Video-on-Demand on the SB–PRAM[*]

Jörg Friedrich Thomas Grün Jörg Keller[†]

Universität des Saarlandes
Computer Science Department
Postfach 151150 – B. 36 R. 108
66041 Saarbrücken, Germany
Email: {jf, gruen, jkeller}@cs.uni–sb.de

Abstract

We investigate the use of random striping to store movies on the SB–PRAM, a massively parallel multiprocessor. Our simulations show that the SB–PRAM provides enough bandwidth between disks and processors to serve more than a thousand users in a Video-on-Demand environment. Furthermore we present a novel method to use a parity scheme in order to decrease disk load in addition to provide fault tolerance.

1 Introduction

For Video–on–Demand servers, the organization of disk storage is one of the most important problems. The most popular schemes to store movies on a set of hard disks are *rate staggering* [5] and *random striping* [6]. Rate staggering allows for a very regular distribution of blocks onto disks, but restricts bit rates to few values. Random striping distributes a movie over n disks by randomly permuting n consecutive blocks over all disks. Random Striping thus aims at obtaining high and equal disk utilization, while the bit rates of different movies can be chosen independently.

To serve many users with a Video–on–Demand server, one might wish to use a multiprocessor instead of a mainframe type server. Many massively parallel machines are distributed memory multicomputers, e.g., Intel Paragon XP/S or nCUBE2 [2]. Unfortunately, random striping does not work well on distributed memory architectures. It does not show locality in disk referencing, thus requiring lots of communications between processors. However, message passing for this type of communication results in a large overhead.

Conventional shared memory architectures are either bus-based and thus restricted to few processors like the Sequent Symmetry [2] or are cache–based like the KSR1/2 [13] or the Stanford DASH [10]. The caches can only be exploited if there is a significant amount of locality in data referencing. However, this does not hold for random striping. Hence, random striping is not very useful on the majority of todays multiprocessors.

A multiprocessor which is suitable to implement random striping is the SB–PRAM [1], currently under construction at the Universität des Saarlandes. We will present experiments that support this claim. A machine that could provide similar features is the Tera computer [3], now marketed as Tera MTA.

The SB–PRAM also provides support to implement parity schemes, e.g. RAID-3/5 [11]. Besides their use to increase reliability, they are also used to decrease disk load. If a disk is overloaded, its data can be reconstructed from all other data and the parities.

The reminder of the article is organized as follows. In Section 2, we briefly describe the SB–PRAM architecture and its use as a Video–on–Demand server. In Section 3, we investigate the performance of random striping on the SB–PRAM. In Section 4, we describe and analyze the use of parity schemes to decrease disk load. Section 5 concludes and discusses future research.

2 SB–PRAM

The SB–PRAM [1] is a multiprocessor with 128 processors and 128 memory modules. Processors send their requests to access a memory location via a butterfly network to the appropriate memory module. An

[*]Research partly funded by the German Science Foundation (DFG) through SFB 124, TP D4.
[†]Supported by a DFG Habilitation Fellowship.

answer to a load request is sent back via a second butterfly network. There are no caches. The address space is randomly hashed among the memory modules, thus avoiding hot spots [8]. The bandwidth between processors and the network is large enough that each processor can issue a memory request in every instruction.

Because of the random hashing, the network latency is uniform, but it is large. It is hidden by having each physical processor run multiple threads. By the use of delayed load, the necessary number of threads per processor can be limited to 32. Each thread has its own register set in hardware, the threads are scheduled round-robin after every instruction.

Thus, the user sees a machine with $128 \cdot 32 = 4096$ *virtual* processors and access to global memory within one instruction. The global memory has a size of 8 GByte. The network supports concurrent access of multiple processors to a memory cell by combining requests. The net bandwidth to the network is 32 MByte/s for each processor and each memory module.

Each processor has two SCSI ports that together have a bandwidth of 10 MByte/s. Further, it has a local memory of 256 KByte that is utilized as a buffer to the disks and other peripherals. In the final version, each processor board will also have a standard PCI interface to which an ATM interface (OC–3) will be attached. Currently, a four processor prototype without PCI bus is being tested.

