A fresh approach to file system quality of service. In *Proceedings of the 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, April 1997.

[2] Ian Leslie et al. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communication*, 14(7):1280–1297, September 1996.

[3] Jonathan Walpole et al. A feedback-driven proportion allocator for real-rate scheduling. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation*, pages 145–158, February 22-25 1999.

[4] G. K. Jeffay and D. Bennett. A rate-based execution abstraction for multimedia computing. In *Proceedings of 5th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, pages 67–78, April 18-23 1995.

[5] Anastasio Molano, Kanaka Juvva, and Ragunathan Rajkumar. Real-time filesystems: Guaranteeing timing constraints for disk accesses in RT-Mach. In *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, December 1997.

[6] Chris Ruemmler and John Wilkes. An introduction to disk drive modelling. *IEEE Computer*, 27(3):17–28, March 1994.

[7] Prashant J. Shenoy and Harrick M. Vin. Cello: A disk scheduling framework for next-generation operating systems. In *Proceedings of Sigmetrics '98, the International Conference on Measurement and Modeling of Computer Systems*, June 1998.