Palo Alto Research Center

Low Latency Logging

Robert B. Hagmann

XEROX

Low Latency Logging

Robert B. Hagmann
Xerox Palo Alto Research Center

CSL – 91 – 1 February 1991 [P91 – 00031]

Abstract: Many database and file systems use a disk log for fast crash recovery. Some of the latency in transaction commit is due to the rotational latency of the disk. This paper shows a design that gives excellent average latency for logging.

CR Categories and Subject Descriptors: H.2.2 [Database Management]: Physical Design — Recovery and restart; H.2.7 [Database Management]: Database Administration — Logging and recovery;

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

1. Introduction

Commit latency can be a performance problem in some database and file systems. Some of this latency in transaction commit is due to the rotational latency of the disk. This paper shows a design that gives excellent average latency while doing log based recovery. Sophisticated readers may wish to skip to sections 2.2 and 2.3 to see the trick.

Many systems require that changes be done atomically to data kept on a disk. One of the common ways to implement atomicity and recovery is by using logging. Briefly, this means that changes made to the database are noted in log records which are written to disk before a transaction can commit. These records are read during recovery to restore the state of the disk. The log is an append – only file. The trick in this paper is how to append quickly. Logging is used in almost all database systems and in some file systems.

One type of logging is "write – ahead logging." Here the new (and sometimes old) values for part of the disk are written to a log. The system maintains the invariant that log records are always written before changed sectors are written to disk.

Conventional wisdom is that logging greatly increases the performance of databases. Since only a single write (to the log) is needed to logically update many physically separate places on the disk, a simple transaction can be committed using a single write. In high throughput systems, multiple transactions are committed by doing a single write (this is part of what is called "group commit" [Gawl85a] [Hell88]). The modified data pages of the database can be held in memory. If they are again modified (a "hot spot"), then only a single write is needed to update the data page. When data pages are written, they can be done in an efficient order (e. g., by using the elevator algorithm).

Logging is usually done to a disk. The disk write to do the logging can be a major part of the latency of doing simple update operations.

For example, let's consider implementing the NFS [SUN86] "write" RPC call using log based recovery. To do the log write, using typical disk specifications, might take:

1 msec to set up I/O and later take an interrupt

15 msec seek time (move the disk arm to the proper cylinder)

8.3 msec rotational latency (wait for the disk to turn)

1 msec transfer time (actually write the data onto the disk)

Higher and lower performing disks are available, but these are values that are typical of high quality, currently available disks. Systems will differ in their interrupt times and I/O setup, but it should be clear that the dominant times are for seek and

rotational latency.

So, there are two large components to doing the log write. The seek time and the rotational latency both require physical motion, and hence are slow as compared with electronic operation. The seek moves the disk head to the proper cylinder. The rotational latency is time spent waiting for the disk to turn so that the desired sector is under the disk head. Average rotational latency is half a revolution, so for disks turning at 3600 RPM that is 8.3 msec.

For medium to large systems, logging is done on a dedicated device. Hence, the disk arm is correctly positioned (or nearly so) and no seek (or a one cylinder seek) is usually all that is required. In such a system, the dominant cost (in time) of doing logging is the rotational latency. This paper presents a way to decrease this cost.

Only some systems are concerned with latency. High throughput systems often trade a small amount of additional latency for higher throughput. Sometimes other operations swamp the time to do the log write, so low latency is not important (e. g., query optimization). Some high throughput systems have a bottleneck in the bandwidth of the log device. The technique in this paper can be used for those systems to increase the effective bandwidth (by increasing the fraction of time transfer actually occurs by decreasing the latency of the start of a write).

However, in some systems it is important to do fast commit. The total time for the NFS write call mentioned above is usually dominated by the disk I/O time. Naive NFS implementations do two, three, or even more write I/O's per "write" call. The network transfer time and RPC overhead are dominated by the disk I/O time.

High performance systems sometimes use stable memory. Typically, this is battery backed up RAM or a RAM Disk. The stable memory serves as either a write buffer for the system (for NFS see Lyon and Sandberg [Lyon89]) or as a write buffer for the log tail. In both cases, stable memory decreases the commit latency by doing memory operations instead of disk I/O's. However, only some computers support stable memory. The way it is integrated into the system varies. There is no common implementation of stable memory that database and file system software vendors can expect. If the software is to be portable, it has to deal with computers that do not have stable memory.