A movie is striped over all disks. Each physical processor manages a certain number of video streams. To deliver a video stream, a set of virtual processors request blocks from the disks. The requests are written into FIFO queues held in the global memory. There is one FIFO queue for every disk. Data structures that handle concurrent access of multiple processors to one queue without serialization are available [12, 14]. On each physical processor, a distinguished set of virtual processors read the queue belonging to their disk. They process these requests and transfer the blocks read to the global memory. The virtual processors that requested the block then transfer it to their ATM–link.

The global memory can be used as a software cache. Thus blocks requested by one stream can be delivered to other streams without accessing the disks. An appropriate caching scheme is interval caching [6].

3 Random striping

We simulate random striping for $d = 256$ disks and $u = 1280$ users. For the simulation, we make the following, simplifying assumptions:

- Users who start watching the same movie within a one-minute interval are grouped together. So, they only occupy one video stream. Techniques like interval caching [6], batching [7] or adaptive piggybacking [9] can achieve this.
- In our current setting, we do not consider interaction like pause, fast forward or rewind. However, our approach should be able to handle these features.
- All disks have a fixed response time. We use this restriction for practical reasons, i.e., we can write down a quick and dirty simulation program and do not need to model a specific disk for use with an discrete event simulation program. Besides, rate staggering needs this assumption too.

We suppose a user to access a disk roughly four times[1] per second. This results in an average request rate of $r = \frac{4 \cdot u}{d} = 20$ accesses per second, requiring a disk service rate of $s > r$. We define s_α as the disk service rate which results in an average disk utilisation of α percent.

Our simulation program proceeds in rounds of one second length. First, it issues four requests per user, filling up the disk request queues. Then, it writes out the queue statistics, processes s disk accesses for each disk and moves on to the next time step.

For the first experiment, we assume a disk utilization of 70 percent, i.e., a disk must be able to satisfy $s_{70} = 28$ requests per second. The result of that simulation run is visualized in Figure 1. As expected, the average queue length is about 20; actually, it is slightly higher than 20, because of previously issued requests that could not be satisfied in the round they were produced.

Although the average queue length is less than the disk service rate, the maximum queue length over all disks is by far higher than s_{70}. It indicates how much data must be prefetched in order to guarantee a continuous playout. With the above parameters it is sufficient to request a block two seconds before it will be played, since the maximum queue length is always less than $2 \cdot s_{70} = 56$.

[1]Normally, one would access a disk less frequently and read larger data blocks in order to get a higher disk throughput. We use this somewhat unrealistic value to get simpler mathematics.

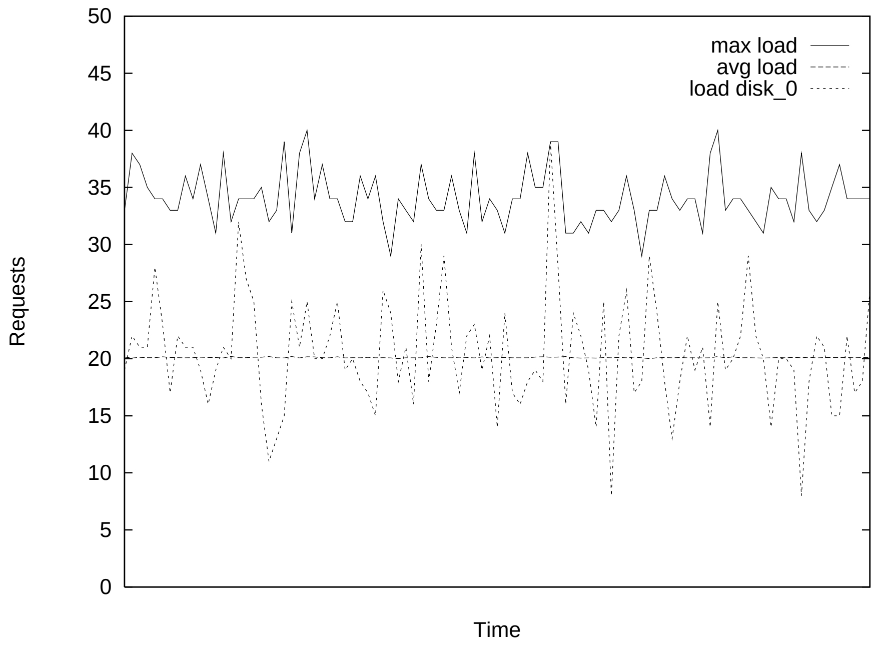

Figure 1: Random Striping ($r = 20$, $s_{70} = 28$)

The diagram illustrates the queue statistics for an interval of 100 seconds taken from the middle of a simulation run. The dotted line represents the variation of queue length for one particular disk, the solid line is the maximum queue length over all $d = 256$ disks.

