Characterizing Parallel File-access Patterns on a Large-scale Multiprocessor*

A. Purakayastha and Carla Ellis
Dept. of Computer Science
Duke University
Durham, NC 27708
{ap,carla}@cs.duke.edu

David Kotz and Nils Nieuwejaar
Dept. of Computer Science
Dartmouth College
Hanover, NH 03755
{dfk,nils}@cs.dartmouth.edu

Michael L. Best
Media Laboratory
MIT
Cambridge, MA 02139
mikeb@media.mit.edu

Abstract

High-performance parallel file systems are needed to satisfy tremendous I/O requirements of parallel scientific applications. The design of such high-performance parallel file systems depends on a comprehensive understanding of the expected workload, but so far there have been very few usage studies of multiprocessor file systems. This paper is part of the CHARISMA project, which intends to fill this void by measuring real file-system workloads on various production parallel machines. In particular, here we present results from the CM-5 at the National Center for Supercomputing Applications. Our results are unique because we collect information about nearly every individual I/O request from the mix of jobs running on the machine. Analysis of the traces leads to various recommendations for parallel file-system design.

1 Introduction

Parallel scientific applications require not only fast computation but also a large and fast I/O subsystem to provide the required data-transfer bandwidth. Such an I/O subsystem must be optimized for the most common traits in the I/O workload. Unfortunately, parallel-I/O workloads have not been thoroughly characterized. There have been I/O-workload characterization studies of mainframes, individual Unix workstations, distributed systems, and some scientific vector applications. However, for parallel scientific applications the few characterizations have either been hypothetical or limited to selected applications.

We started the CHARISMA (**CHAR**acterizing I/O in Scientific Multiprocessor Applications) project with the goal of collecting traces from various production parallel scientific workloads. We recorded such details as individual reads and writes, so that spatial and temporal patterns can be determined. The first results of the CHARISMA project involve a tracing study done on an Intel iPSC/860 at NASA's Ames research facility[19]. This paper describes results from a different platform, the CM-5 at the National Center for Supercomputing Applications (NCSA). Some of the questions that we are trying to answer from the collected data are:

About Jobs: How many jobs ran concurrently? On how many processors did they run? How long did they run? How many files did each job open?

About Files: How many files were only read, only written, or both read and written? How large were the files? For how long were they open? How much inter-processor file-sharing was there?

About I/O requests: How were the request sizes distributed? Which request sizes transferred the most data? How many different request sizes were there per file? How much access-sequentiality was there?

About Policies: Did the data suggest usefulness of a particular kind of caching or prefetching?

In Section 2 we outline related work and provide some background information. In Section 3 we describe our data-collection methods. In Section 4 we discuss the analysis results. We conclude in Section 5 by summarizing important observations and inferences.

2 Background

In this section we first outline previous work on file-I/O characterization; then we briefly describe some related multiprocessor file systems; finally we summarize the relevant components of the CM-5 system.

2.1 Workload Studies

There have been many file system workload studies. Smith [30] studied file-access behavior of IBM mainframes. Porcar [25] analyzed dynamic trace data for files

*This work was supported in part by the National Science Foundation under grant number CCR-9113170, the National Center for Supercomputing Applications, NASA Ames Research Center under agreement number NCC 2-849, and Thinking Machines Corporation.

To appeared: IPPS '95 p 165-171

in an IBM batch environment. Floyd and Ellis [12, 13] and Ousterhout *et al.* [23] studied file-access patterns from isolated Unix workstations. Baker *et al.* [3] studied access patterns in Sprite, a distributed Unix system. Ramakrishnan *et al.* [28] studied file access patterns in a commercial computing environment, on a VAX/VMS platform.

There have been a few studies of I/O from scientific workloads. Del Rosario and Choudhary [10] provided an informal characterization of some grand challenge applications. Powell [26] concentrated mainly on file sizes on a Cray-1. Miller and Katz [21] and Pasquale and Polyzos [24] studied I/O-intensive Cray applications. Jensen and Reed traced file archive activity on a Cray at NCSA [16].

Experimental studies of I/O from parallel scientific programs running on multiprocessors have been rather limited. Crockett [7] and Kotz and Ellis [18] described hypothetical characterizations of a parallel scientific file system workload. Cormen and Kotz [6] discussed desirable characteristics of parallel I/O algorithms. Reddy *et al.* [29] studied I/O from parallelized sequential applications, but their applications were handpicked and I/O was not parallel. Cypher *et al.* [8] studied selected parallel scientific applications, mainly to establish temporal patterns in I/O rates. Galbreath *et al.* [15] used anecdotal evidence to provide a high level picture of I/O from some parallel applications. Bagrodia *et al.* proposed using *Pablo* to analyze and characterize specific applications [2].