This design is not part of any file system or database. Hence, this paper cannot report on the operational performance of the technique proposed here.

2. Design Overview

2.1 Normal system operation for a typical system

A system using logging uses a (logically) append only log. New sectors are added to the end of the log to indicate changes in the state of the system. Items are added to the end of log buffer in memory either with or without force. Force means that the call is blocked until the disk write for the log with the item returns (and all previous log writes have returned too). Without force means that the item is just buffered and the call is returned from immediately. Transaction commit records are done with force.

Write ahead logging causes logical updates to the disk to be made first in the log. Once the log write has occurred, the system may then write the data pages in the database or file system. The system is not required to write the pages immediately. Depending on the checkpoint and recovery invariants of the system, these writes may be substantially delayed. Gray and Putzolu [Gray87] estimated in 1987 that the delay should be 5 minutes for a database. By delaying the write, further updates to the same pages avoid the initial write. Various optimizations can be performed, such as using the elevator algorithm, to minimize seek and latency. Particularly in the case of UNIX file systems, many files have a very short lifetime [Oust85]. The file bytes for these files never have to be written to disk, except initially to the log.

Some logging designs write a header on every log page to identify it. This is commonly done in database systems since modern disk controllers with track buffers will write sectors out of order and recovery has to have some means to detect these partial writes. Other designs only write headers on the first page of a contiguous write. The log is usually initialized to a known state (e. g., filled with "noop" records).

2.2 Changes to normal operations (this section and the next has the trick in it)

Typical systems always append to the end of the log. This is done to make the log linear. The physical position of the records in the log corresponds to the time order that the log records were created. This paper proposes a way to properly order the log records without using physical position.

First of all, the system has to know (approximately) the rotational position of the disk for the log and the physical layout of the sectors. This information may be available from the disk controller. Otherwise, the rotational position can be approximated by knowing or computing the disk's rotational speed (assuming constant angular velocity, CAV, disks) and when the done interrupt occurs for disk

read/write operations. Other systems have used this approximation in the past. The physical layout of the sectors may require some (substantial!) reverse engineering of the disk controller and disk drive. Bad sector forwarding and bad track forwarding may also complicate this. For this paper, we will assume that the physical layout of the sectors is easy to compute and not worry about bad sectors and tracks.

This design requires that a header is on every log page for identification. The header contains a new field, the page sequence number, that is incremented for each logical page written to the log. (If the log records already have log sequence numbers, they can be used instead.)

When starting to write on a new cylinder in the log, no pages of the log have yet been written on any disk head. When a request for a log write is issued, the system examines the size of the write, the rotational position of the disk head, and a map of which pages have been written. It finds a (set of) run(s) of pages that minimizes latency using first fit. For the first log write on a new cylinder, that would be the next sector to come under the disk head for disk head 0. This computation has to account for I/O startup time.

As more log writes occur for this cylinder, more pages are written. The system selects the lowest latency set of sectors, possibly using multiple heads that can do the write (e. g., write using head 1 for awhile then switch to head 3). Once the cylinder reaches some threshold of occupancy (e. g., 90%), or reaches some minimum latency guarantee, then the next cylinder is started. Pages on this cylinder that have not yet been written are not updated.

2.3 Changes to log reading

Logs are read in one of two ways. First, during normal operations or during recovery, the log is read by backchaining to abort a particular transaction or set of transactions. Second, during recovery the log is read linearly in ascending or descending order.

If backchaining of log records is needed, then both the log sequence number (LSN) and some way to find the sector address for the record are needed. For log records in the same log write, an offset physical disk sector can be used, while for log records in previous writes, a physical disk sector can be used. Backchaining can then read the proper sector and find the record of interest via the LSN.

The harder case is linear access during crash recovery. An entire cylinder is read into memory. Sectors with headers that are clearly old are discarded. These are easy to

determine since all sectors have page sequence numbers and the next page sequence number is known (either from a checkpoint or from processing the preceding or succeeding cylinder). All of the rest of the sectors are of interest. They are sorted by page sequence number and then processed in page sequence number order.

Hence, the log can be read either by backchaining or by sequential scan.

