CS–1994–33

Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor[1]

Apratim Purakayastha and Carla Schlatter Ellis[2]
David Kotz and Nils Nieuwejaar[3]
Michael Best[4]

Department of Computer Science

Duke University

Durham, North Carolina 27708–0129

October 24, 1994

[1] 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.

[2] Duke University, email: [ap,carla]@cs.duke.edu
[3] Dartmouth College, email: [dfk,nils]@cs.dartmouth.edu
[4] M.I.T, email: mikeb@media.mit.edu, also affiliated with Thinking Machines Corporation

Characterizing Parallel File-Access Patterns on a Large-Scale Multiprocessor[*]

Apratim Purakayastha and Carla Schlatter Ellis[†]
David Kotz and Nils Nieuwejaar[‡]
Michael Best[§]

October 24, 1994

Abstract

Rapid increases in the computational speeds of multiprocessors have not been matched by corresponding performance enhancements in the I/O subsystem. To satisfy the large and growing I/O requirements of some parallel scientific applications, we need parallel file systems that can provide high-bandwidth and high-volume data transfer between the I/O subsystem and thousands of processors.

Design of such high-performance parallel file systems depends on a thorough grasp of the expected workload. So far there have been no comprehensive usage studies of multiprocessor file systems. Our CHARISMA project intends to fill this void. The first results from our study involve an iPSC/860 at NASA Ames. This paper presents results from a different platform, the CM-5 at the National Center for Supercomputing Applications. The CHARISMA studies are unique because we collect information about every individual read and write request and about the entire mix of applications running on the machines.

The results of our trace analysis lead to recommendations for parallel file system design. First, the file system should support efficient concurrent access to many files, and I/O requests from many jobs under varying load conditions. Second, it must efficiently manage large files kept open for long periods. Third, it should expect to see small requests, predominantly sequential access patterns, application-wide synchronous access, no concurrent file-sharing between jobs, appreciable byte and block sharing between processes within jobs, and strong interprocess locality. Finally, the trace data suggest that node-level write caches and collective I/O request interfaces may be useful in certain environments.

[*] 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.

[†] Duke University, email: [ap,carla]@cs.duke.edu

[‡] Dartmouth College, email: [dfk,nils]@cs.dartmouth.edu

[§] M.I.T, email: mikeb@media.mit.edu, also affiliated with Thinking Machines Corporation

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. A parallel file system and 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. In Section 2 we see that there have been I/O-workload characterization studies of mainframes, individual Unix workstations, distributed systems, and some scientific vector applications. For parallel scientific applications, however, the few characterizations have either been educated guesses or sketchy I/O-workload studies with selected applications.

To bridge this obvious gap in our understanding of the parallel I/O workload, we started the CHARISMA (**CHAR**acterizing **I**/O in **S**cientific **M**ultiprocessor **A**pplications) project in June 1993. CHARISMA is a coordinated joint effort involving people from universities, national research laboratories, and industry[1]. The goal of this project is to collect traces from production scientific workloads, recording such details as individual reads and writes. We collect generic I/O information from applications across many platforms along with machine-specific information 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[KN94]. This paper describes results from our second platform, the CM-5 at the National Center for Supercomputing Applications (NCSA). The machine has a sizeable, established, nationwide user population. 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? How many distinct applications were represented by the jobs?

- **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-node file-sharing was there? How many were temporary files?

- **About I/O requests:** What were the request sizes? How were the request sizes distributed? Which request sizes transferred the most data? How many different request sizes were there per file? How much sequentiality was there in access? What was the typical elapsed time between repeated reads or writes of the same block?

- **About Policies:** Is there evidence to suggest usefulness of a particular kind of caching or prefetching? Are there indications of the usefulness of deferred writes?

- **About File Systems:** Based on our data, what parts of existing parallel file systems need rethinking?

[1] An outline of the CHARISMA project is available online via the World Wide Web, at URL http://www.cs.dartmouth.edu/research/charisma.html.

In Section 2 we outline related work on workload characterization and parallel file system design. In Section 3 we describe our data-collection and analysis methods. In Section 4 we discuss the results. We conclude in Section 5 by describing our overall perspective and our future directions.

2 Background

In this section we first outline some previous efforts to characterize file I/O on various platforms. Then we briefly describe some related multiprocessor file systems, and summarize the relevant components of the CM-5 system.

2.1 Workload Studies

There have been many file system workload studies. Smith [Smi81] studied file-access behavior of IBM mainframes to predict the effects of automatic file migration on files used by an interactive editor. Porcar [Por82] analyzed dynamic trace data for files in an IBM batch environment. Floyd and Ellis [Flo86, FE89] and Ousterhout *et al.* [ODH$^+$85] studied file-access patterns from isolated Unix workstations. Baker *et al.* [BHK$^+$91] studied access patterns in Sprite, a distributed Unix system. Ramakrishnan *et al.* [RBK92] studied file access patterns in a commercial computing environment, on a VAX/VMS platform. The workload was dominated by office-management and transaction-processing applications.

Studies of I/O from scientific workloads have been fairly limited in scope and generality. Del Rosario and Choudhary [dC94] provided an informal characterization of some grand-challenge applications. Powell [Pow77] concentrated mainly on files sizes on a Cray-1. Miller and Katz [MK91] and Pasquale and Polyzos [PP93] studied I/O-intensive Cray applications.

Experimental studies of I/O from parallel scientific programs running on multiprocessors have been rather limited. Crockett [Cro89] and Kotz and Ellis [Kot93] described hypothetical characterizations of a parallel scientific file system workload. Cormen and Kotz [CK93] discussed desirable characteristics of parallel I/O algorithms. Reddy *et al.* [RB90] studied I/O from parallelized sequential applications, but their applications were handpicked and I/O was not parallel. Cypher *et al.* [CHKM93] studied self-selected parallel scientific applications, mainly to establish temporal patterns in I/O rates. Galbreath *et al.* [GGL93] have used anecdotal evidence to provide a high level picture of I/O from some parallel applications.

The only file system workload study of a production parallel scientific computation environment was that of Kotz and Nieuwejaar [KN94], as part of our CHARISMA project. The eventual goal of CHARISMA is to characterize differences between different platforms and programming styles; to isolate similar behaviors across platforms and pinpoint their causes; and to make use of that knowledge in file system and I/O subsystem design. Kotz and Nieuwejaar [KN94] characterized I/O on an iPSC/860 at NASA Ames. This paper describes a similar study on the NCSA CM-5, which has a larger user population and two different programming models. We traced a wide range of applications from a large number of distinct users.

2.2 Existing Parallel File Systems

A single, coherent parallel file-access model has not yet emerged. The parallel-I/O models are often closely tied to the machine architecture as well as to the programming model. Nonetheless, the fundamental idea of declustering file blocks across many disks for parallel access has been a common feature. Typically jobs can access files in different I/O “modes”, which determine how a file pointer is shared among clients running in individual nodes [Cro89, BGST93, Kot93, Pie89, Roy93]. HFS [KS93] and the KSR1 [Ken92] file system, use a memory mapped interface. On the nCUBE [DdR92] and in Vesta [CFPB93] the user has more control over data layout. These systems provide a per-process logical view of the data. In PIFS (Parallel Interleaved File System) [Dib90], the file system controls which processor handles which part of the file to exploit memory locality. Intel’s CFS (Concurrent File System) [FPD93, Nit92, Pie89], on the other hand, provides a Unix-like interface with a choice of four I/O modes to coordinate parallel access. On the CM-5 the I/O model depends on the programming model that is being used; one model provides a Unix-like interface with different I/O modes.

2.3 The CM-5

The CM-5 has a scalable architecture, containing from tens to thousands of processing nodes (PNs) based on RISC processors, and a few Control Processors (CPs). Each PN has its own memory. PNs can fetch instructions from the same address in their memories to execute the same instructions (SIMD-style), or from different addresses to execute independent instructions (MIMD-style). Typically a group of PNs (called a *partition*) is managed by a CP. Each job executes on a single partition but may exchange data with other partitions. Within individual partitions, jobs are timeshared. The PNs communicate via two scalable interprocessor communication networks called the Data and the Control Networks [Thi93b].

The CM-5 supports a variety of I/O devices [Thi93b, Thi93a]. Devices like the Datavault, VMEIO Host Computer, and CM-IOPG reside on a high-speed multidrop CMIO bus and are abstracted by the CMFS file system. However, the device of interest to us is the SDA (Scalable Disk Array). Our tracing study concentrates solely on the SDA file access because it was the main 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 bandwidths of 33-264 MB/sec. The SDA (or any I/O device) is managed by an IOCP (Input/ Output Control Processor), which provides software support for file systems, device drivers, and communication protocols. The SFS (Scalable File System) resides on the SDA. It is an enhancement of the Unix file system with extensions to support parallel I/O and very large files. Moreover, each CP can have a set of local Unix file systems with which the SFS can share name space. The Unix file system typically holds the executables, and the user’s private files. The SFS is optimized for parallel high-volume transfer, even though it is a fully general file system.

The CM-5 supports two primary programming models (data- and control-parallel), each with its own I/O model. In this paper 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; all nodes appear to be executing identical code though they may be operating on different data. CMF I/O is a library of support routines that are layered on top of SFS. They allow users to read and write arrays (or portions thereof) to the SDA via either special library calls or normal Fortran READ and WRITE statements. Issues of data distribution and I/O parallelization are hidden from the user. CMMD may be embedded in a variety of familiar programming languages like C, C++, and f77. Under CMMD the user sees multiple threads of control, one for each PN. CMMD I/O (again layered on top of SFS) provides a variety of I/O "modes" – in some, action is taken by a single PN; in others, all PN's co-operatively perform a parallel I/O [Thi93c, BGST93].