The only file system workload study of a production parallel scientific computation environment was that of Kotz and Nieuwejaar [19], as part of the CHARISMA project. They instrumented an iPSC/860 at NASA Ames. Here we describe a similar study on the NCSA CM-5 that captured a wide range of applications from a large number of users.

2.2 Existing Parallel File Systems

Existing parallel-I/O models are often closely tied to the machine architecture as well as to the programming model. Typically jobs can access files in different I/O "modes", which determine how a file pointer is shared among clients running on individual nodes [7, 4, 18, 14, 22]. The Hurricane [20] and KSR1 [17] file systems use a memory-mapped interface. The nCUBE [9] and Vesta [5] file systems allow more user control over data layout by providing per-process logical views of the data. In PIFS (Bridge) [11], the file system controls which processor handles which part of the file to exploit memory locality.

2.3 The CM-5

The CM-5 is a scalable message-passing multiprocessor. It may contain from tens to thousands of processing nodes (PNs) and a few Control Processors (CPs). Each PN has only private memory. The PNs communicate via scalable interprocessor communication networks. Typically a group of PNs (called a *partition*) is managed by a CP. Several jobs timeshare a single partition.

The CM-5 supports a variety of I/O devices. The device of interest to us is the Scalable Disk Array(SDA). We concentrated only on the SDA file access because it was the primary high-volume, high-bandwidth storage device on the CM-5 at NCSA. The SDA is an expandable RAID-3 disk system that typically provides 25-200 Gbytes of disk space and I/O bandwidth of 33-264 MB/sec. The SDA is managed by an Input/Output Control Processor (IOCP). The Scalable File System (SFS) resides on the SDA. It is an enhancement of the Unix file system with extensions to support parallel I/O and very large files. Each CP can also have a set of local Unix file systems, which typically hold the executables and the user's private files. The SFS is optimized for parallel high-volume transfer.

The CM-5 supports two primary programming models (data- and control-parallel), each with its own I/O model. We characterize I/O from programs written in either CMF (a Fortran-like data-parallel programming language) or CMMD (a control-parallel messaging library). The CMF programming model presents a single thread of control to the user. CMF I/O is a library of support routines that allows users to access arrays in SDA files via either special library calls or normal Fortran READ and WRITE statements. CMMD allows multiple threads of control, one for each PN. CMMD I/O provides a variety of I/O "modes" – in some, action is taken by a single PN; in others, all PNs co-operatively perform parallel I/O [4].

3 Tracing Methodology

The 512-node NCSA CM-5 is generally divided into 5 partitions of size 32, 32, 64, 128 and 256 nodes; at times the machine is reconfigured as a single 512-node partition. The SDA has 118 data disks and 1 parity disk for a total capacity of about 138 Gbytes. A single file system resides on the SDA. The logical block size of this file system is 29.5 KB and the physical disk block size is 59 KB. There are roughly 1000 user accounts on this machine. The CMF users dominate the CMMD users by roughly 7 to 3 [1].

3.1 Trace Collection

The CHARISMA project is a multiplatform tracing project. We defined a generic set of trace records that logged events such as open, close, read, write, truncate/extend, link/unlink, etc. The actual format of the records differed slightly depending on the platform and programming model. Detailed formats of event records are listed in [27].

We specifically considered user-program I/O only to and from the SDA. Serial NFS I/O was not considered because we expected that it would have much less data traffic due to limited bandwidth.

We instrumented the run-time CMF I/O libraries to collect traces. The normal I/O library was replaced by our tracing library, and all CMF programs linked to the tracing library by default. Almost all CMF jobs that ran on the NCSA CM-5 in the 23-day period from June 28, 1994 (about 10:15 AM) to July 20, 1994 (about 11:30 AM) were traced. Some users preferred not to have their jobs traced and turned off tracing at run-time. Tracing was continuous except for scheduled maintenance periods. In total, 1760 jobs were traced. They represent 434 distinct applications (different executable path names) run by 384 users.

We were concerned both with performance degradation and perturbation of the workload caused by tracing. We wrote the per-job trace files onto the serial Unix file system to avoid contention with SDA I/O. We buffered the trace records in memory and wrote them to the trace file in large blocks. User-provided anecdotal evidence showed less than 5% overhead in execution time for several applications.

The CMMD tracing library was developed off-site, so we were not allowed to make it the default library. We relied on volunteer users who linked their programs to the CMMD tracing library for us to collect traces. We gathered traces from June 23, 1994 to July 6, 1994 from 127 jobs representing 29 distinct applications run by 11 distinct users. The user population represented heavy (probably sophisticated) SDA users who were interested in parallel I/O behavior. This workload was I/O-intensive compared to the CMF workload. This difference should be considered when interpreting the CMMD data.