3. Comparison with Other Work

3.1 IMS WADS

After this paper was written, Jim Gray pointed out that this technique is similar to the way IBM's IMS can log its data. This technique is covered by a U.S. patent [Gawl85b]. In IMS it called the Write Ahead Data Set or WADS. Since WADS does not seem to be available in the open literature, a brief description of the relevant parts of WADS follows.

Many IBM disks have (or had) a fixed head surface as well as having several moving head surfaces. The disks are "count – key – data" architecture disks. I/O is done by searching sectors until one with a matching key is found, then the I/O operation (read or write) is applied to the sector.

Each track in the fixed head surface is initialized with fixed size sectors with every sector having the same key. A write to a fixed head track with the proper key will have very little latency: any sector will match the key. The latency is certainly less than a sector time, once the search for the key has started. Only one log data write is performed to a fixed head track, and then the next track is used. Once all tracks are used, all the log data written to all the fixed head sectors is rewritten to the log tail. With proper buffering, no reads are necessary (the log data is still buffered in memory).

When recovering from a crash, the fixed head disk surface must be examined to find records that were not copied to the log tail. Between writes to the log, a different fixed head track is used for every log write, so no log data is ever overwritten.

There are two major differences between this technique and the one proposed in this paper. First, all of the space on the cylinder is used, where in WADS only one write per track was done. Second, the WADS technique only works for count – key – data disks, while the proposal in this paper works with any fixed sector size architecture (e.g., IPI, SCSI, or SMD).

3.2 Ping – pong log writes to the log tail

Jim Gray also pointed out another unpublished trick for writing the end of a log for low volume systems with simplex disks. The log tail has a partially filled log sector already written on disk. We would like to rewrite this sector with a new sector with even more log data in it. The problem is that when writing to the log tail, one must not clobber the good data already written. If a partly filled log sector at the log tail is overwritten, a crash may leave the sector with a bad checksum. We have then lost the previous contents of the sector. For empty pages, simply write blindly. For non – empty pages, write to the next page. Continue to alternate between these two pages until the page fills up. Duplicate data can easily be eliminated during recovery.

4. Performance Considerations

In this section, a typical disk drive [NEC87] is assumed with the following characteristics:

3600 RPM

16.7 msec rotation

8.3 msec average latency

15 msec average seek time

2.4 megabytes/second transfer rate

Also, the system is assumed to add 1 msec of combined setup and interrupt time.

4.1 Performance during write for zero latency

This section estimates the gain possible if the latency can be made zero. Note that the technique proposed in this paper will not attain zero latency. This section is an estimate of the upper bound on the performance improvement.

4.1.1 Performance during write – dedicated log device

For an 8K log write, the transfer takes 3.4 msec. The total time is 3.4 + 1 + 8.3 = 12.7 msec. Eliminating the latency gives 4.4 msec. If latency could be set to zero, then the latency could be reduced by a factor of 2.9. The factor is about 6.8 for 1K log writes.

This analysis ignores the probability of a one cylinder seek before writing. For the NEC disk described below in section 4.6 and with 8K log writes, the probability of a seek is 1%. One cylinder seeks are about 5 msec, so the error in ignoring the seeks is

only 0.05 msec.

4.1.2 Performance during write – non – dedicated log device

For an 8K log write, the total time is 3.4 + 1 + 8.3 + 15 = 27.7 msec. Eliminating the latency gives 19.4 msec. If latency could be set to zero, then the latency could be reduced by 30%.

4.2 Sensitivity to errors in rotational position estimate

A systematic error in estimating the rotational position can greatly impact performance. If the estimate is wrong such that the write just misses the sector, then a whole rotation must be paid. In fact, the latency function is a saw tooth function of the error. The period is the time for a rotation. The maximum is the time for a rotation and the minimum is zero.

The selection of the sectors to write and the I/O is a real time operation. Non – predictability in the time to get an I/O started can cause high variance in the latency.

4.3 Performance during transaction abort

Log records are backchained by both the log sequence number (LSN) and the physical sector address. For log records in the same log write, the physical disk sector can be expressed as an offset. For log records in previous log write, the physical disk sector is known.

While reading the log for transaction abort, backchaining reads exactly the same number of sectors from approximately the same number of cylinders. As cylinders of the log are not completely filled, a few more or a few less cylinders will have to be accessed.

In any case, the performance is very close to the performance of a traditional system.

4.4 Performance during recovery

During recovery, the only difference from normal recovery is doing scans.