3 Tracing Methodology

The CM-5 at the National Center for Supercomputing Applications (NCSA) was chosen as our target machine because this is one of the most widely used CM-5 machines in the United States. The user population is distributed all across the nation and there are approximately 1000 user accounts on this machine.[2] The CMF user population dominates CMMD users by about 7 to 3.[3] We have captured a large variety of generic classes of CMF and CMMD applications.[4]

The CM-5 at NCSA has 512 nodes. Generally it is divided into 5 static partitions of size 32, 32, 64, 128 and 256 nodes. The partitions are reconfigurable; at times the machine is reconfigured as a single 512-node partition depending on user needs. Each node has a RISC-based CPU, a private network interface and 4 vector units with a collective memory size of 32 MB/node. The SDA has 118 data disks and 1 parity disk for a total capacity of about 138 Gbytes. A single file system called '/sda1' 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.

We must concede that although we have tried to capture different programming styles, our observations are specific to the target computing environment. This dependence is why the CHARISMA project is trying to trace different platforms.

3.1 Trace Collection

The CHARISMA project is a multiplatform tracing project. We defined a generic set of trace records, an appropriate subset of which can be collected at any machine. This generic set logged events such as open, close, read, write, truncate/extend, link/unlink, etc. The format of our generic records was similar across the

[2] Information obtained via personal communication with NCSA CM5 systems staff.

[3] Information obtained via personal communication with NCSA consulting staff.

[4] A short description of some of the applications we traced in this study are available via the World Wide Web at URL http://www.cs.dartmouth.edu/cs_archive/pario/anecdote10.html.

platforms.[5] In addition to generic trace records, we also collected some machine specific records such as, the Physical I/O records on the CM-5. Moreover in CMF where I/O is primarily handled at the application level (as opposed to the 'client' or node level in CMMD), some 'client' records were not collected and some 'client' fields were preassigned. Table 1 summarizes the trace records and their fields.

In addition to the above fields, each record had a record-type field and timestamp fields. The read/write records and the physical-I/O records had two sets of timestamp fields to record start and end times. The high-precision timestamps were 8-byte integer values containing microseconds elapsed after epoch. To accommodate large files (of size larger than 2 Gbytes), the file size and file offset fields were also 8-byte integers. Unique file id's were constructed using the tuple [device number, inode number]. The application id's are just the process id's unique to a partition, the client id's are the particular node numbers in a partition.

We considered I/O only to and from the SDA. Serial NFS I/O was not traced or accounted for, because we expected that it would have much less traffic due to limited bandwidth. Specifically, this study focuses on parallel-I/O by user programs to and from the SDA. Hereafter, all measurements refer to the SFS on the SDA in the CM-5 at NCSA.

3.1.1 Tracing CMF applications

We instrumented the run-time CMF I/O libraries to collect traces. CMF programs normally link to the standard CMF I/O library during compilation. During the tracing period, 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 applications 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. The instrumentation had a mechanism for users to shut off tracing of a particular program by setting an environment variable. Some users (for example, industrial partners of NCSA) requested this feature and made use of it, thereby not having their applications traced. We had, however, a mechanism to count the total number of CMF programs that were compiled and run during the tracing period even if they suppressed trace generation. Out of 1943 such jobs in that period, 1760 were traced. This figure, however does not include programs that were compiled before June 28 (before the tracing library was installed) and ran in the tracing period. The 1760 jobs traced represent 434 distinct[6] applications run by 384 distinct users. except for 6 hour periods on Fridays when the whole machine was brought down for maintenance. During the tracing period there was no other significant unexpected downtime.

We were concerned both with the degree to which tracing would affect the overall performance and with how tracing would perturb the I/O we set out to study. 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 to minimize tracing overhead. Since overhead was a big concern for the NCSA systems staff, they had some users beta-test the tracing library before installing it as default. For a wide variety

[5] Although we have defined a generic format, the semantics of events may be subtly different on different platforms and care must be taken in analyzing traces.

[6] Applications with different path names for executables were considered distinct.

Record Type	**Fields in the Record**
Header	format and version numbers, start date, system type, system id, configuration (nodes, memory, disks, etc.), timestamp unit
Application Load	executable path name, user id, application id, number of clients, client id list
Application Exit	application id
Client Open File (CMMD only)	pathname, file id, client id, file descriptor, file size, creation time, open mode (r,w,rw,create etc.)
Application Open File	application id, file id, file descriptor, file size, creation time, open mode
Client Close File (CMMD only)	client id, file id, file size
Application Close File	application id, file id, file size
Read/Write request	operation type (r,w, sync/async etc.), client id or application id, file descriptor, file offset, size of I/O
Truncate/Extend	client id, file descriptor, original size, new size
Link/Unlink	client id, file id, new number of links
Set I/O Mode (CMMD only)	client id, file descriptor, new access mode
Fcntl call (CMMD only)	client id, application id, machine specific request code
Physical I/O	file descriptors, operation type, application id, number of clients, offset, size of I/O

Table 1: Different of types of trace records that were collected

of applications the testers reported less than 5% overhead in execution time. No on-the-fly trace compaction was necessary because disk space was not an issue with NCSA providing us 1.3 Gbytes of intermediary storage. We transferred data to Duke daily, for tape storage.

3.1.2 Tracing CMMD applications

While we can classify the CMF workload as a "general" workload, the CMMD workload was self-selecting. We developed the CMMD tracing library at Thinking Machines Corporation on an in-house version of CMMD. Since it was developed off-site, the NCSA systems staff were reluctant to make it the default library. Hence we relied on volunteering users who linked their programs to the CMMD tracing library for us to gather traces. We gathered traces from June 23 to July 6, for a period of two weeks. We obtained traces from 127 jobs representing 29 distinct applications run by 11 distinct users. Because they volunteered for this tracing study, they were all heavy (probably sophisticated) SDA users who were interested in parallel I/O behavior. We can perhaps classify this workload as an I/O-intensive workload compared to the "general" 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-node process is responsible for running a server loop. We monitored CMMD I/O by piggybacking trace records on the client/server protocols. 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 communication of trace records was embedded into the normal client/server I/O protocols we believe that perturbation was minimal.

3.2 Trace Analysis

In CMF the trace records were all emitted from the CP. Hence CMF records had timestamps from the same clock, which meant that there was no clock skew to synchronize. In CMMD however, the PNs generated timestamps, and hence in principle there was a clock skew problem. But the clock skew between PNs was on the order of microseconds and not on the order of milliseconds, allowing us to ignore the skew in the context of I/O operations that almost always took milliseconds or more to complete. Therefore, minimal postprocessing was required before trace analysis.

4 Results

Tables 2 and 3 summarize the data collected on a day-by-day basis throughout the tracing period for CMF and CMMD applications respectively. The tables list the number of files opened per day; each file opened was classified by whether it was actually only read, only written, both read and written, or neither read nor written. We also counted the number of files that were created and deleted within the same job, and called them temporary files.

We collected both temporal and spatial information about jobs, files, and I/O requests. Since this is purely a workload study we do not present any performance figures, such as time to complete specific I/O requests, etc. We first characterize jobs, then files, and then individual I/O requests. We then analyze for sequentiality, sharing, and synchronization in access patterns.

4.1 Jobs

Figures 1 and 2 show the number of nodes used by CMF and CMMD jobs. A large number (about 60%)of CMF jobs[7] used the smallest available partition of size 32 nodes. We speculate that use of a small number of nodes by the majority of jobs is a characteristic of any general parallel workload. The same trend was also observed in our iPSC study [KN94]. But an appreciable number of CMF jobs as well as iPSC jobs used a large number of nodes: more than 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 number of nodes. More than 50% of CMMD jobs used 256 nodes or more.

The duration of CMF and CMMD jobs is shown in Figures 3 and 4. The number of CMF jobs that ran for each quantum such as 0 to 10 seconds, 10 to 100 seconds, and so on upto 10000 to 100000 seconds 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). They were 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). In addition, 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 run by a cooperating group of users, and are stable, debugged applications.

We also observed that load conditions varied widely over the tracing period, typically heavy from Monday to Thursday, falling off over Friday[8] to Sunday. Each day's peak was typically around 2 to 3 o'clock in the afternoon. Clearly an effective file system 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

In Table 2 we observe that 1760 CMF jobs opened a total of 3780 files, about 2 SDA files per job. On the other hand, in Table 3 we see that 127 CMMD jobs opened 904 files, about 7 files per job. We attribute this difference to two factors: that CMMD nodes could individually open files while CMF jobs only open files collectively, and that the CMMD workload was self-selecting and I/O intensive. That the CMMD workload was relatively I/O-intesive was also manifested by the fact that 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.[9] It should be

[7] We have used both the words 'job' and 'application' throughout the paper. We mean an 'application' to be a program and a 'job' to be an execution of the program.

[8] This was partially caused by the mandatory maintenance period on Friday.

[9] These averages ignore the few files that were both read and written.

noted that even the lower CMF figures are an order of magnitude bigger than what was observed in our iPSC study (read 1.2 MB/file, wrote 3.3 MB/file). The users seem to have made use of the higher disk capacity and bandwidth that the CM-5 offers over the iPSC at NASA Ames. Very few files (5.8% of those accessed by CMF jobs and 5.9% of those accessed by CMMD jobs), were used for both read and write. This occurance is consistent with observations in Unix file systems made by Floyd [Flo86], and with iPSC results. This is not surprising, since co-ordinating parallel read-writes from several nodes is difficult.

In Table 4 we see that about 25% of CMF jobs did not open any SDA files at all. It is likely that they were compute intensive jobs doing I/O via NFS. About 63% of CMF jobs opened 1–4 files on the SDA. In Table 5, we find that all traced CMMD jobs opened at least one SDA file, which is expected from a self-selecting group of users interested in SDA I/O. Indeed, CMMD jobs in general opened more files on the SDA than CMF ones. Both CMF and CMMD jobs had multiple files opened concurrently in their lifetime. Like our iPSC study, we stress that file systems must optimize access to several concurrently open files within the same job.