CMMD I/O is implemented as a client/server architecture in which a privileged CM-5 host process is responsible for running a server loop. We monitored CMMD I/O by piggybacking trace records on the client/server protocols, thereby achieving minimal perturbation. The actual trace records were produced on the CM-5 compute nodes, communicated to the host server, then written to the local Unix file system. Since the clock-skew between PNs was in the order of microseconds, we ignored the theoretical inconsistency in PN-generated timestamps in the context of I/O operations that took milliseconds to finish.

4 Results

In this section we first characterize jobs, then files, and then individual I/O requests. We then analyze for sequentiality, sharing, and synchronization in access patterns.

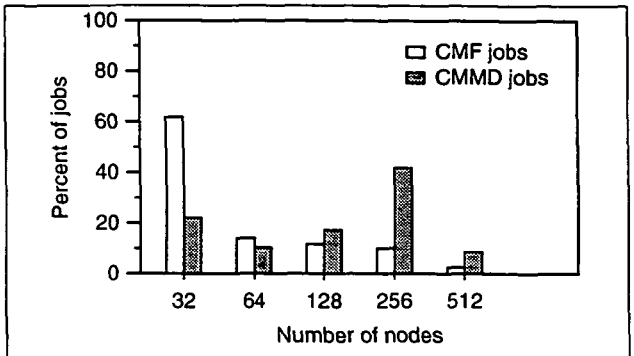


Figure 1: Distribution of the number of nodes used by jobs (choices limited by partition sizes).

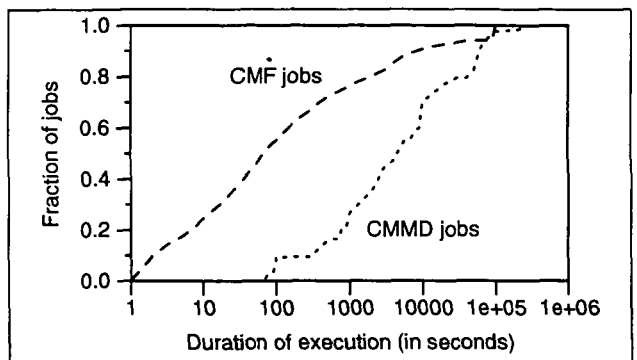


Figure 2: Cumulative Distribution Function (CDF) of duration of jobs.

4.1 Jobs

Figure 1 shows the number of nodes used by jobs. About 60% of CMF jobs used the smallest available partition of size 32 nodes. About 20% of CMF jobs use 128 nodes or more. On the other hand, since the CMMD workload was self-selecting and included fairly large and I/O-intensive applications, we observe a bias toward large numbers of nodes.

The duration of jobs is shown in Figure 2. The number of CMF jobs that ran for each logarithmic quantum were comparable to each other, but taking a closer look we found that there were only a few distinct applications with lifetimes between 1000 and 100000 seconds (about 50), each rerun a large number of times. In contrast, the number of distinct applications that ran for less than 1000 secs is large (more than 400). Indeed, many jobs (about 164) took less than 2 seconds to complete. We believe that these were mostly aborted executions and would be a feature in any general workload. We did not find very short-lived jobs in the CMMD workload, because these were large debugged applications. We also observed that load conditions varied widely over the tracing period. Clearly an effective file sys-

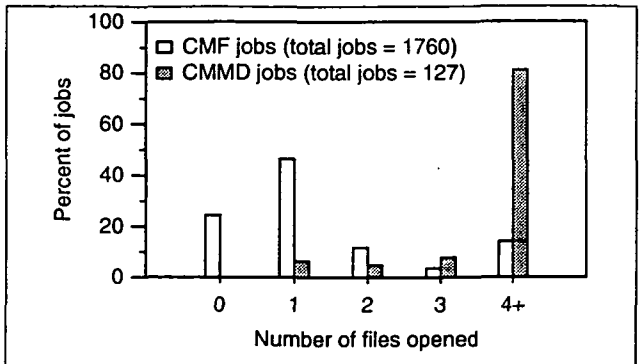


Figure 3: Distribution of number of files opened per job.

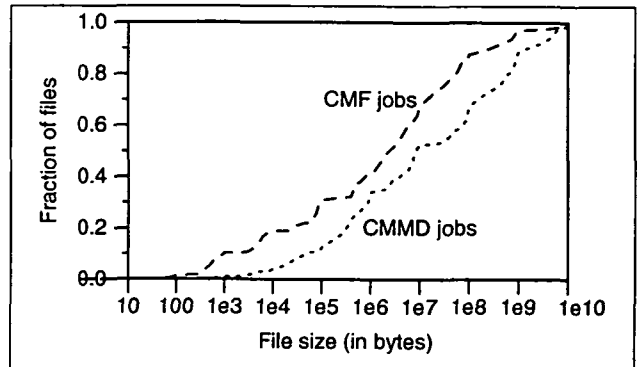


Figure 4: CDF of the number of files of each size at close.