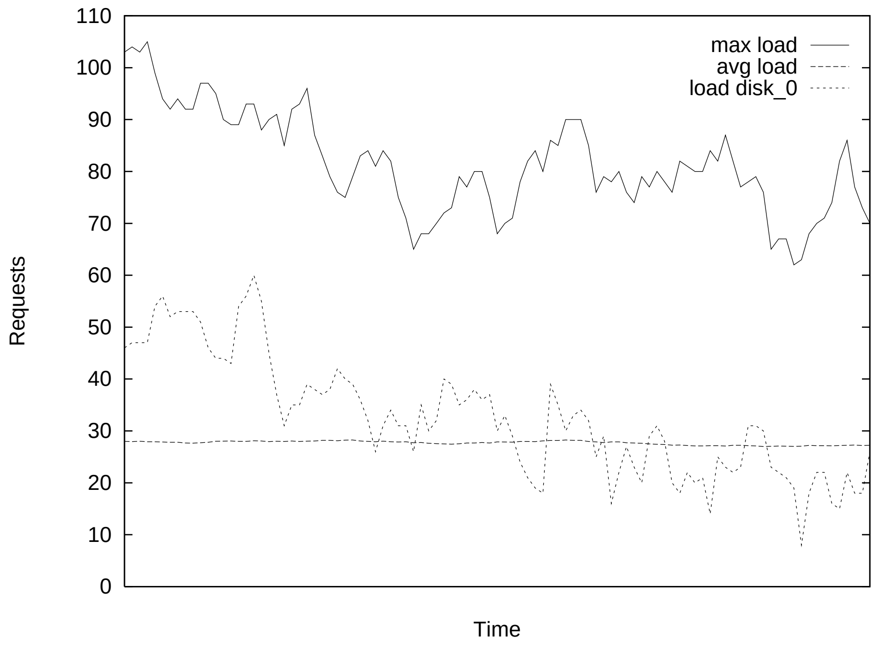

Figure 2: Random Striping ($r = 20$, $s_{95} = 21$)

Things change dramatically if disk utilization is increased to 95 percent, i.e., the disk service rate is set to $s_{95} = 21$ (see Figure 2). Then, the average queue length increases to 28, which means that about seven old requests cannot be satisfied in the time interval they are produced. The maximum queue length is even greater than one hundred, leading to a prefetch time of more than 5 seconds.

These figures suggest that random striping is not suited for distributed memory computers for the following reasons: Either *all* processing nodes must have a large memory sufficient for holding the number of disk blocks related to the maximum queue size, or they must temporarily store the data at their neighbor nodes.

Although a shared memory computer is not concerned with data distribution and the communication overhead involved with it, it suffers from a unequal load distribution on its disks too. In the next section we show how to reduce the prefetch size considerably.

4 Parity schemes

RAID [11] is a popular method for preventing data loss due to disk failures. For Video-on-Demand servers, the use of "hot standby" disks or RAID-5 disk subsystems has been proposed in [4, 5]. We now describe how to employ a parity scheme in order to equalize disk load and, thus, to reduce prefetch time.

As in the previous section, movies are distributed randomly to the disks. Additionally, one data block for parity is calculated over p data blocks as sketched in Figure 3. We denote those $p+1$ blocks (p data + 1 parity) as a *parity group*. With RAID, the parity block is only used when a data disk fails. In our approach, we choose the p disks with the smallest load from a parity group and reconstruct the data of the remaining block if necessary. In other words, we do not further schedule requests to disks with a high load.

