Streaming *vs*. Latency in Information Mass-Transit

Philip Machanick
Department of Computer Science
University of the Witwatersrand
2050 Wits, South Africa
philip@cs.wits.ac.za

Abstract. At many levels of computer system design there is a severe mismatch between ease of achieving bandwidth and latency goals. This paper suggests a new approach to achieving user-level latency goals, by focusing on streaming data as fast as possible and sampling the streamed data to satisfy individual requests. By analogy with transportation, the proposed approach is a mass-transit approach, by contrast with conventional strategies which are closer conceptually to a a single person in a car. Two examples are given: disk delay lines, modelled on one of the oldest kinds of computer memory, and a scalable video on demand (VoD) architecture.

1.0 Introduction

The most important latency goals often relate to response times seen by users, which are slow by computer standards, but scaling up to large numbers of users presents a problem. Examples include transaction-based systems and web sites. Typical video streaming requirements are a reasonable fit to best-case properties of a hard drive if a single stream is supported, but are not a good fit to switching rapidly between multiple streams (high bandwidth but also high latency).

This paper introduces the idea that latency requirements can be faked by wasting bandwidth.

The essential idea is analogous to mass transit versus personal transport. A car seems the more efficient way of taking an individual to a desired destination (if cost is not an object) than a bus, since it can be operated to the specific requirements of the individual traveller. However, if everyone separately drives a car, roads become clogged. In principle, roads could be scaled up to handle the traffic, but buses, car-pooling and trains are the more efficient solution. The loss of efficiency in scheduling for the individual is offset by the efficiency of requiring less road space. As long as the mass transit systems cover enough of the potential routes to do most of the job a car would do, the end result is potentially a faster journey.

Two examples are presented here to illustrate the point. In both cases, a large amount of bandwidth appears to be wasted. However, a more traditional model would waste more time repositioning disk heads, or initiating new network transactions. In this sense, latency is faked by wasting bandwidth, but the overall effect is more efficient than the traditional approach—just as a well-configured bus system is more efficient than a purely car-based system, even if the occasional empty bus may seem wasteful.

The two examples explored here are an alternative disk architecture for high-volume transaction-based systems, and a scalable video on demand architecture. Both represent cases where adding latencies of many separate transactions results in a very low overall rate of transactions, before latency goals become hard to achieve.

2.0 Disk Delay Lines

In a transaction-based system (perhaps a traditional system such as airline reservations, or a large web site; with the growth of internet commerce, there is convergence between the concepts), requirements for user-level latencies are typically in the region of 1 to 10 seconds, depending on the system. Given that internet users are used to long waits, even longer latencies are sometimes tolerable. However, scaling up a latency requirement of the order of 1 second to thousands of transactions per second runs up against the relatively high latency of a disk. If all the requests go to one disk, even with a relatively low latency (access time) of 7ms, the disk alone cannot support more than 142 transactions per second, ignoring any other latencies (operating system, processing the request, network overheads, etc.).

The proposal here is to rethink the basis on which transaction-processing systems use a disk to attempt to use the more easily scalable property, bandwidth, to achieve user-level latency goals.

Here let us work through specific numbers; generalization is not too hard. The requirements are 100,000 transactions per second, maximum response time 1s. Further, assume that the disk may occupy a maximum of 0.5s of that response time. Assume that for 1Gbyte of data a disk with latency 7ms and transfer rate 40Mbyte/s is available, and a typical transaction reads 128 bytes (writes are a complication dealt with later), an amount small enough to suggest worrying about latency rather than fast streaming. The basic requirement is clearly out of reach. Although a response time of 0.5s (disk time only) is much larger than 7ms, the requirement of 100,000 transactions per second requires that each transaction take at most 10μs, almost 1000 times faster than the disk's access time. But what of the disk's 40Mbyte/s transfer rate? That looks more promising, as that allows 128 bytes to transfer in 3μs. The problem is that we cannot stream the data continuously since we require random accesses … but let's carry on with this line of thought.

What happens if we stream the disk continuously? Assuming this can be done with no pauses, the time to sweep the entire data set is 25.6s, still not so promising, since we want an operation to take at most 0.5s. What we have is 50 times too slow, in terms of user-level response time. Next step: replicate the data on 50 disks, synchronized so the data is timed to be equal distances apart as the disks stream, which should be possible, given that synchronizing disks is a solved problem for RAID. Now, any one item to be read is at most 0.5s away at any given time. The next trick is to queue requests (assuming the time for this is trivial compared with 0.5s) in a tagged buffer, which can detect when a given request matches the address of data being streamed off the disk (similar to cache tags). If the buffer can hold 50,000 requests, then up to 100,000 can be dispatched per second, assuming the worst case, that every request waits the maximum delay.

Writes present a more difficult problem. They could be buffered, and to avoid losing bandwidth or latency to them, ideally the drives should have a separate set of write heads. Obviously any reads that refer to buffered writes should pick up the buffered copy. Assuming the write problem can be handled, if a compute server can keep up with the required rate of transaction handling and the network interface is up to it (someone else's problem, only the disk subsystem is considered here), this design meets the stated requirements.

The competition is RAID [Chen *et al.* 1994], RAIS (redundant array of inexpensive servers), RAM-based databases and large-scale or mainframe-style disk systems. RAID cannot deliver the required latency. RAIS is used for large web sites; in principle, a RAIS system (or other distributed design) could better this design, but presents hard design problems like maintaining data consistency (in the presence of writes) and partitioning workload. A RAM-based solution gives better latency more easily and could beat the peak bandwidth of this solution, but doesn't match its potential fault tolerance or its nonvolatility across power failures. A high-end or mainframe-style disk subsystem is expensive and even so would have difficulty in achieving the design goals of the example presented here (under the assumptions here, 50 disks with 7ms access time couldn't support more than about 3,500 transactions per second if accessed the standard way).