If the bottleneck for recovery is log reading, then additional buffering and a brief sort is needed before processing a cylinder of log data. If the log is only 90% full, reading takes a bit longer.

Usually the recovery bottleneck is the reads and writes to the database or file system. The number and timing of these are the same using the modified scheme or

the traditional scheme. Again, additional buffering and a brief sort is all that is added. Hence, there is little performance impact for recovery.

4.5 Extra disk storage

Once a cylinder in the log reaches a threshold, a new cylinder is started. Thus, more disk storage is needed for the log. Various techniques can be used to address this. First of all, the easiest solution is to simply buy a bit more disk storage. Disks are not that expensive.

Another way to address this is to compress the log during tape write [Kaun84]. Or the log can be compressed when changing extents. The log can also be staged from the non – linear form to a linear and compacted form.

4.6 Write times for a page of a large NFS file

The NFS protocol, Version 2 [SUN86], has a maximum 8K per write. The NFS specification states that the write is atomic. Hence, the write must be committed to disk. This means that for a traditional UNIX file system, the data page, the inode (contains the file byte size), and the indirect blocks (contains what pages belong to the file) all must be updated. This assumes a file big enough to need indirect blocks.

Timing estimates are given for two different networks. Table 1 is for Ethernet, a standard LAN that has a clock rate of 10 megabits per second. Table 2 is for FDDI, an impending standard LAN that has a clock rate of 100 megabits per second.

Performance is related to the transfer rate to stable memory or disk. One product stable memory is PrestoServe for SUN's made by Legato. While accurate measurements of its transfer rate are not available, it can be approximated from the data in paper about SPARCserver 490 tuning [Meht91]. Figure 2 shows that a server running 275 NFSops per second uses about 20% more of the CPU with PrestoServe than without PrestoServe. Scaling data from Table 2 that shows 34.6 disk writes/second at 521 NFSops/second, gives 18.3 writes per second. Assuming 8K pages, this is about 150KB/second. As this consumes about 20% of the CPU, the CPU saturates at about 750KB/second. In the following, the rate of 1 MB/second will be used to account for the errors in this estimate.

Timing estimates are broken down into the following categories. This assumes an otherwise idle client and server.

Overhead: 1msec per packet on either end, 1 msec per I/O

Net transfer: time to send 8K over the net

Seek: total seek time (12 msec per seek)

Latency: total disk latency; assume 1 msec for LLL

Disk transfer: total time actually transferring to disk

Total (% net busy): total time (percentage of net busy)

The designs considered, both for Ethernet and FDDI, are:

Naive: "synchronously force data, inode and indirect block pages"

Log: straight forward logging to a non – dedicated disk

LLL: low latency logging to a non – dedicated disk

Ded. log: straight forward logging to a dedicated disk

Ded. LLL: low latency logging to a dedicated disk

SRAM: logging to fast stable memory (10 MB/second bandwidth)

SBoard: logging to stable memory with a 1 MB/second bandwidth

Design	Overhead	Net xfer	Seek	Latency	Disk xfer	Total (% net busy)
Naive	5	6.6	36	25	3.1	75.7 (9%)
Log	3	6.6	12	8.3	2.9	32.8 (20%)
LLL	3	6.6	12	1	2.9	25.5 (25%)
Ded. log	3	6.6	0	8.3	2.9	20.8 (32%)
Ded. LLL	3	6.6	0	1	2.9	13.5 (49%)
SRAM	2	6.6	0	0	1	9.6 (69%)
SBoard	2	6.6	0	0	10	18.6 (35%)

Table 1: Write times for a page of a large NFS file using Ethernet

Design	Overhead	Net xfer	Seek	Latency	Disk xfer	Total (% net busy)
Naive	5	0.7	36	25	3.1	69.8 (1%)
Log	3	0.7	12	8.3	2.9	26.9 (2.6%)
LLL	3	0.7	12	1	2.9	19.6 (3.5%)
Ded. log	3	0.7	0	8.3	2.9	14.9 (4.7%)
Ded. LLL	3	0.7	0	1	2.9	7.8 (9%)
SRAM	2	0.7	0	0	1	3.7 (26%)
SBoard	2	0.7	0	0	10	12.7 (5%)

Table 2: Write times for a page of a large NFS file using FDDI

Note that the 1 msec estimates for packet overhead become the dominant time for the Dedicated logging FDDI, SRAM Ethernet, and SRAM FDDI designs. Trimming this overhead exposes the disk transfer time as the bottleneck.