The presented method implies that all data from p disks of a parity group must be read prior to further being processed, a possible drawback of this method. For reducing memory consumption on the set-top-box side, we assume that the data from p successive original data blocks are distributed round-robin over p disks (see Figure 4).

The corresponding parity block is stored on the $p+1$-th disk of the parity group. For transmission over the ATM network, the data words are taken round-robin from the p chosen blocks, i.e., we first read word 0 from the p chosen blocks, then word 1, etc. This reordering does not cost any extra instructions on the

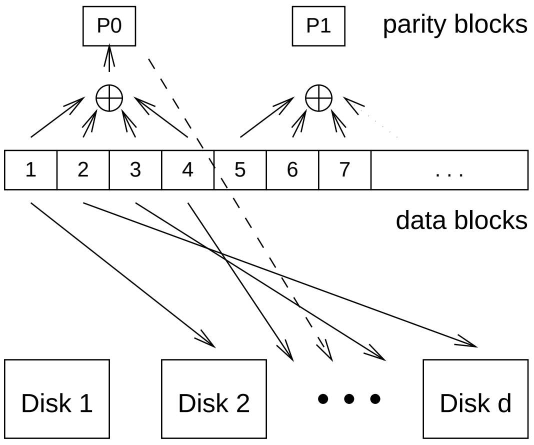

Figure 3: Parity Scheme

A parity block (P0, ...) is calculated from $p = 4$ data blocks of a movie. Data and parity blocks are spread randomly over all disks.

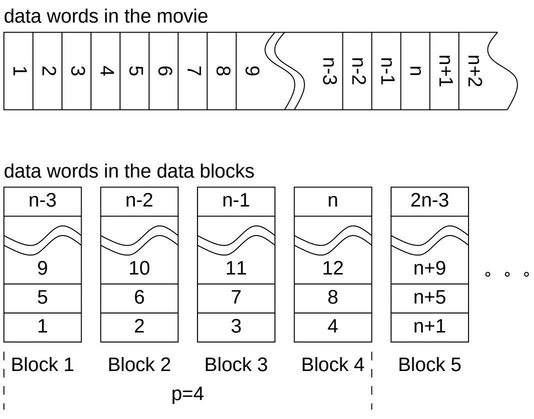

Figure 4: Word numbering

The movie's data is partitioned into chunks of $n = p{\cdot}l$ data words. Each chunk is distributed round-robin to p data blocks of lenght l.

SB-PRAM, because the disk blocks are first copied to the global memory and thus, the transmitted data words have to be copied to the local memory of the transmitting processor anyway.

The reconstruction process involves EXORing p successive data words in order to get the missing data word. However, this does not consume any server CPU time. Either it can be carried out by the set-top-box in software, if the processor of the set-top-box is powerful enough. Otherwise it can be done by extra hardware on the data path between the local memory of a server processor and the local memory of the set-top-box. This hardware can be incorporated into the DMA logic of the ATM sender or receiver.

For $p = 4$ and a 95 percent disk utilization (see Figure 5), the average queue length with this new approach is about 20 and the maximum queue length is less than 25. So, a prefetch time of two seconds, which is equivalent to 42 requests, is more than sufficient to assure an uninterrupted display of movies.

The dotted curve, showing the request queue length for one particular disk, shows a characteristic behavior. If the load is lower than the average load, large deviations from the average can occur. Contrarily, if the load is higher than the average, the deviation is much smaller, the peaks seem to be cut off. This happens because the most loaded disk of a parity group is never accessed.

4.1 Disk failure

A disk failure is detected when a disk does not answer a request within a given time interval. If this happens, all requests must be rescheduled, which possibly causes interruptions in the movie display, and the disk must be marked as defective. No further requests will be sent to that disk, so it can be safely replaced. If a new disk is inserted, the data for it can be reconstructed using the other disks.

During the repair phase all user requests will be supported by the $d-1$ remaining disks. If the defective disk is contained in a parity group, the p intact disks have to be chosen. Figure 6 depicts the simulation results for this scenario. It can be seen that the load is only marginally higher than in the simulation with all disks working. The system can support all users even if one disk fails. If a second disk fails, the system will stop to work as soon as the two defective disks are in one parity group of a movie. Two-dimensional parity schemes can handle a two disk error without interruption of movie display.