We find 3.8% of files accessed by CMF jobs (Figure 2) and 4.9% of files accessed by CMMD jobs (Figure 3) were temporary files, as opposed to 0.76% in our iPSC study. This may indicate that more applications went for "out-of-core" solutions than on the iPSC. However, the number of temporary files are still very few and we believe that only a few distinct applications generated those files.[10]

Files accessed by CMF jobs (Figure 5) as well as CMMD jobs (Figure 6) were large. About 35% of files accessed by CMF jobs were larger than 10 MB and 50% of files accessed by CMMD jobs were larger than 10 MB. During the tracing period, 34 opens were on files that were larger than 10 Gbytes. File sizes observed in our iPSC study though large, were not nearly as large as observed on the SDA. We speculate that the difference was attributable more to the available disk capacity and bandwidth than to the inherent requirements of the workload, and that all workloads would have used more if more was available. Any parallel file system must therefore be designed to accommodate efficient access to very large files.

Figures 7 and 8 show the durations for which files were kept open by CMF and CMMD jobs respectively. For CMMD jobs, more than 50% of files were kept open for more than 1 hour. For CMF jobs, however, only 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 [FE89]. Presumably these long durations were the result of long-running jobs (figures 3 and 4). Along with appreciable difference in prevalent file sizes this is another important characteristic in which this parallel file system workload has differed from Unix environments traced.

[10] We found that number of temporary files were high in the days that certain applications ran. We speculate that the temporary files were generated by only these few applications.

4.3 I/O Request Sizes

Figures 9 and 10 show sizes of write requests from CMF and CMMD jobs respectively. For CMF jobs, the size of 90% of write requests was between 100 and 1000 bytes. For CMMD jobs, the size of 90% of write requests was between 10 and 500 bytes. Though write request sizes from CMF jobs were a little bigger, we expected them to be much bigger because CMF writes 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. The request sizes in both the cases appeared to be insensitive to the 29.5 KB block size of the SFS. In both CMF and CMMD, more than 90% of the data were transferred by write requests of size more than 4000 bytes although 90% of write requests were smaller than 4000 bytes. This similar behavior across CMF, CMMD, and iPSC applications leads us to believe that this feature of I/O requests is fundamental to a large class of parallel applications. The file system design must therefore concentrate on optimizing both low-latency small I/O's and high-bandwidth large I/O's to accommodate this bipolar workload. Figures 11 and 12 show read request sizes from CMF and CMMD applications and they exhibit behavior similar to the write requests.

4.4 Sequentiality

Uniprocessor scientific applications access files in a predominantly sequential manner [MK91]. For parallel applications we need to define *sequential* access carefully. As in the iPSC study, 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 look at collective application access patterns and for CMMD jobs we look at per-node access patterns. Figures 13 and 15 show access sequentiality and consecutiveness from CMF jobs, and Figures 14 and 16 show the same from CMMD jobs. Both CMF and CMMD jobs indeed showed predominantly sequential pattern 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 partially contributed to less measured sequentiality in read access compared to write access. Read-write files, consistent with our observations on the iPSC, were predominantly non-sequentially accessed from both CMF and CMMD jobs. Since accesses from CMF jobs were considered collectively from the application, it shows 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 on the same node to be the "interval size." Consecutive accesses have zero interval size. Table 6 shows number of different interval sizes used for each file and Table 8 shows number of different request sizes used per file accessed by CMF jobs. Tables 7 and 9 show the same for files accessed from CMMD jobs. 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). However, we had appreciable number of 2 or 3 request sizes in files accessed by CMF jobs. This may have been caused by instances where programmers chose to use one file to store various arrays of different sizes. 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 slight difference to the use of independent I/O-modes from CMMD jobs. Although compared to the iPSC study the CM-5 workload exhibits less regularity, the access patterns from both CMF and CMMD applications were still 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[11] , we found 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. In effect, much of CMMD I/O was done in a CMF-like fashion. 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 was used to do only 8.7% of total I/O. This mode was predominantly used to read common information for all nodes, and was never used for write. *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. From the above data, one may be inclined to conclude that applications only need fast synchronous I/O since apparently that is what they used predominantly. But we have anecdotal evidence from users that indicates that they really want high-performance independent I/O but they do not use it on the CM-5 because of poor performance. This behavior 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.

Of files written by CMF jobs (Table 10), 95% were completely unshared (that is, 0% byte-shared). This fact is not surprising, since it is rarely meaningful for some bytes to be written multiple times. Of files read

[11] Just before our tracing began, CMF Version 2.1 final was installed which had this capability. Low usage of this feature may be partially attributed to the fact that it was relatively new.

by CMF jobs, about 24% were completely shared (100% byte-shared), replicating the data set on all nodes. Since all bytes of those files were shared, all blocks of those files were shared as well, that is, they were 100% block-shared. Table 10 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. Finally, files that were both read and written 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 (Table 11) 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 on the I/O control processor.

4.8 Elapsed time between re-writes and re-reads

Figures 17 and 18 show elapsed time between consecutive accesses to the same block from CMF and CMMD jobs respectively. 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 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 an appreciable number of re-reads (about 10% for both CMF and CMMD) that were about 3000 sec apart for CMF applications and 100 sec apart for CMMD applications. 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”. Physical I/O provides the highest-bandwidth parallel I/O to the SDA but places data in a non-transferable device-dependent manner. In CMF only 26 jobs (less than 2% of total number of jobs), and in CMMD 23 jobs(about 19% of the total number of jobs) used physical I/O. But physical I/O from CMF jobs accounts for 23 Gbytes of data transfer, which is 24.4% of total I/O done by CMF jobs. Similarly physical I/O from CMMD jobs accounts for 28 Gbytes which is about 30% of total I/O done from CMMD jobs. These indicate that although many users did not use or need this feature, but some users with significant I/O needs did find the feature very useful. We recommend that parallel file systems provide ‘raw’ I/O backdoors such as this, until higher-level I/O becomes comparable in speed.

5 Conclusion

Our results describe the character of the file-access patterns on the NCSA CM-5, for two programming models: CMF data-parallel programs, and CMMD message-passing programs. Comparing the two programming models, and comparing both to our previous results on the iPSC/860, we found both interesting similarities and interesting differences. 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 caching write-only files may be feasible, if a good cache-consistency solution can be found.

Across all platforms, small I/O requests dominated. 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. We believe, however, that 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, even though independent modes were available. Most of these synchronous-sequential accesses from CMMD applications requested an equal amount of data for all nodes. It appears that a collective-I/O request interface from a CMMD-like environment would be useful.

We can now make comparisons with the initial iPSC study. The study described in this paper allowed us to begin to see differences caused by factors in the specific environments, as well as some similarities that may be inherent to the parallel programming enterprise. It is still only a beginning toward understanding the interplay among “natural” I/O programming styles, the influence of file-system constructs in the programming model, and the effect of hardware facilities available. We hope that the CHARISMA project motivates further attention to I/O-workload characterization as parallel file-system semantics evolve.

Acknowledgments

We thank Michael Welge who provided CM-5 access without which this tracing study would not have been possible. We also thank the NCSA systems and consulting staff, especially Curtis Canada, for providing the systems support needed to accomplish this project. Many thanks to all the NCSA users who cooperated with us, especially Robert Sugar, Diane Cook, Kathryn Johnston, Fady Najjar, Kapil Mathur, Greg Bryan, Chris Kuszmaul, and Tom Cortese. We also thank David Phillimore at Thinking Machines Corporation for all the help he provided in building the CMF tracing libraries, and Doreen Revis at Duke for providing numerous CM-5 insights and helping to acquire sources and documentation.

Date	# Jobs	# files opened	read	written	both	none	temp	MB read	MB written
Jun 28	130	186	63	107	16	0	5	6211.05	9971.77
Jun 29	118	194	71	123	0	0	1	1011.04	800.71
Jun 30	111	276	101	154	21	0	13	8234.32	13422.03
Jul 01	41	159	21	135	3	0	8	864.13	925.02
Jul 02	58	62	21	41	0	0	0	500.12	561.00
Jul 03	20	15	12	3	0	0	0	300.69	122.13
Jul 04	46	38	17	17	1	3	0	489.32	924.13
Jul 05	109	163	35	120	8	0	29	4342.25	6211.13
Jul 06	103	192	59	122	11	0	19	344.14	532.14
Jul 07	95	103	43	60	0	0	11	209.13	281.34
Jul 08	44	81	61	19	1	0	0	81.21	100.13
Jul 09	47	57	20	37	0	0	0	135.34	223.13
Jul 10	29	87	31	56	0	0	3	12.13	113.43
Jul 11	88	294	103	168	23	0	10	1095.96	5621.01
Jul 12	122	225	80	114	31	0	0	434.36	829.12
Jul 13	150	274	60	184	30	0	0	812.26	1782.14
Jul 14	95	190	51	137	2	0	3	1342.16	893.24
Jul 15	40	128	29	98	1	0	12	341.21	632.93
Jul 16	25	33	16	17	0	0	0	12.17	83.43
Jul 17	54	128	68	60	0	0	5	239.43	233.21
Jul 18	85	184	65	104	15	0	0	1011.21	1713.23
Jul 19	122	643	213	376	53	1	23	7124.33	11313.93
Jul 20	28	68	31	34	3	0	0	211.31	341.13
Totals	1760	3780	1271	2286	219	4	142	35359.27	57631.46
		100%	33.6%	60.5%	5.8%	0.1%			

Table 2: A day-by-day summary of the traces collected from CMF jobs.

