DISK DELAY LINES

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

Abstract

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. While a transaction-based system presents performance challenges other than disk latency, it is interesting to develop a model of disk architecture in which disk latency no longer presents a challenge, which allows system designers to focus on other areas in which performance goals may be hard to meet. The Disk Delay Line concept relies on the fact that a disk can stream data quickly. A single-Disk Delay Line is a disk which constantly streams its entire contents, and a request for data or for a write waits until the required portion of the data stream appears. A given latency goal can be achieved by replicating disks, with copies of streams evenly spaced apart in time, and a given number of transactions per second can be supported by sufficient memory to buffer requests.

1. Introduction

User-level goals for response times are slow by computer standards. However, scaling up presents a problem, because latencies over large numbers of competing requests are added. Transaction-based systems and web sites, for example, run into this problem: latencies for users in seconds may be acceptable (if sub-second response time is ideal).

This paper considers a novel approach to disk usage, which exploits the disk’s strong point, streaming, while avoiding adding latencies across individual transactions.

The general idea is that a disk should be streamed constantly, and requests be buffered and serviced as soon as the data they requested becomes available. Writes can also be handled, but require a little more complexity. The idea is expensive to implement for very low latencies, but scales well to very high transaction rates.

The model is called a Disk Delay Line (DDL); the explanation for this name is saved up as a surprise in the Conclusion.

The remainder of this paper is organized as follows. Section 2 presents an example to illustrate how a DDL can be designed. Section 3 briefly summarizes problems not addressed in the design, and Section 4 describes competing technologies. The final section concludes with a recommendation for disk designers, and explains the origin of the DDL name.

2. Disk Delay Line Design

Here let us work through specific numbers; generalization is not hard. The requirements are 100,000 transactions per second, maximum response time 1s. Further, assume that the disk may occupy at most 0.5s of the 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 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 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—this is 50 times too slow. 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 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 are more difficult. They could be buffered, and to avoid losing bandwidth or latency, drives should have separate write heads. Obviously any reads that refer to buffered writes should pick up the buffered copy. Assuming that 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.

3. Other Challenges

While it appears to be reasonably simple to build a disk system as described in the previous section, building an overall system that could keep up with a transaction rate like 100,000 TPS would be a major challenge. Certainly, no system shipping today claims any number close to this: the best figure reported by May 1998 for TPC-C, an online transaction-processing benchmark, was 102541 transactions per minute, or 1709 TPS, on an Alpha with 96 processors, costing over $14-million [TPC 1998].

Part of the problem at least that designers run into in high-end TPS systems is working around the fact that disk latency is poor: a lot of effort has to be put into partitioning databases, balancing load between parts of multiprocessor and distributed systems, and working around the resulting irregular memory reference patterns that are a poor fit to a conventional memory hierarchy.

Perhaps a DDL-based system would avoid some of these problems; a more detailed design of an overall system is required to assess whether a DDL architecture solves other TPS design problems.

4. Competition

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).

5. Conclusion

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, with many fewer writes than reads, could benefit from this disk architecture. Transaction-based systems which typically reference small amounts of data per transaction, such as online transaction processing (OLTP) systems could also benefit.

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

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.

[TPC 1998] *Top Ten TPC-C Results by Performance*, Transaction Processing Performance Council <http://www.tpc.org/new_result/ttperf.idc>, May 1998 (last update at time of writing).