The dedicated low latency logging using Ethernet delivers performance of one half of the LAN. This is extraordinary single – client service for a file server. Compare this to the 9% delivered by the naive (and the most common!) implementations. However, when the FDDI network is used, the transfer rate difference between the disk and the network is a mismatch. The 2.4 megabyte per second transfer rate of the disk, the bottleneck, is about a fifth of the rate of the network. Disk bandwidth can be increased if necessary by a variety of techniques [Kim86] [Patt88] [Sale86]. Higher transfer rate disks are available, but costly.

Also note the importance of good transfer rate to the stable memory. Where the stable memory is slow as in SBoard, Dedicated logging has similar performance. Dedicated low latency logging out performs SBoard.

A simple simulator was written to determine the expected latencies. A few values computed by the simulator follow in Table 3. The simulations were for disks turning at 3600 RPM and a sector size of 512 bytes. 8K log records were written. Clocking stands for a fixed deterministic time between write requests (typical of a fast producer situation). Log % full is the threshold for when to move the log to the next cylinder. Sectors is the number of sectors in a track (32 for 1 megabyte per second disks, 80 for the NEC disk, and 100 for 3 megabyte per second disks). Tracks is the number of logical read/write heads. Latency % of disk I/O is the percent for the average disk I/O that is latency. A dedicated log device was assumed.

Clocking	% log full	Sectors	Tracks	Latency msec	% latency of disk I/O
Yes	70	32	5	.11	1.3%
Yes	80	32	5	.24	2.9%
Yes	90	32	5	.92	11%
No	70	32	5	.47	5.7%
No	80	32	5	.82	9.9%
Yes	70	80	27	.11	3.3%
Yes	80	80	27	.11	3.3%
Yes	90	80	27	.11	3.3%
No	70	80	27	.11	3.4%
No	80	80	27	.15	4.5%
No	90	80	27	.30	9.0%
No	95	80	27	.52	15.7%
Yes	70	100	5	.13	4.8%
Yes	80	100	5	.29	11%
Yes	90	100	5	1.1	42%
Yes	60	100	1	1.0	37%
Yes	80	100	1	1.3	50%
No	50	100	5	.17	6.4%
No	70	100	5	.36	14%
No	75	100	5	.49	19%
No	80	100	5	.59	22%
No	60	100	1	1.4	52%
No	80	100	1	2.2	83%

Table 3: Latency Simulations

Note that the expected value for latency for normal logging for 32 sectors is 100% of the I/O time (there is as much latency as transfer, since the transfer takes half a track). The worst value above for 5 heads is 11% (90% full). For 100 sectors, the expected value for latency grows to about 300% (transfer is a sixth of a track and latency takes half a track). The worst value above for 5 heads is 42% (90% full). At 80% full for 5 head disks, logging latency is reduced by about a factor of 30. The disks with 27 heads are the NEC disks [NEC87]. Note that the latency only starts to climb at about 90% log full for these disks.

The surprise may be how high latency is for one head. In the simulation, with five

heads and 80% full, it was very rare that a delay of more than two sector times was required. Runs from various heads could be pieced together with minimal missed sectors. With only one head, when a run had to be broken, at least the next 16 sectors were already taken.

The simulations assume a particular disk's characteristics. The number of heads is the logical number of heads (some disks do parallel transfers using multiple heads). Disks that have a better seek time, a much higher transfer rate, and that turn somewhat faster are under development and/or are used in high performance systems. One of the components of disk performance that is improving the most is transfer rate, which makes this technique more important.

5. Extensions

Throughout this paper, it has been assumed that the log is not replicated and that it resides on a single drive. That drive may be dedicated to logging to save disk seek time.

The log is key to recovery. Loss of the log would make contents of the database or file system unpredictable. Even though many file systems do have unpredictable contents after a crash, this is not a desirable feature.

By keeping two copies of the log, the reliability of the system is greatly enhanced. By preallocating the logs to the physical middle cylinders on a disk, average seek time can be decreased. A disk head should be seeked to the log tail during periods of inactivity (after a suitable delay). As suggested by a referee, by using more than one area of a disk for logging and selecting which area to use based on head position, seek time can be decreased.