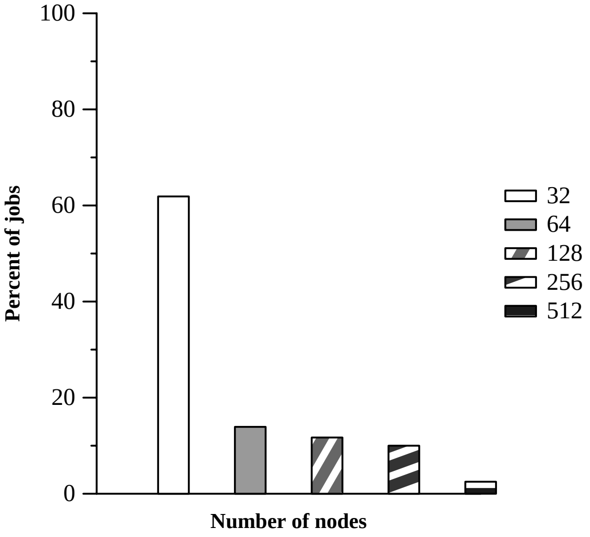

Figure 1: Number of nodes used by CMF jobs.

Date	# Jobs	# files opened	read	written	both	none	temp	MB read	MB written
Jun 23	6	28	8	20	0	0	4	121.24	544.31
Jun 24	11	131	17	106	8	0	13	1672.15	11493.90
Jun 25	15	86	21	62	3	0	2	8943.34	8143.14
Jun 26	13	153	42	102	9	0	12	5341.13	6401.55
Jun 27	18	164	39	114	11	0	1	4100.01	9100.03
Jun 28	9	71	32	35	4	0	0	814.12	1112.41
Jun 29	9	28	11	17	0	0	0	312.41	3142.33
Jun 30	8	34	7	20	5	2	3	112.05	7443.51
Jul 01	6	13	5	8	0	0	0	302.14	344.05
Jul 02	3	18	8	8	2	0	2	139.92	900.02
Jul 03	2	5	3	2	0	0	0	314.41	411.13
Jul 04	7	12	4	7	1	0	0	200.14	135.56
Jul 05	12	98	29	67	2	0	4	5942.34	10078.74
Jul 06	8	63	31	28	4	0	3	1891.11	6443.21
Totals	127	904	257	596	49	2	44	30206.51	65693.89
		100%	28.5%	65.9%	5.4%	0.2%			

Table 3: A day-by-day summary of the traces collected from CMMD jobs.

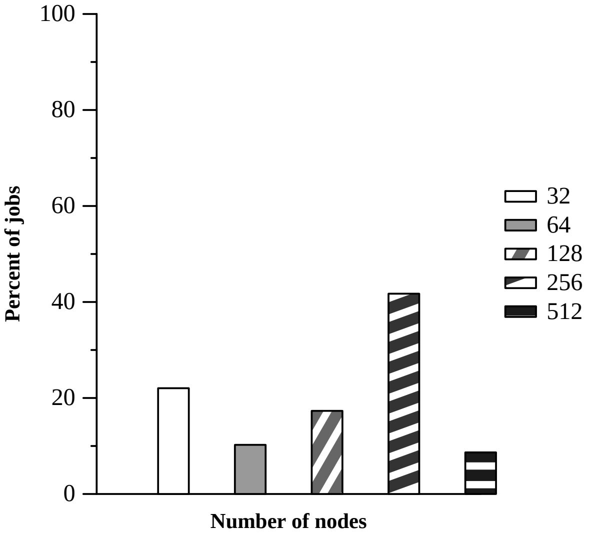

Figure 2: Number of nodes used by CMMD jobs.

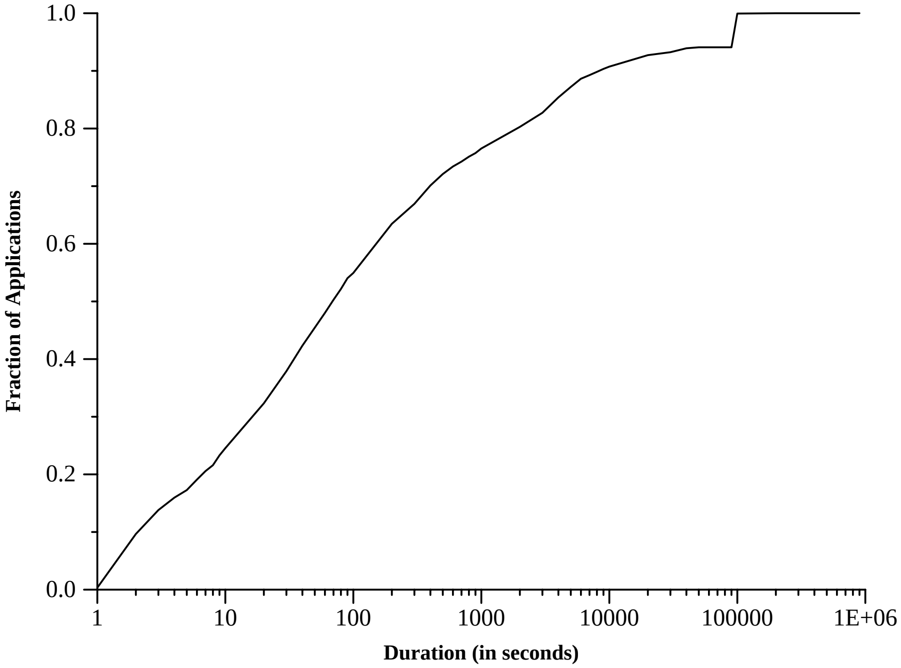

Figure 3: CDF (Cumulative Distribution Function) of duration (in seconds) of CMF jobs.

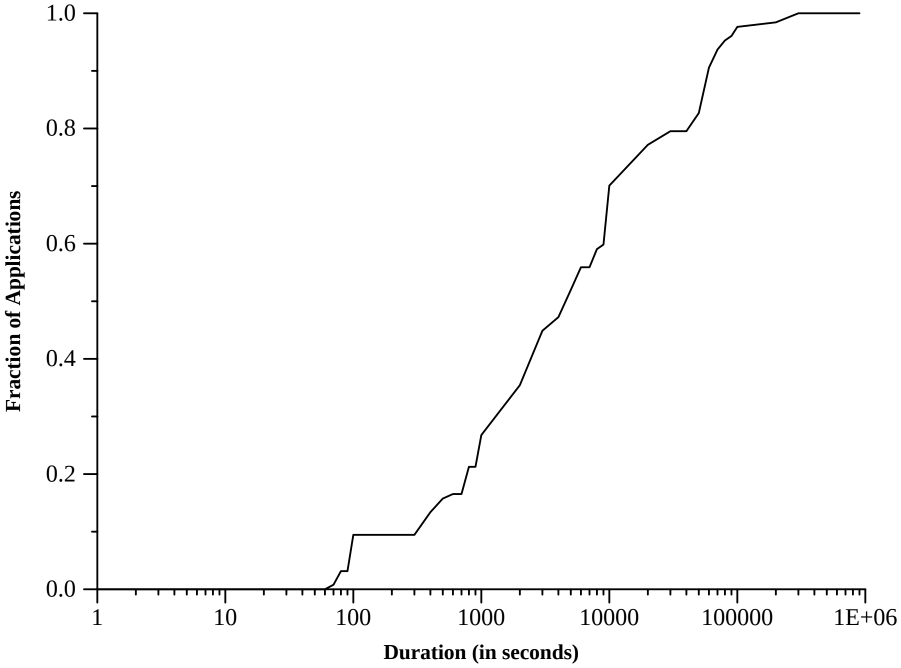

Figure 4: CDF of duration (in seconds) of CMMD jobs.

Number of files opened	Number of jobs
0	431
1	813
2	205
3	63
4	31
5	19
6	31
7	16
8	13
9	4
10+	134

Table 4: Number of files opened per CMF job.

Number of files opened	Number of jobs
0	0
1	8
2	6
3	10
4	14
5	11
6	12
7	33
8	13
9	6
10+	14

Table 5: Number of files opened per CMMD job.

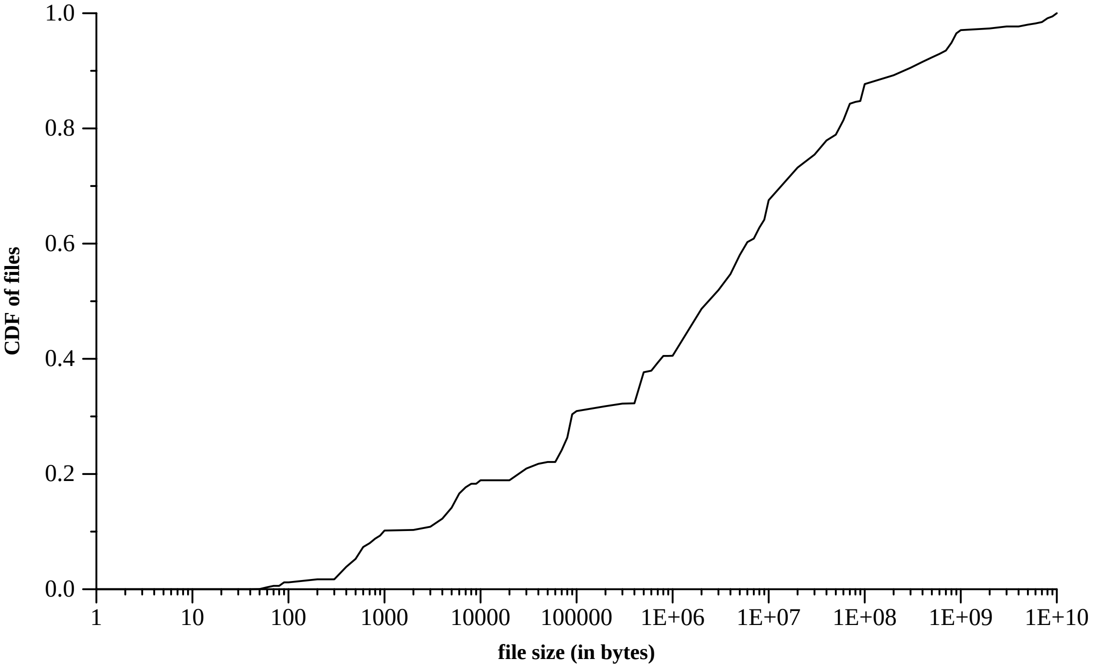

Figure 5: CDF of file size for files opened by CMF jobs.