5 Conclusion and Future Research

We have shown that it is possible to use random striping economically with Video-on-Demand servers. If the new parity scheme is employed, a disk utilization of nearly 100 percent can be achieved with moderate prefetching sizes. Although the proposed scheme can be utilized in mainframe servers, the realization on parallel shared memory computers like the SB–PRAM is better if scalability is required.

We believe that random striping with parity is well suited for movies recorded with variable bit rate coding like MPEG-II or for a collection of movies that are coded with constant but different bit rates. The parity scheme or a partial replication can be chosen independently for each movie, giving the system the chance to make use of its resources in the most efficient way.

The prefetch size of two seconds described in this paper is only an upper bound for the buffer memory needed. We will carry out a more detailed simulation, where a real disk will be modeled more accurately and a global disk scheduler will be employed to trigger disk accesses. This scheduler determines in what order disk accesses are to be performed and which buffers will be assigned to them. In doing so, the buffer pool can be global, a fact that possibly reduces the overall buffer space.

Another research direction is the online reorganization of the Video-on-Demand server: New movies shall be stored onto disk while the system is still serving a few users. Changes in user behavior must be taken into account by partially replicating a movie or by changing the parity scheme used for it.

References

[1] Ferri Abolhassan, Reinhard Drefenstedt, Jörg Keller, Wolfgang J. Paul, and Dieter Scheerer. On the physical design of PRAMs. *Computer Journal*, 36(8):756–762, December 1993.

[2] George Almasi and Allan Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings, 2nd edition, 1994.

[3] Robert Alverson, David Callahan, Daniel Cummings, Brian Koblenz, Allan Porterfield, and Burton Smith. The Tera computer system. In *Proc. 1990 Internat. Conf. on Supercomputing*, pages 1–6. ACM, 1990.

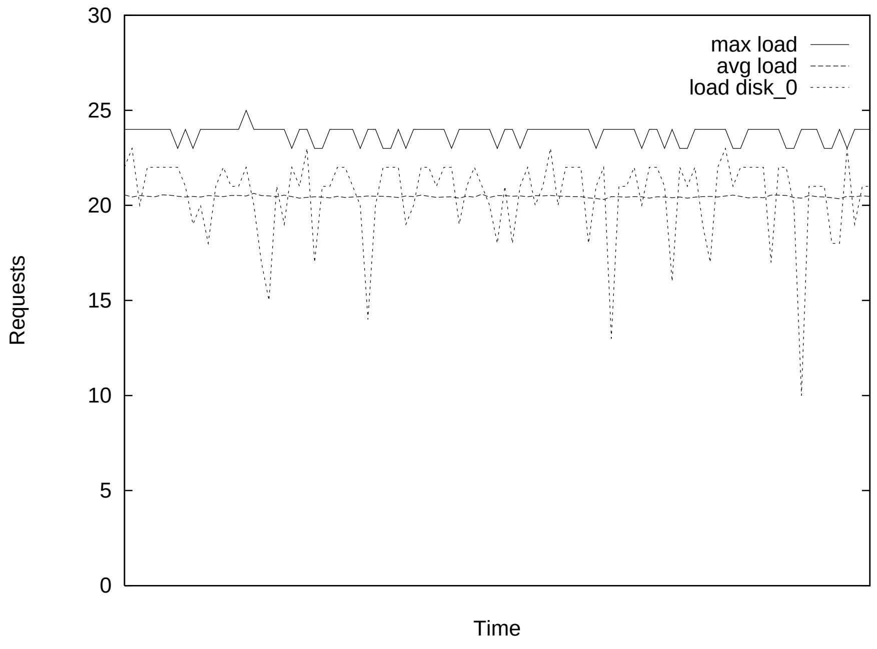

Figure 5: Parity scheme ($r = 20$, $s_{95} = 21$)