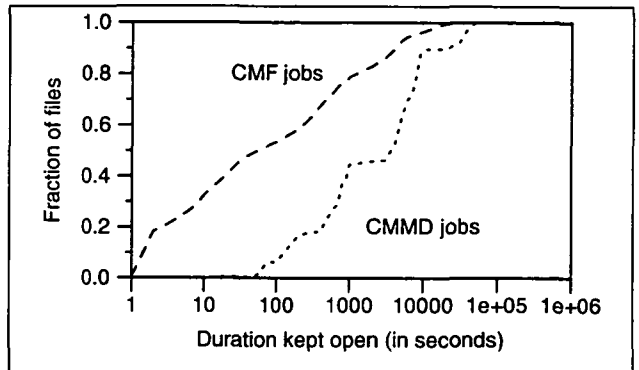


Figure 5: CDF of the duration for which jobs kept files open.

tem must allow efficient access over a range, from small short-lived jobs to large, long jobs and respond to varying system load conditions.

4.2 Files

A total of 1760 CMF jobs opened only 3780 files, while 127 CMMD jobs opened 904 files. We attribute this difference to two factors: CMMD nodes could individually open files (CMF jobs could not), and the CMMD workload was self-selecting and I/O intensive. For the same reason, CMF jobs read 27.8 MB/file and wrote 25.2 MB/file on average, while CMMD applications read 117.5 MB/file and wrote 110.2 MB/file. It should be noted that even the lower CMF figures are an order of magnitude bigger than what was observed in the iPSC study (read 1.2 MB/file, wrote 3.3 MB/file). Write traffic dominated read traffic. CMF jobs used 2286 write-only files and 1271 read-only files; they wrote 57 Gbytes and read 35 Gbytes. CMMD jobs used 596 write-only files and 257 read-only files; they wrote 65 Gbytes and read 30 Gbytes. Very few files (5.8% of those accessed by CMF jobs and 5.9% of those accessed by CMMD jobs) were used simultaneously for both read and write, which is consistent with observations in Unix file systems made by Floyd [12], and with the iPSC results. It is also not surprising, since co-ordinating parallel read-writes from several nodes is difficult.

Figure 3 shows the number of files used per job. About 25% of CMF jobs did not open any SDA files at all, and 63% of CMF jobs opened 1–4 files on the SDA. CMMD jobs opened more files, which is expected from a self-selecting group of users interested in SDA I/O. Both CMF and CMMD jobs had multiple files opened concurrently in their lifetime. Therefore we stress that file systems must optimize access to several concurrently open files within the same job.

Figure 4 shows that file sizes were large compared to Unix environments traced. About 35% of files accessed by CMF jobs and 50% of files accessed by CMMD jobs were larger than 10 MB. During the tracing period, 34 files larger than 10 Gbytes were opened. Parallel file systems must therefore be designed to accommodate efficient access to very large files.

Figure 5 shows the durations for which files were kept open by jobs. For CMMD jobs, more than 50% of files were kept open for more than 1 hour. For CMF jobs about 15% of files were kept open for more than 1 hour. On the whole, file-open durations are much larger than observed in Floyd's Unix file system study [13].

4.3 I/O Request Sizes

Figure 6 shows the size of write requests from jobs. For CMF jobs, the sizes of 90% of write requests were less than 1000 bytes. For CMMD jobs, the sizes of 90% of write requests were less than 400 bytes. We expected CMF write requests to be much bigger because they are collective write requests from all nodes in a job, while CMMD requests were from individual nodes. In CMMD jobs we observed that nodes typically wrote small sequential portions of files. In both CMF and CMMD, more than 90% of

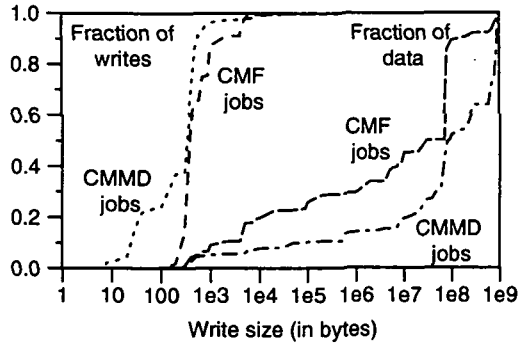


Figure 6: CDF of the number of writes by each request size and the amount of data written by each request size.