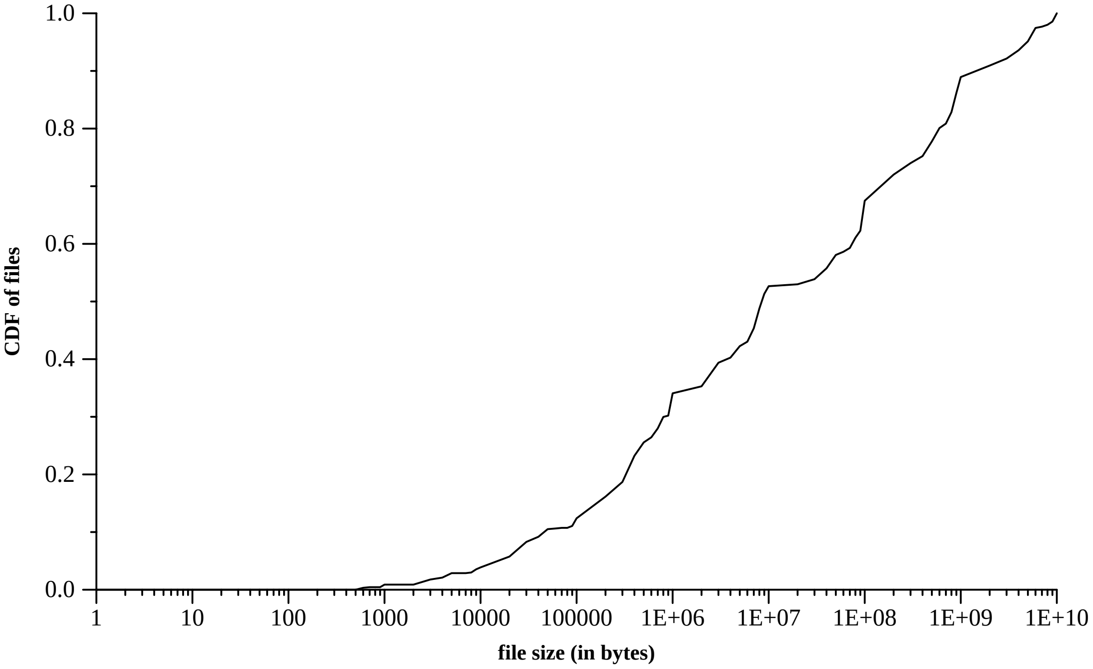

Figure 6: CDF of file size for files opened by CMMD jobs.

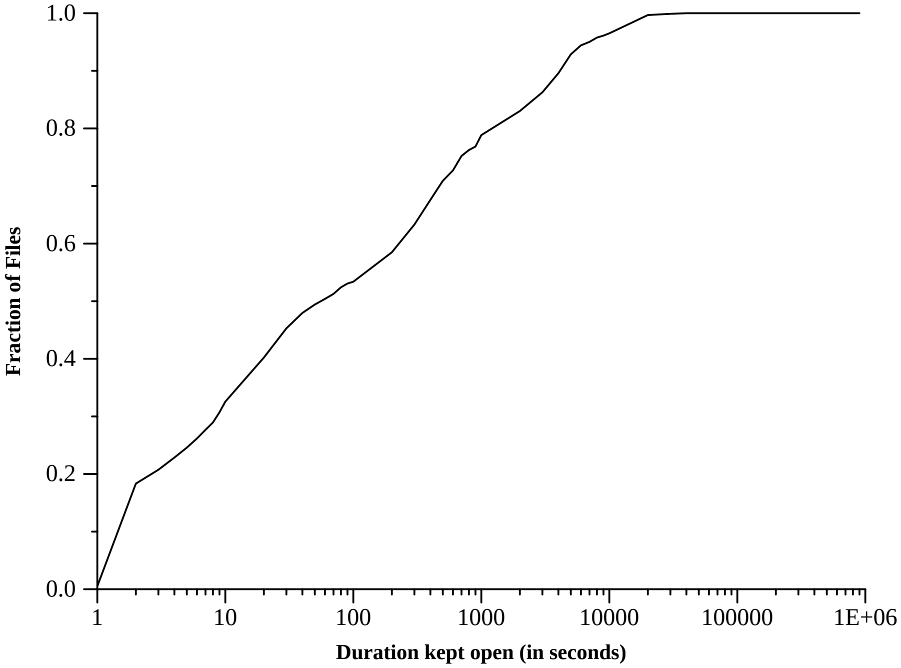

Figure 7: CDF of duration for which CMF jobs kept files open.

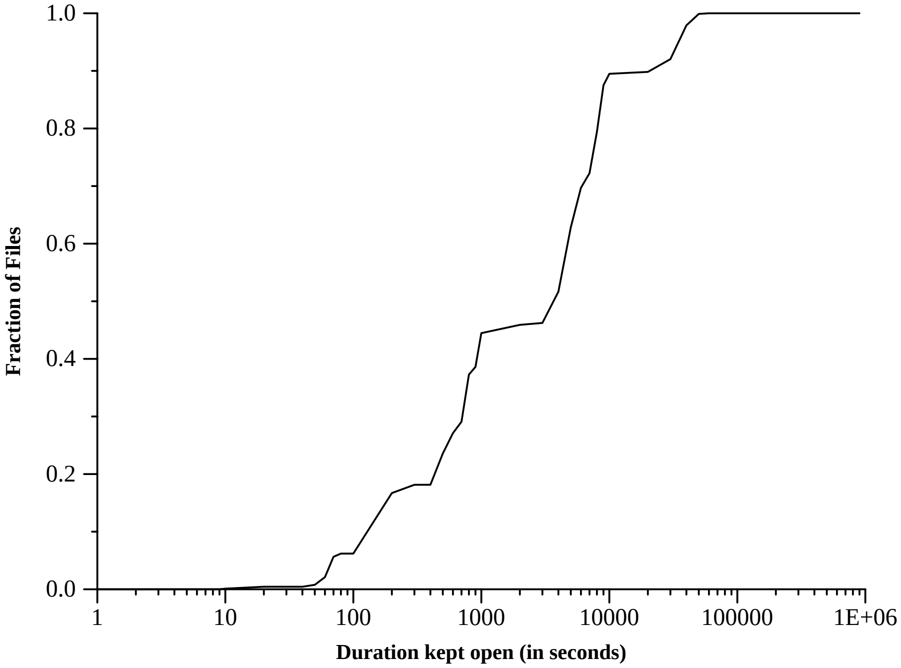

Figure 8: CDF of duration for which CMMD jobs kept files open.

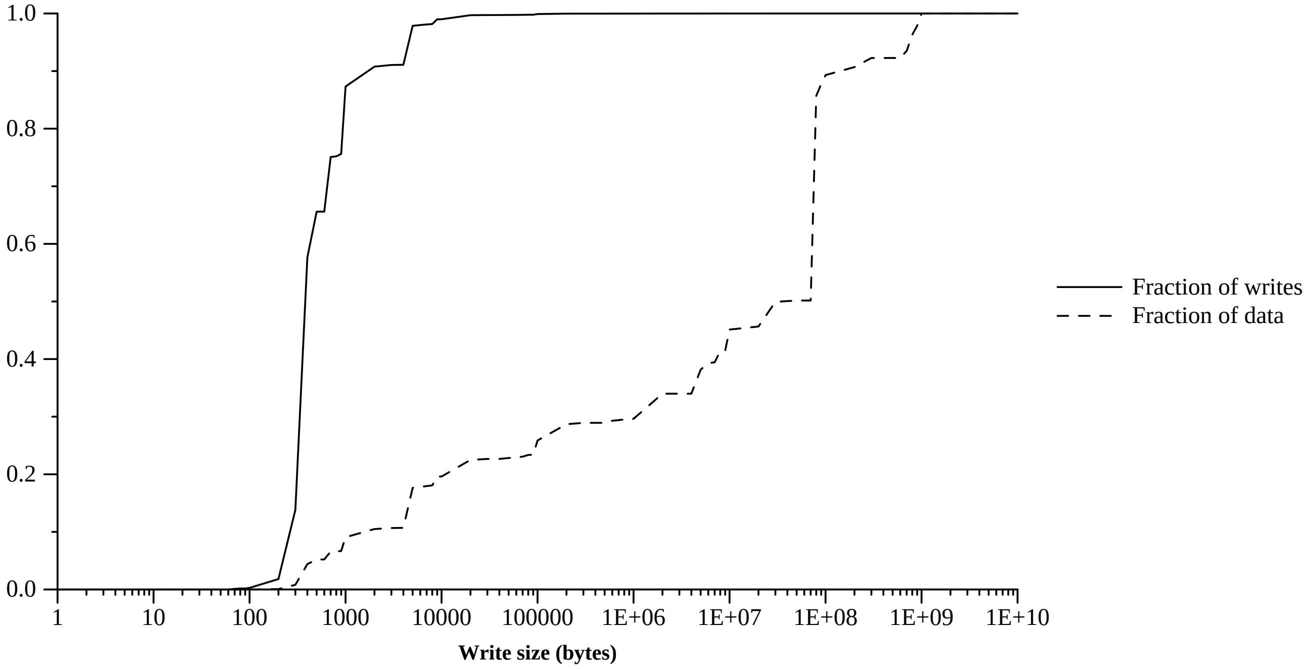

Figure 9: CDF of the number of writes by request size and amount of data transferred by requests from CMF jobs.