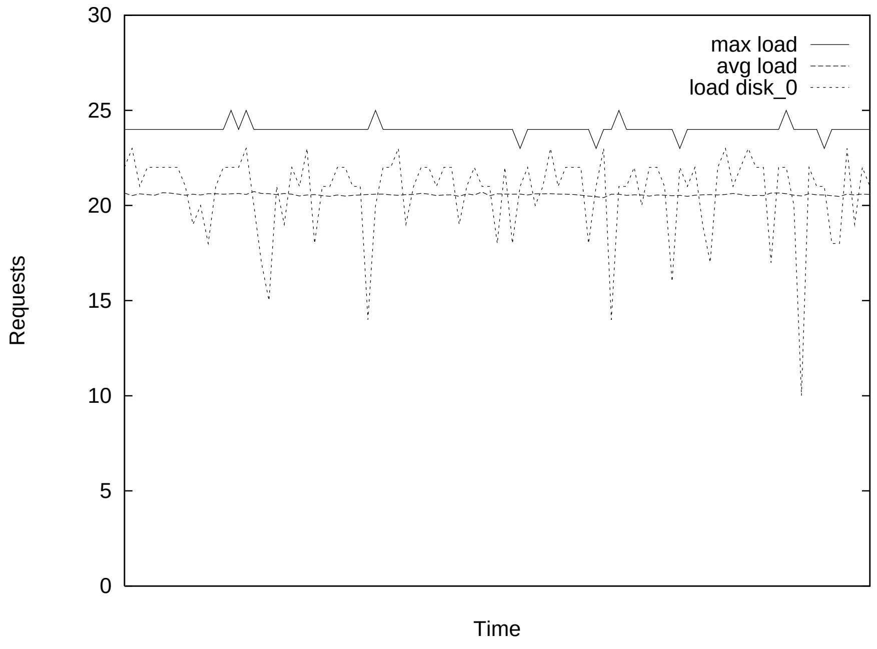

Figure 6: Parity scheme ($r = 20$, $s_{95} = 21$, 1 disk failed)

[4] Christine Blank. The FSN challenge: Large-scale interactive television. *IEEE Computer*, 28(5):9–12, May 1995.

[5] Ming-Syan Chen, Dilip D. Kandlur, and Philip S. Yu. Storage and retrieval methods to support fully interactive playout in a disk–array–based video server. *Multimedia Systems*, 3(3):126–135, 1995.

[6] Asit Dan, Daniel M. Dias, Rajat Mukherjee, Dinkar Sitaram, and Renu Tewari. Buffering and caching in large-scale video servers. In *Proceedings of the CompCon '95*, 1995.

[7] Asit Dan, Dinkar Sitaram, and P. Shahabuddin. Scheduling policies for an On-Demand video server with batching. In *Proc. 2nd Annual ACM Multimedia Conference and Exhibition*, 1994.

[8] Curd Engelmann and Jörg Keller. Simulation-based comparison of hash functions for emulated shared memory. In *Proc. PARLE '93, Parallel Architectures and Languages Europe*, Lecture Notes in Comput. Sci. 694, pages 1–11. Springer, June 1993.

[9] Leana Golubchik, John C. S. Lui, and Richard Muntz. Reducing I/O demand in Video-on-Demand storage servers. In *Proc. Joint International Conference on Measurement & Modeling of Computer Systems Sigmetrics '95/Performance '95*, pages 25–36, 1995.

[10] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Wolf-Dietrich Weber, Anoop Gupta, John Hennessy, Mark Horowitz, and Monica S. Lam. The Stanford DASH multiprocessor. *Comput.*, 25(3):63–79, March 1992.

[11] David A. Patterson, G. Gibson, and Randy Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the ACM International Conference on Management of Data*, pages 109–116, 1988.

[12] Jochen Röhrig. Implementierung der P4-Laufzeitbibliothek auf der SB-PRAM. Diplomarbeit, Universität des Saarlandes, FB Informatik, 1996.

[13] J. Rothnie. Kendall Square Research introduction to the KSR1. In D.B. Johnson, F. Makedon, and P. Metaxas, editors, *Proceedings of the Dartmouth Institute for Advanced Graduate Study in Parallel Computation Symposium*, pages 200–210, June 1992.

[14] James M. Wilson. *Operating System Data Structures for Shared-Memory MIMD Machines with Fetch-and-Add.* PhD thesis, Dept. of Computer Science, New York University, June 1988.