To write a log record, choose disks that have minimum I/O time. I/O time is computed as the time to finish existing I/O's for this disk (if any), do a seek, and perform the log transfer using the algorithm above. Note that log sectors are written onto any disk log. The backchaining now has to also have a drive number, and may have multiple chain addresses for the copies. Log records may be split and parts written to different disks.

For recovery, log records are recovered from any log. To start with for each disk, a cylinder of log records is read. All of the cylinders are sorted by page sequence number. The log records are processed in ascending page sequence number order, regardless of which disk they came from. When all of the records have been used from a cylinder for a disk, a new cylinder is read, sorted, and added to the list of records to

be processed. Of course, this read can be done ahead of time. Duplicates are eliminated. If more than one copy of the log record is read, the copies are checked against each other. This is good for detection of certain classes of errors. (There should be checksums or a third copy to resolve which copy was good.)

If a drive is underutilized, it will tend to become the logging device (or one of the logging devices). As seek times will often be zero, the system will perform like it has a dedicated logging device.

6. Conclusions

This paper proposes a method to do low latency logging for file systems and databases. It greatly improves the latency for log writing. It is particularly effective with a dedicated log device. Multiple disk heads and higher transfer rates are exploited by this technique. The only technique that performs better is to use stable memory. The primary reason for this is the higher bandwidth to stable memory than to disk.

This technique can also be used in high throughput systems where the log device's effective bandwidth is a bottleneck. "Dropped revolutions" are nearly eliminated.

Not all systems are tuned for low latency. For systems requiring low latency for logging and where the transfer rate of the disk for logging is not the bottleneck, then this technique can deliver a dramatic latency improvement.

References

[Gawl85a] D. Gawlick and D. Kinkade. Varieties of Concurrency Control in IMS/VS Fast Path, Tandem Computers Technical Report TR85.6.

[Gawl85b] D. Gawlick, J. Gray, W. Iimura and R. Obermarck. ``Method and Apparatus for Logging Journal Data Using a Log Write Ahead Data Set,'' U. S. Patent 4507751 issued to IBM, March 1985.

[Gray87] J. Gray and F. Putzolu. ``The 5 Minute Rule for Trading Memory of Disc Assesses and The 10 Byte Rule for Trading Memory for CPU Time,'' Proceedings of SIGMOD '87, May 1987, 395 – 398.

[Hagm86] R. Hagmann. ``A Crash Recovery Scheme for a Memory Resident Database System,'' IEEE Transactions on Computer Systems, Vol. 11, No. 1, pp. 839 – 843, March 1986.

[Hell88] Helland et al. Group Commit Timers in High Volume Transaction Systems, Tandem Computers Technical Report TR88.1.

[Kaun84] J. Kaunitz AND L. Van Ekert. ``Audit Trail Compaction for Database Recovery,'' Communications of the ACM, Vol. 27, No. 7, pp. 678 – 683, July 1984.

[Kim86] M. Kim. ``Synchronized Disk Interleaving,'' IEEE Transactions on Computers, Vol. 35, No. 11, Nov. 1986, 978 – 988.

[Lyon89] R. Lyon and R. Sandberg. Breaking through the NFS Performance Barrier, paper available through Legato Systems, Inc.

[Meht91] V. Mehta and R. Khemani. Tuning the SPARCserver 490 for Optimal NFS Performance, SUN Microsystems white paper.

[NEC87] NEC Disk Drive D2363 9 – Inch Winchester Disk Drive Product Description, NEC Corporation.

[Oust85] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. ``A Trace – Driven Analysis of the UNIX 4.2 BSD File System,'' Proceedings of Tenth SOSP, Dec. 1985, 15 – 24, (OSR Vol. 19, No. 5).

[Patt88] D. Patterson, G. Gibson, and R. Katz. ``The Case for Redundant Arrays of Inexpensive Disks (RAID),'' Proceedings of SIGMOD '88, June 1988, 109 – 116.

[Sale86] K. Salem and H. Garcia – Molina, ``Disk Striping,'' Proceedings of the Second Data Engineering Conference, Feb. 1986, 336 – 342.

[SUN86] Network File System Protocol Specification, Version 2, Revision B, Feb. 1986.

XEROX

Low Latency Logging

Robert B. Hagmann

CSL-91-1