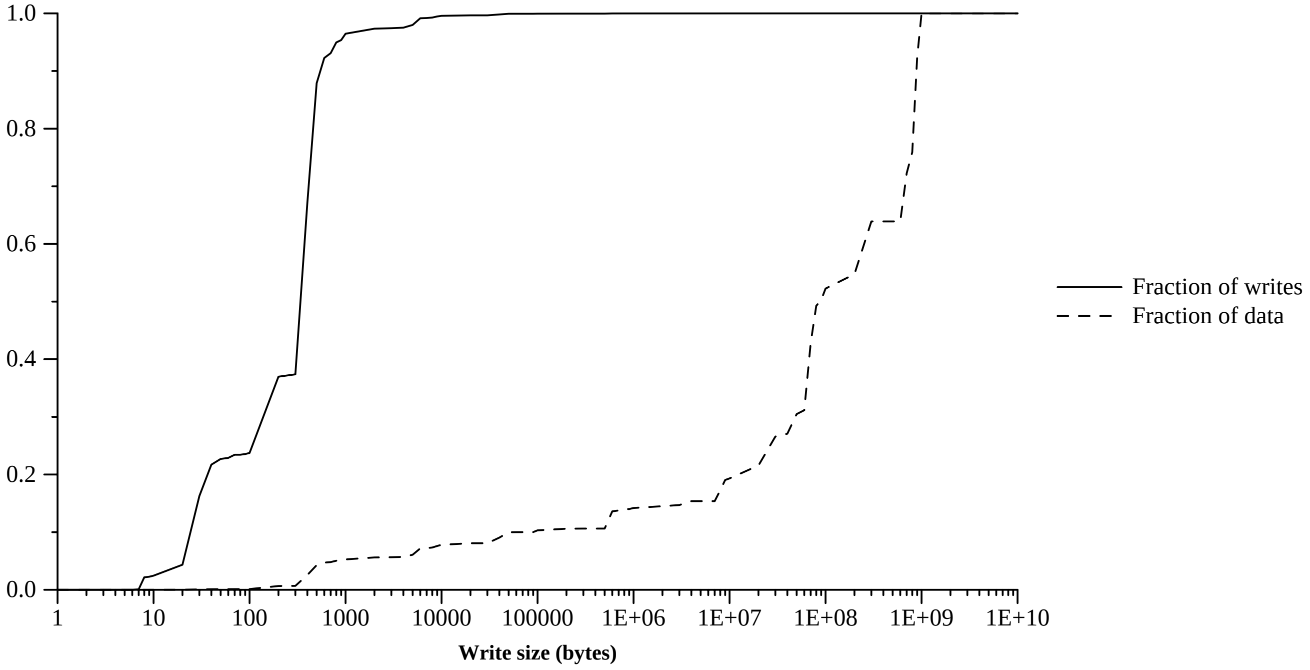

Figure 10: CDF of the number of writes by request size and amount of data transferred by requests from CMMD jobs.

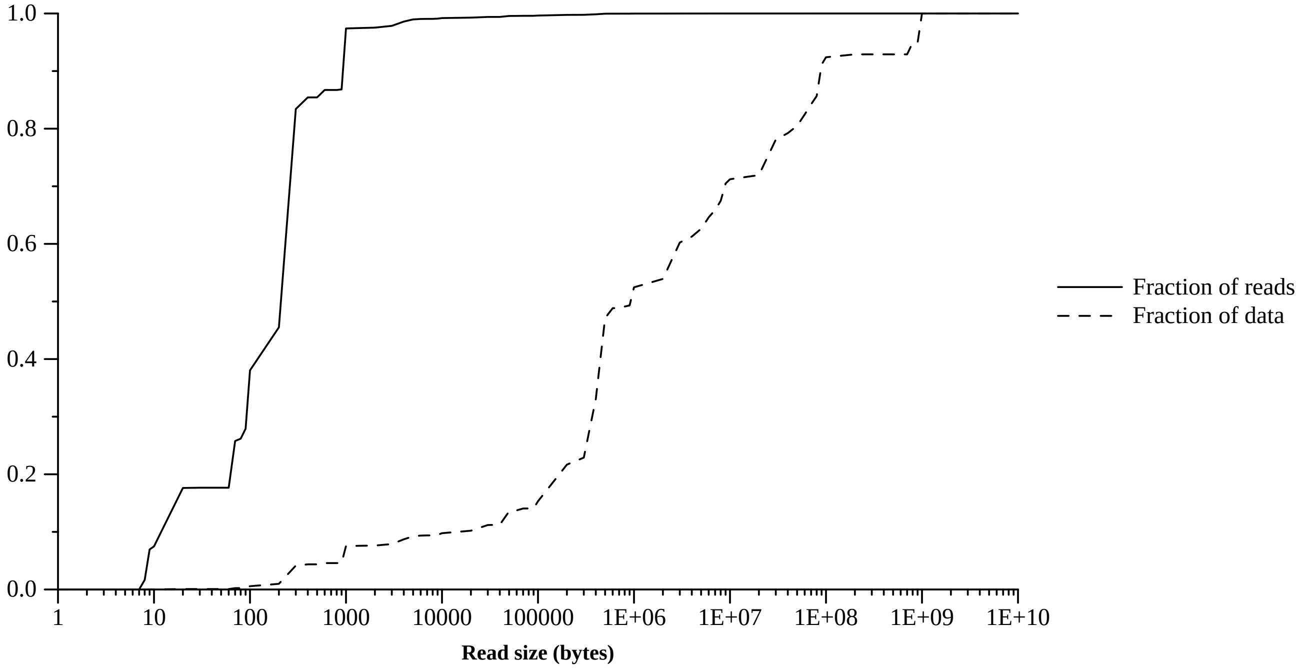

Figure 11: CDF of the number of reads by request size and amount of data transferred by requests from CMF jobs.

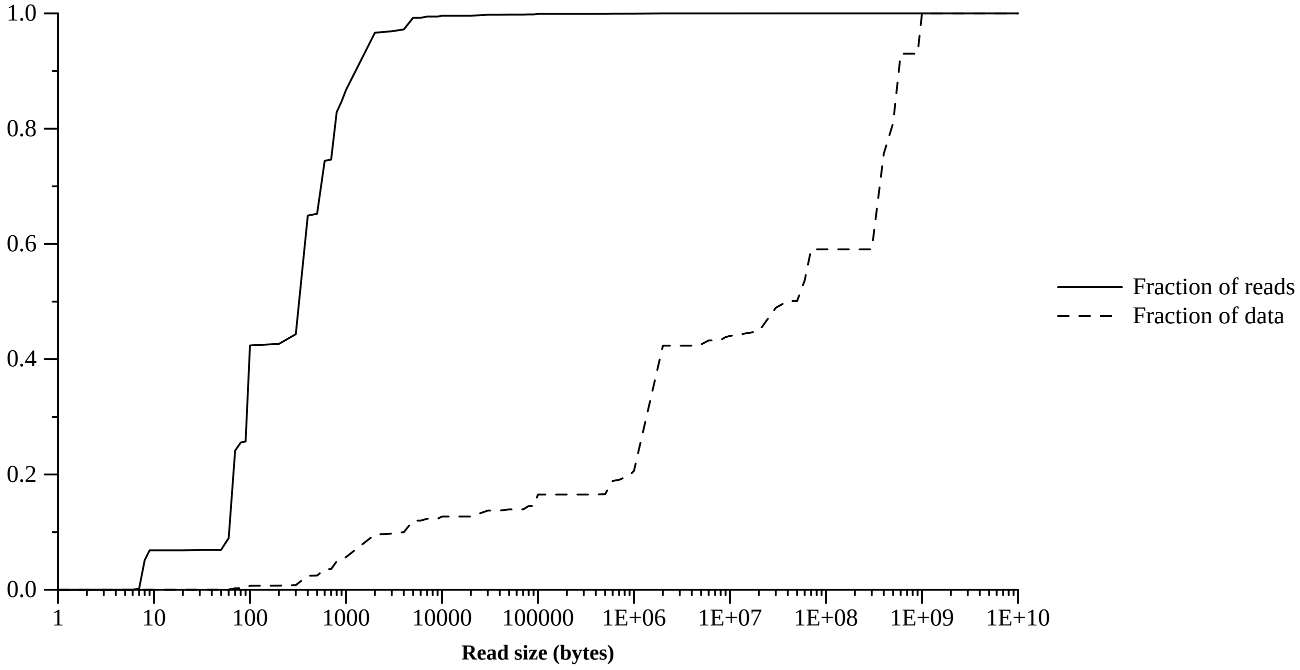

Figure 12: CDF of the number of reads by request size and amount of data transferred by requests from CMMD jobs.

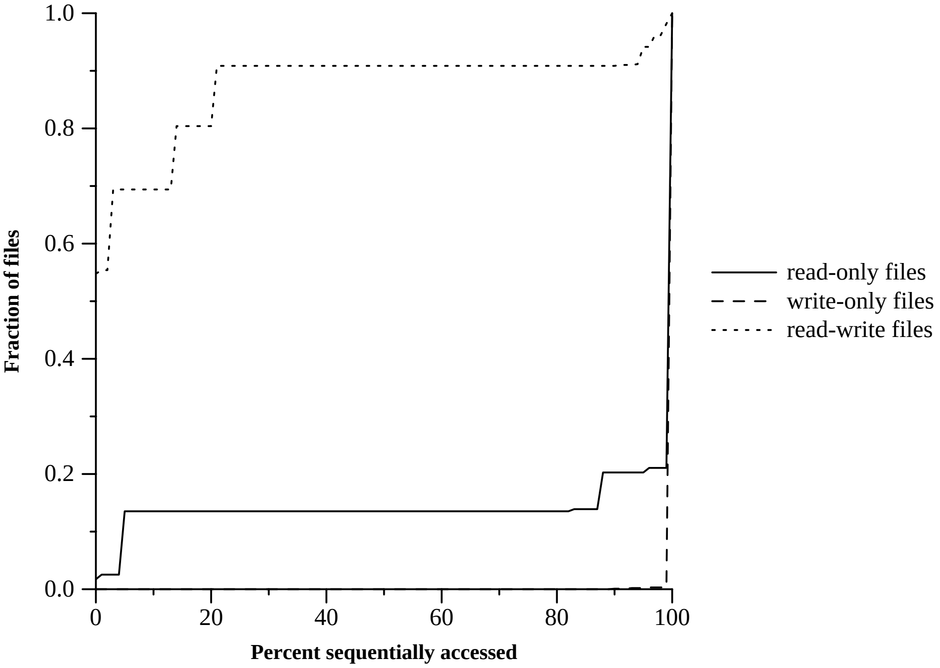

Figure 13: CDF of sequential access to files from CMF jobs.