The case made here is for disk designers for large-scale transaction-based systems to abandon the futile pursuit of lower latency and focus on maximizing bandwidth (e.g. by higher numbers of heads, and a faster interconnect to each disk). Large-scale web sites, in particular, with many fewer writes than reads, could benefit from this disk architecture.

Is the idea so new? Not really, a memory that continuously streams sequentially is one of the oldest ideas in digital computers: some early computers used an acoustic mercury delay line which had very similar properties [Bowden 1953].

3.0 A Scalable Video on Demand Architecture

Another classic example of the latency-scaling problem is video on demand (VoD). Acceptable latencies for actions like fast forward or changing to a new movie may run to many seconds, even a minute or more (given how slow the competition, a VCR, is), yet achieving these latencies with a very high number of users is problematic, as witnessed by the complexity of proposed architectures for large-scale VoD systems [Taylor *et al.* 1995, Jadav and Choudhary 1995, Dan *et al.* 1994].

The approach proposed here again is to blow away large amounts of bandwidth, generally easier to scale up than latency, to achieve the desired response times.

The proposed approach is to stream each movie at 1 minute intervals, which for a typical 2-hour movie requires 120 times the bandwidth of a single stream. At the user end, a relatively simple set-top box sends signals to an interface to the high-speed interconnect in a local station remote from the user's residence to perform actions like fast forward (skipping to another version of the same movie which is ahead in time) or to another movie. The worst-case latency for simpler operations is 1 minute, and can be improved and extended to other cases like rewind by caching the movie on local disk.

A high-capacity interconnect is required to carry the full capacity of the overall service. However, a local station containing interfaces to the high-speed interconnect can tune to a desired channel, so a relatively low-speed link is needed to each user (sufficient bandwidth for one movie plus control signals). While the amount of bandwidth in total is high (for 1,000 simultaneously available HDTV-standard movies, 2Tbit/s), the bandwidth is shared between many users. If a budget of 32Kbit/s is allowed per user, 69,000 new users have to be added to justify adding a single new HDTV movie. For broadcast-quality service, only 23,000 new users would be needed to justify adding one more movie to the service with this bandwidth budget.

Contrast requirements with a full video on demand system where any user can be at any stage of viewing any movie: 120 times the bandwidth of all available movies is a relatively modest total requirement.

To phase such a system in, a lower-specification service can be offered at first (e.g., MPEG-2 with a modest number of movies). The attractive feature of the design is that the cost per user goes down as users are added, making it possible to increase the number of movies or video specification, or both, as the user base increases. Since there is no traffic back to the server, server requirements are purely a property of the number and quality (in terms of video spec) of movies on offer. This contrasts with other VoD proposals, where an increase in the number of users requires either or both of an increase in the complexity of the server or interconnect.

4.0 Conclusions

The ideas presented here suggest a paradigm shift in disk and network strategies which attempts to build on the inherently greater ease of scaling up bandwidth, as opposed to latency. The fact that many latency requirements at the user level translate to times that are extremely long by computer standards suggest that we are doing something radically wrong if latency is such a hard problem in computer system design.

While the streaming-oriented approaches here might at first sight appear to be very wasteful in terms of bandwidth, in reality they make much more effective utilization of bandwidth than systems in which the available bandwidth is largely wasted waiting for other parts of the system to be ready, or waiting to set up a new transaction.

The big challenge with gaining wider acceptance for these ideas is to show that they apply more widely than these examples, as significant resources would need to be put into engineering disks and networks to achieve the required bandwidth. Certainly, for a real large-scale database, for example, disk manufacturers would have to be persuaded to ship units with much higher bandwidth than the example used in this paper. The example works because the database only contains 1Gbyte of data; a database containing tens of Gbytes to Tbytes of data would require much higher bandwidth to achieve the given latency goals.

However, the required bandwidths are inherently less of an engineering problem than the latencies required of traditional solutions, which suggests that it will be worth taking this idea further. Potential future work includes more detailed evaluation of the given examples, extending the principle to other examples, examining how deeply the basic architecture can fit to overall system design, investigating the potential for adding fault tolerance to the disk delay line and exploring implementation of real-time systems based on some of the ideas presented in this paper.

Acknowledgments. This work was done while on sabbatical hosted by Trevor Mudge at Department of Electrical Engineering and Computer Science, University of Michigan. Financial support was provided by the University of the Witwatersrand and the South African Foundation for Research Development.

References.

[Bowden 1953] B. Bowden (ed.). *Faster than Thought*, Pitman, London, 1953.

[Chen *et al.* 1994] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz and D. A. Patterson. RAID: High-Performance, Reliable Secondary Storage, *ACM Computing Surveys*, vol. 26 no. 2 June 1994, pp 145-185.

[Dan *et al.* 1994] A. Dan, D. Sitaram and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching, *Proc. 2nd ACM Int. Conf. on Multimedia*, San Francisco, CA, October, 1994, pp. 15-23.

[Jadav and Choudhary 1995] D. Jadav and A. Choudhary. Designing and Implementing High-Performance Media-on-Demand Servers, IEEE Parallel & Distributed Technology, vol. 3 no. 2, Summer 1995, pp. 29-39.

[Taylor *et al.* 1995] H. Taylor, D. Chin, S. Knight and J. Kaba. The Magic Video-on-Demand Server and Real-Time Simulation System, *IEEE Parallel & Distributed Technology*, vol. 3 no. 2, Summer 1995, pp. 40-51.