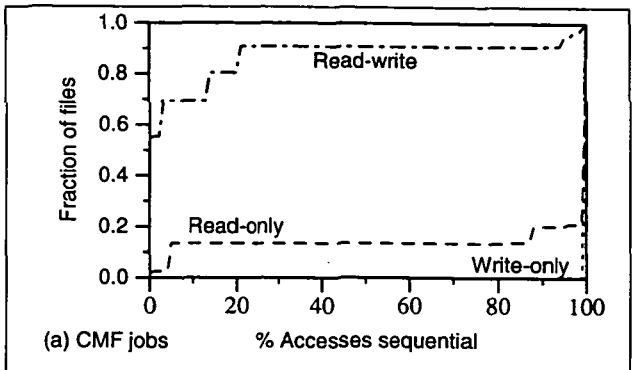
the data were transferred by write requests of size more than 4000 bytes although 90% of write requests were smaller than 4000 bytes. The file system therefore must achieve low latency for small requests (that are numerous) and high bandwidth for large requests (that transfer most of the data). Read request sizes from jobs exhibit behavior similar to the write requests.

4.4 Sequentiality

We define a *sequential* request to be one at a higher file offset than the previous request, and a *consecutive* request to be one that begins exactly at the same offset where the previous request finished. For CMF jobs we looked at the collective access pattern and for CMMD jobs we looked at per-node access patterns. Both CMF and CMMD jobs showed (Figures 7 and 8) predominantly sequential patterns in accessing read-only and write-only files. Some CMF and CMMD jobs read data in reverse of the order in which it was previously written out, which resulted in less sequentiality in read access compared to write access. Read-write files were predominantly non-sequentially accessed. Since accesses from CMF jobs were considered collectively from the application, they show more consecutiveness than CMMD applications, where we looked at per-node patterns. 95% of write-only files and 60% of read-only files were accessed more than 90% consecutively from CMF jobs, while only 60% of write-only files and 20% of read-only files were accessed more than 90% consecutively by CMMD jobs.

4.5 I/O Request Intervals

We define the number of bytes skipped between one request and the next to be the "interval size." Figure 9 shows number of different interval sizes used for each file and Figure 10 shows number of different request sizes used per



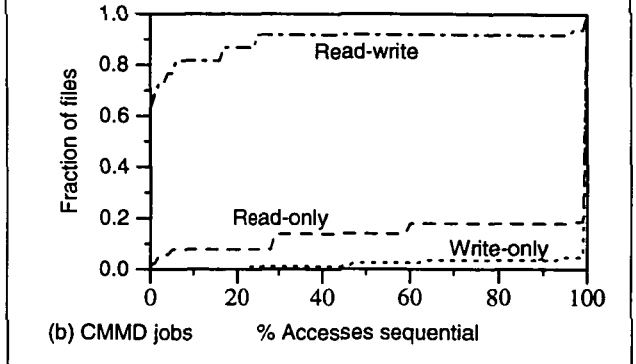
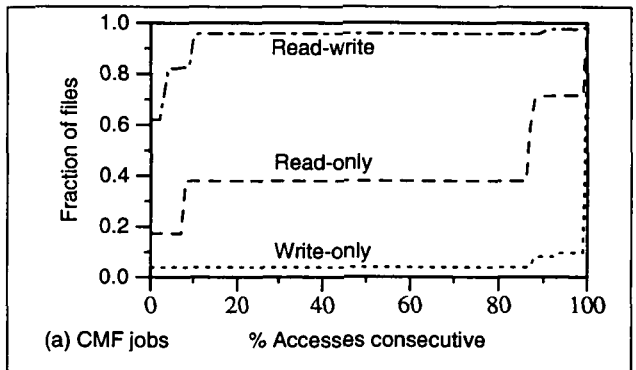


Figure 7: CDF of sequential access.



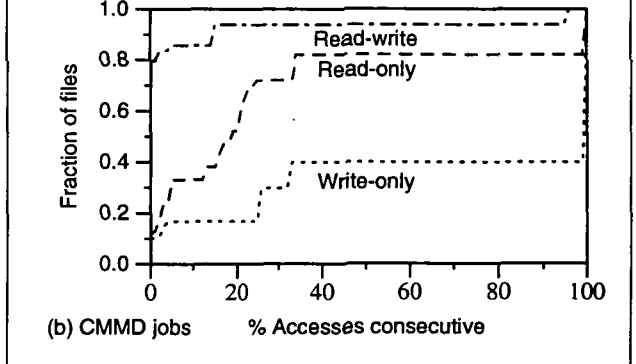


Figure 8: CDF of consecutive access.