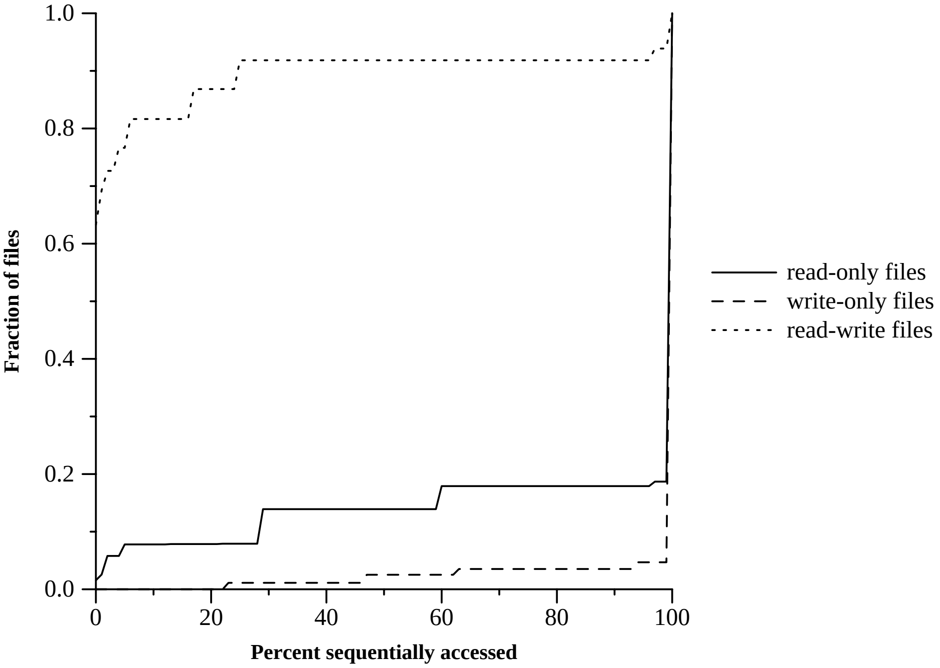

Figure 14: CDF of sequential access to files from CMMD jobs.

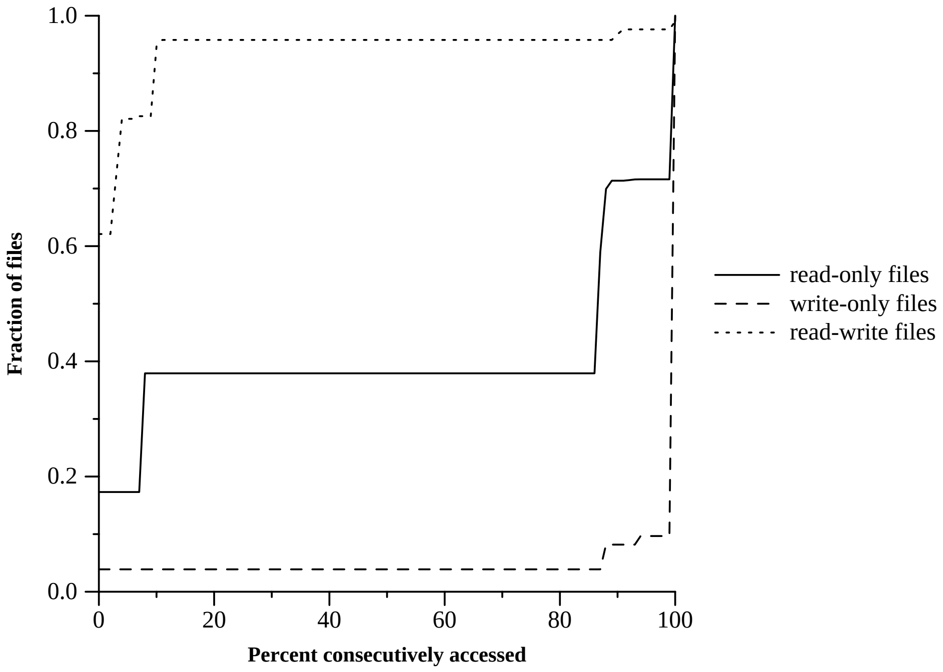

Figure 15: CDF of consecutive access to files from CMF jobs.

Number of different interval sizes	number of files	Percent of total
0	1243	32.9
1	1501	39.7
2	781	20.7
3	53	1.4
4+	202	5.3

Table 6: Number of different interval sizes used in each file accessed by CMF jobs.

Number of different interval sizes	number of files	Percent of total
0	303	33.5
1	310	34.3
2	143	15.8
3	64	7.1
4+	84	9.3

Table 7: Number of different interval sizes used in each file accessed by CMMD jobs.

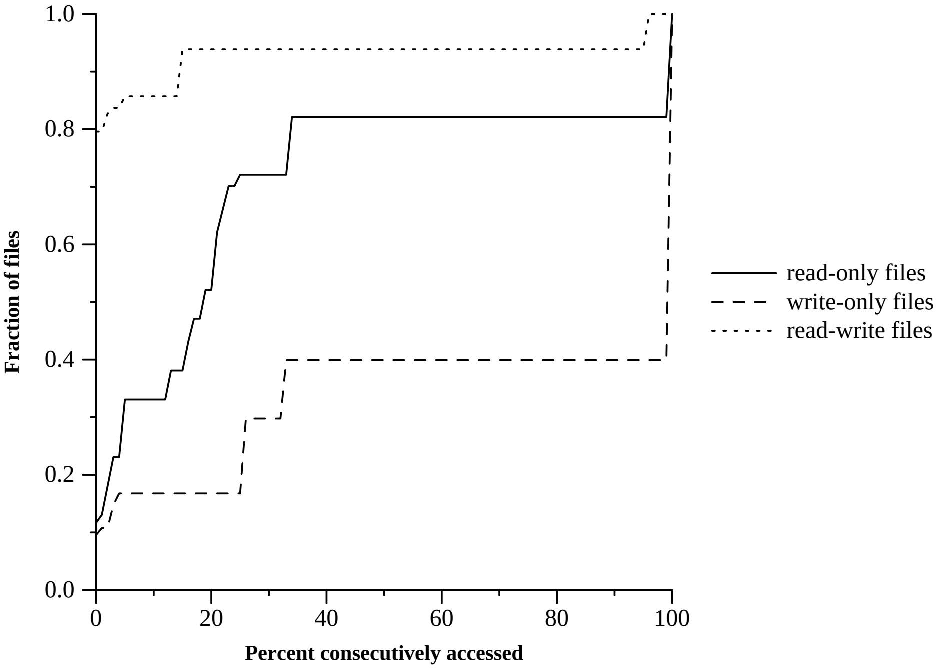

Figure 16: CDF of consecutive access to files from CMMD jobs.

Different request sizes	number of files	Percent of total
0	4	0.1
1	849	22.5
2	1124	29.7
3	1315	34.8
4+	488	12.9

Table 8: Number of different request sizes used in each file accessed by CMF jobs.

Different request sizes	number of files	Percent of total
0	2	0.2
1	343	37.9
2	319	35.4
3	83	9.2
4+	157	17.3

Table 9: Number of different request sizes used in each file accessed by CMMD jobs.

Percent	Read		Written		Both		Overall	
shared	byte	block	byte	block	byte	block	byte	block
= 0	29.43	10.78	95.45	88.58	63.01	1.37	71.27	57.28
$<$ 1	31.31	5.27	4.29	5.21	18.26	8.22	14.18	5.40
$<$ 50	12.12	12.43	0.26	4.20	8.22	18.72	4.71	7.80
$>$= 50	3.38	4.80	0.00	1.09	1.37	12.33	1.22	2.99
$>$ 99	0.00	2.44	0.00	0.92	0.91	0.91	0.05	1.43
= 100	23.76	64.28	0.00	0.00	8.23	58.45	8.47	25.00

Table 10: Byte and block sharing in files accessed by CMF jobs. The numbers in the 8 columns from the right indicate the percent of files that were *x-percent* byte or block shared, where x is the value in the leftmost column of the row corresponding to a particular number.

Percent	Read		Written		Both		Overall	
shared	byte	block	byte	block	byte	block	byte	block
= 0	10.51	0.00	93.46	39.77	34.69	0.00	66.48	26.22
$<$ 1	12.45	0.00	3.02	18.96	4.08	2.04	5.75	12.61
$<$ 50	10.12	5.45	1.17	23.32	4.08	6.12	3.87	17.26
$>$= 50	5.45	2.33	2.35	15.77	36.73	10.20	5.09	11.62
$>$ 99	0.39	1.17	0.00	1.34	0.00	2.04	0.11	1.33
= 100	61.09	91.05	0.00	0.84	20.41	79.59	18.47	30.75

Table 11: Byte and block sharing in files accessed by CMMD jobs. The numbers in the 8 columns from the right indicate the percent of files that were *x-percent* byte or block shared, where x is the value in the leftmost column of the row corresponding to a particular number.