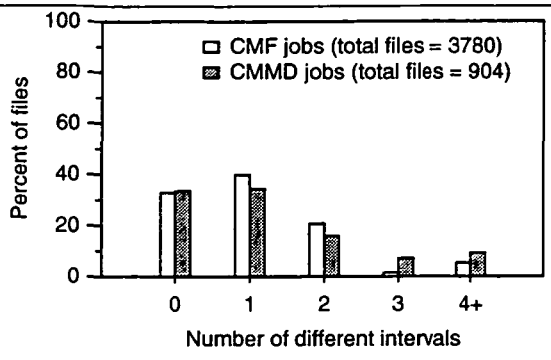


Figure 9: Distribution of number of distinct request intervals per file.

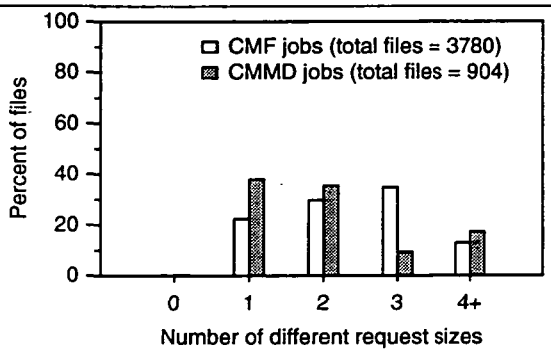


Figure 10: Distribution of number of distinct request sizes per file.

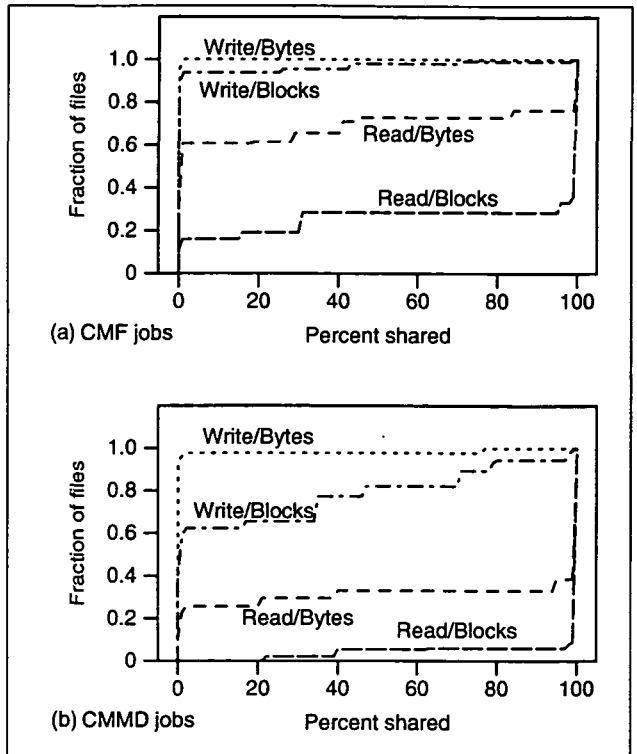


Figure 11: CDF of sharing between nodes in read-only and write-only files at byte and block granularity.

file, considering global and per-node patterns for CMF and CMMD jobs respectively. From CMF jobs 33% of files were accessed as a whole in one request; about 40% of files accessed by CMF jobs were accessed with just 1 interval; about 79% of those 1-interval files were 100% consecutively accessed (interval size 0). From CMMD jobs about a third of all files were accessed as a whole in one request. The percentage of files having 3 or more intervals of access is more for CMMD than in CMF. We attribute this difference to the use of independent I/O modes from CMMD jobs. Overall, the access patterns from both CMF and CMMD applications were predominantly regular (that is, few different interval and request sizes).

4.6 Synchronization

Though CMF users could perform asynchronous nodal I/O via CMMD calls, only 18 (1%) jobs in total used it. CMMD applications also chose to do the bulk (78%) of their I/O in *synchronous-sequential* mode. This mode allows nodes to read/write sequential and possibly unequal file portions in parallel. Most of these accesses also had equal request sizes from nodes. The *local-independent* mode was hardly ever used (0.88% of total I/O), which is expected, because that mode does not provide efficient parallel I/O from SDA files. *Synchronous-broadcast* mode, in which the file pointer is at the same file position in all nodes, accounted for 8.7% of total I/O, and mainly read common information for all nodes. *Global-independent* mode, which allows all nodes to access a single file for independent reading and writing, was only used to do 11.9% of total I/O. Anecdotal evidence suggests that users really want high-performance independent I/O but they do not use it on the CM-5 because of poor performance. This is an example of how the capabilities of an existing machine influence user behavior.

4.7 Sharing

A *shared* file is one that is opened by more than 1 job or node. When the opens overlap in time the file is said to be *concurrently shared*. A file is *write-shared* if any of the opens involve writing the file. We did not find any files shared between jobs, although most files were shared among the nodes of a single job. Thus, we looked at sharing at the byte and block granularity (Figure 11). Of files written by CMF jobs (Figure 11(a)), 95% were completely unshared (that is, 0% byte-shared), because it is rarely mean-

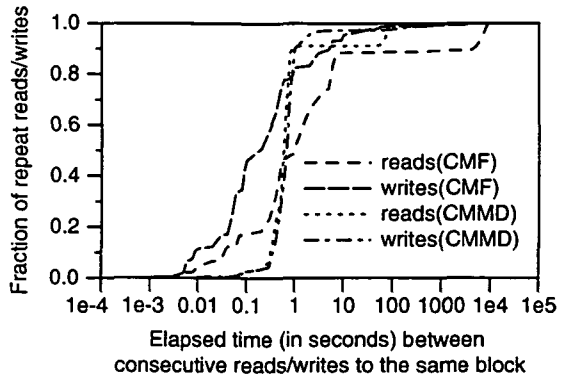


Figure 12: CDF of elapsed time between re-read and re-write of blocks.

ingful for some bytes to be written multiple times. Of files read by CMF jobs, about 24% were completely shared (100% byte-shared), replicating the data set on all nodes. Figure 11(a) also shows that 64% of all read-only files were 100% block-shared, although only 24% were 100% byte-shared. This difference implies that 40% of read-only files had all of their blocks shared despite having few bytes shared, a situation called *false sharing*. This situation occurred when the data set was partitioned among the nodes in such a way so that some of the file blocks contained data destined for different nodes. Read-write files tended to have little byte sharing, because it is relatively difficult to coordinate concurrent read/write access to shared data, but plenty of block sharing, for the same reasons as above.

In CMMD jobs (Figure 11(b)) we found more byte-sharing than in CMF jobs. We do not know the reason for this difference. It may simply be the nature of the particular CMMD applications involved.

Overall, the low amount of write sharing and the high amount of read sharing indicates that caching may be useful, even on the nodes themselves, in addition to caching on the I/O control processor.

4.8 Time between re-writes and re-reads

Figures 12(a) and 12(b) show elapsed time between consecutive accesses to the same block. For CMF jobs, the re-write time was between 0.01 sec and 1 sec in 85% of re-writes. For CMMD jobs, the re-write time was between 0.1 sec and 1 sec in 90% of re-writes. This short interval indicates that writes could be buffered and deferred by perhaps 10 seconds. Combining this fact with low write sharing across nodes, we feel that per-node write caches can be useful. Reads generally showed the same behavior as writes but there were some re-reads that were about 3000 and 100 sec apart for CMF and CMMD applications respectively. We attribute this to long program loops, re-reading the same information at the beginning of each iteration.

4.9 Physical I/O

Both CMMD and CMF provide the users with the facility of "physical I/O", which provides the highest-bandwidth parallel I/O to the SDA but places data in a non-transferable device-dependent layout. Only 26 CMF jobs and 23 CMMD jobs used physical I/O. However, physical I/O accounted for 24.4% and 30% of total I/O from CMF and CMMD jobs respectively. This indicates that some specialized users found the feature very useful.

5 Conclusion

We found important differences between file-access patterns on multiprocessors and Unix platforms including larger files, longer file-lifetimes, dominance of writes, and more concurrent interprocess file-sharing. Low-latency and high-bandwidth both must be achieved to satisfy numerous small requests as well as large requests that transfer most data. We believe that simple extensions of normal Unix I/O will not be effective and that interface re-design is required.

We also found important differences with previous studies of vector scientific applications, particularly the dominance of small I/O requests. To some extent, this is the result of partitioning a data set across many processors, particularly in patterns that did not conform to the layout of data within the file. However, it may also be inherent in some of these applications, since we found that CMF applications— which make only collective-I/O requests— also made small requests.

In CMMD applications we observed that most I/O was done in *synchronous-sequential* mode, with equal amounts of data per node. It appears that a collective-I/O request interface from a CMMD-like environment would be useful.

We found that write traffic (number of files only written, bytes written) was consistently higher than read traffic. Files were not shared between jobs. Most read-only files were either completely shared across nodes within a job or were completely unshared. Write-only files were rarely shared, re-write times to the same blocks were short, and request sizes were small, indicating that node-caching of write-only files may be feasible, if a good cache-consistency solution can be found.

Acknowledgments

We thank Michael Welge for providing access to the NCSA CM-5. We thank Curtis Canada of NCSA for systems support. Many thanks to all NCSA users, especially

Robert Sugar, Diane Cook, Kathryn Johnston, Fady Najjar, Kapil Mathur, Greg Bryan, Chris Kuszmaul, and Tom Cortese. Thanks to David Phillimore at TMC for helping with the CMF sources, and Doreen Revis at Duke for helping us acquire sources and documentation.

References

[1] Personal communication with NCSA consulting staff and NCSA CM-5 systems staff, June 1994.

[2] R. Bagrodia, A. Chien, Y. Hsu, and D. Reed. Input/output: Instrumentation, characterization, modeling and management policy, 1994. On WWW at http://www.ccsf.caltech.edu/SIO/SIO.html.

[3] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a distributed file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212, Oct 1991.

[4] M. L. Best, A. Greenberg, C. Stanfill, and L. W. Tucker. CMMD I/O: A parallel Unix I/O. In *Proceedings of the Seventh International Parallel Processing Symposium*, pages 489–495, 1993.

[5] P. F. Corbett, D. G. Feitelson, J.-P. Prost, and S. J. Baylor. Parallel access to files in the Vesta file system. In *Proceedings of Supercomputing '93*, pages 472–481, 1993.

[6] T. H. Cormen and D. Kotz. Integrating theory and practice in parallel file systems. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 64–74, Hanover, NH, June 1993.

[7] T. W. Crockett. File concepts for parallel I/O. In *Proceedings of Supercomputing '89*, pages 574–579, 1989.

[8] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 2–13, 1993.

[9] E. DeBenedictis and J. M. del Rosario. nCUBE parallel I/O software. In *Eleventh Annual IEEE International Phoenix Conference on Computers and Communications*, pages 0117–0124, April 1992.

[10] J. M. del Rosario and A. Choudhary. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, March 1994.

[11] P. Dibble. *A Parallel Interleaved File System.* PhD thesis, University of Rochester, March 1990.

[12] R. Floyd. Short-term file reference patterns in a UNIX environment. Technical Report 177, Dept. of Computer Science, Univ. of Rochester, March 1986.

[13] R. Floyd and C. Ellis. Directory reference patterns in hierarchical file systems. *IEEE Transactions on Knowledge and Data Engineering*, 1(2):238–247, June 1989.

[14] J. C. French, T. W. Pratt, and M. Das. Performance measurement of the Concurrent File System of the Intel iPSC/2 Hypercube. *Journal of Parallel and Distributed Computing*, 17(1–2):115–121, January and February 1993.

[15] N. Galbreath, W. Gropp, and D. Levine. Applications-driven parallel I/O. In *Proceedings of Supercomputing '93*, pages 462–471, 1993.

[16] D. W. Jensen and D. A. Reed. File archive activity in a supercomputing environment. In *International Conference on Supercomputing*, pages 387–396, 1993.

[17] Kendall Square Research. *KSR1 technology background*, January 1992.

[18] D. Kotz. Multiprocessor file system interfaces. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, pages 194–201, 1993.

[19] D. Kotz and N. Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of Supercomputing '94*, pages 640–649, Nov 1994.

[20] O. Krieger and M. Stumm. HFS: a flexible file system for large-scale multiprocessors. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 6–14, Hanover, NH, June 1993.

[21] E. L. Miller and R. H. Katz. Input/Output behavior of supercomputer applications. In *Proceedings of Supercomputing '91*, pages 567–576, November 1991.

[22] B. Nitzberg. Performance of the iPSC/860 Concurrent File System. Technical Report RND-92-020, NAS Systems Division, NASA Ames, December 1992.

[23] J. Ousterhout, H. DaCosta, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of 10th Symposium on Operating System Principles*, pages 15–24, December 1985.

[24] B. K. Pasquale and G. C. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of Supercomputing '93*, pages 388–397, 1993.

[25] J. Porcar. File migration in distributed computer systems. Technical Report LBL-14763, Lawrence Berkeley Lab, July 1982.

[26] M. L. Powell. The DEMOS file system. In *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pages 33–42, November 1977.

[27] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. Technical Report CS-1994-33, Dept. of Computer Science, Duke University, October 1994.

[28] K. K. Ramakrishnan, P. Biswas, and R. Karedla. Analysis of file I/O traces in commercial computing environments. In *Proceedings of ACM SIGMETRICS and PERFORMANCE '92*, pages 78–90, 1992.

[29] A. Reddy and P. Banerjee. A study of I/O behavior of Perfect Benchmarks on a multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 312–321, 1990 1990.

[30] A. Smith. Analysis of long term file reference patterns and their applications to file migration algorithms. *IEEE Trans. Softw. Eng.*, SE-7(4):403–417, July 1981.