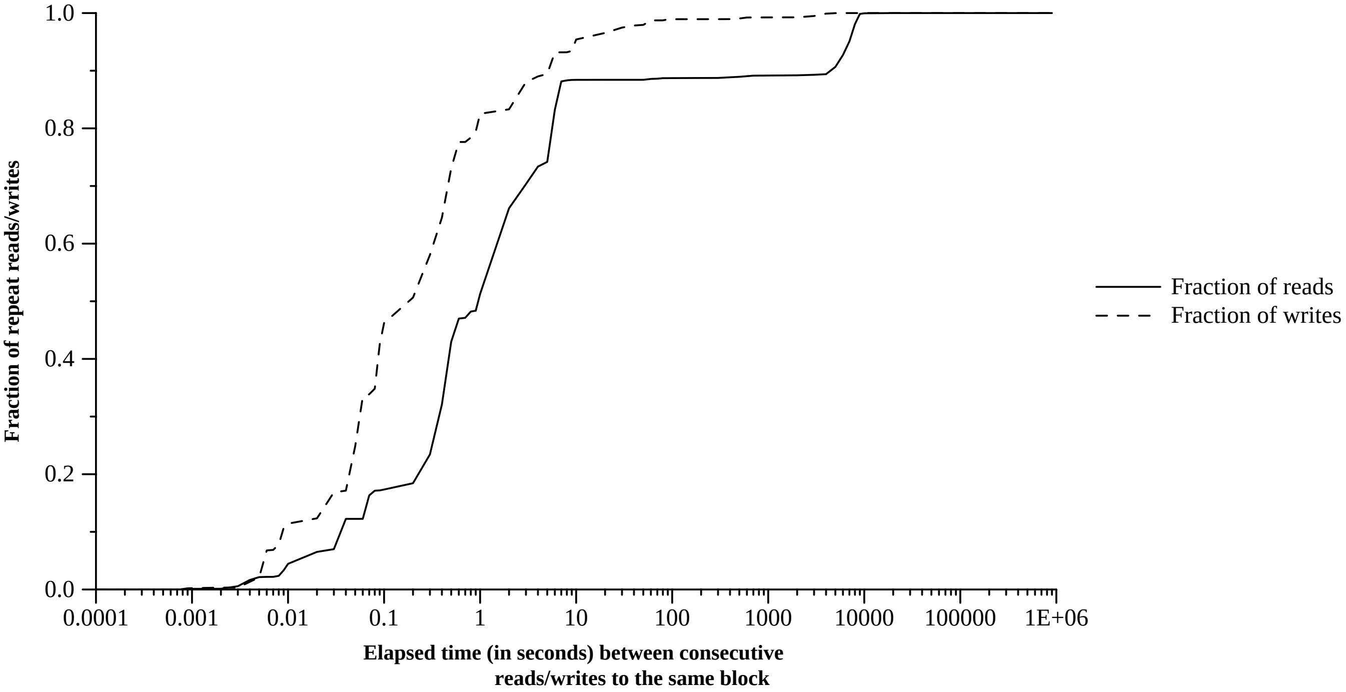

Figure 17: CDF of elapsed time between re-read and re-write of same blocks from CMF jobs.

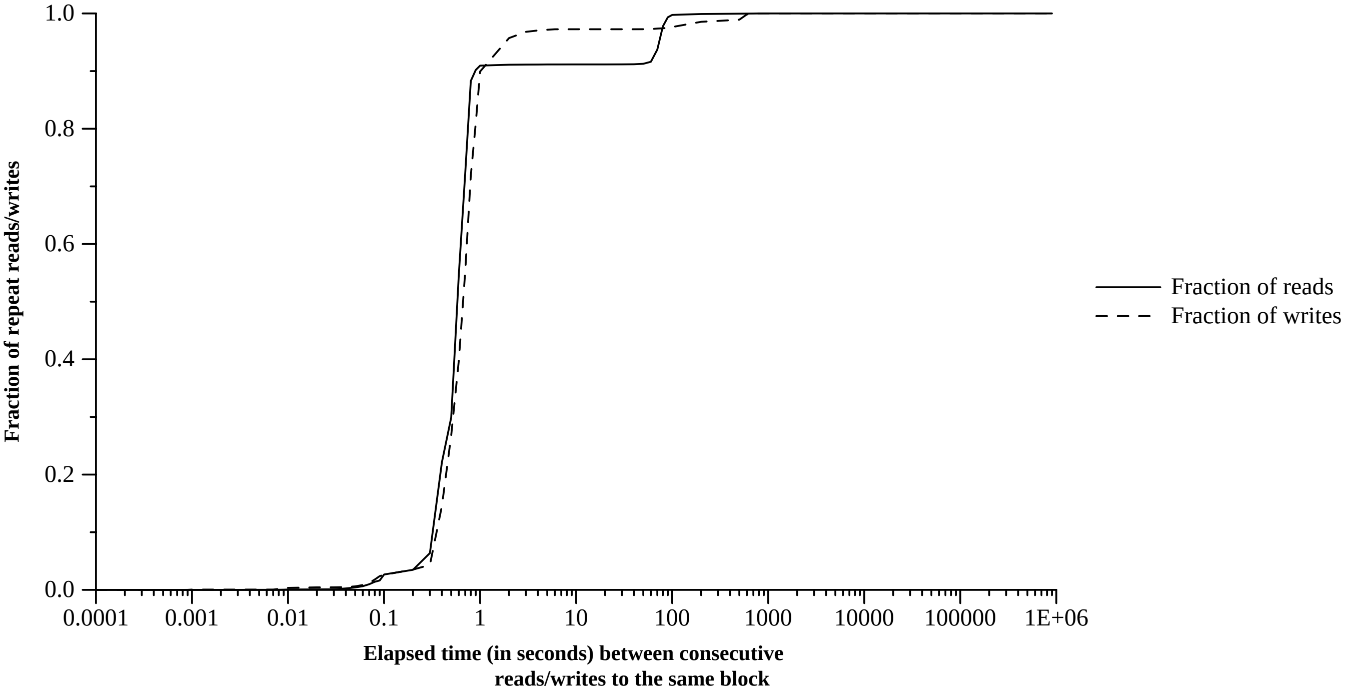

Figure 18: CDF of elapsed time between re-read and re-write of same blocks from CMMD jobs.

References

[BGST93] Michael L. Best, Adam Greenberg, Craig Stanfill, and Lewis W. Tucker. CMMD I/O: A parallel Unix I/O. In *Proceedings of the Seventh International Parallel Processing Symposium*, pages 489–495, 1993.

[BHK$^+$91] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a distributed file system. In *Proceedings of 13th ACM Symposium on Operating Systems Principl es*, pages 198–212. Association for Computing Machinery SIGOPS, October 1991.

[CFPB93] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, and Sandra Johnson Baylor. Parallel access to files in the Vesta file system. In *Proceedings of Supercomputing '93*, pages 472–481, 1993.

[CHKM93] 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.

[CK93] Thomas H. Cormen and David Kotz. Integrating theory and practice in parallel file systems. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 64–74, Hanover, NH, June 1993. Dartmouth Institute for Advanced Graduate Studies. Revised from Dartmouth PCS-TR93-188.

[Cro89] Thomas W. Crockett. File concepts for parallel I/O. In *Proceedings of Supercomputing '89*, pages 574–579, 1989.

[dC94] Juan Miguel del Rosario and Alok Choudhary. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, March 1994.

[DdR92] Erik DeBenedictis and Juan Miguel del Rosario. nCUBE parallel I/O software. In *Eleventh Annual IEEE International Phoenix Conference on Computers and Communications (IPCCC)*, pages 0117–0124, April 1992.

[Dib90] P. Dibble. *A Parallel Interleaved File System*. PhD thesis, University of Rochester, March 1990.

[FE89] 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.

[Flo86] R. Floyd. Short-term file reference patterns in a UNIX environment. Technical Report 177, Dept. of Computer Science, Univ. of Rochester, March 1986.

[FPD93] James C. French, Terrence W. Pratt, and Mriganka 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.

[GGL93] N. Galbreath, W. Gropp, and D. Levine. Applications-driven parallel I/O. In *Proceedings of Supercomputing '93*, pages 462–471, 1993.

[Ken92] Kendall Square Research. *KSR1 technology background*, January 1992.

[KN94] David Kotz and Nils Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of Supercomputing '94*, November 1994. To appear. Currently available as Dartmouth College technical report PCS-TR94-211.

[Kot93] David Kotz. Multiprocessor file system interfaces. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, 1993. To appear.

[KS93] Orran Krieger and Michael 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. Dartmouth Institute for Advanced Graduate Studies.

[MK91] E. L. Miller and R. H. Katz. Input/Output behavior of supercomputer applications. In *Proceedings of Supercomputing '91*, pages 567–576, November 1991.

[Nit92] Bill Nitzberg. Performance of the iPSC/860 Concurrent File System. Technical Report RND-92-020, NAS Systems Division, NASA Ames, December 1992.

[ODH+85] 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.

[Pie89] Paul Pierce. A concurrent file system for a highly parallel mass storage system. In *Fourth Conference on Hypercube Concurrent Computers and Applications*, pages 155–160, 1989.

[Por82] J. Porcar. File migration in distributed computer systems. Technical Report LBL-14763, Lawrence Berkeley Lab, July 1982.

[Pow77] M. L. Powell. The DEMOS file system. In *Proceedings of the Sixth ACM Symposium on Operating System Principles*, pages 33–42, November 1977.

[PP93] 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.

[RB90] A. Reddy and P. Banerjee. A study of I/O behavior or perfect benchmarks on a multiprocessor. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 312–321, 1990 1990.

[RBK92] K. K. Ramakrishnan, P. Biswas, and Ramakrishna Karedla. Analysis of file I/O traces in commercial computing environments. In *Proceedings of ACM SIGMETRICS and PERFORMANCE '92*, pages 78–90, 1992.

[Roy93] Paul J. Roy. Unix file access and caching in a multicomputer environment. In *Proceedings of the Usenix Mach III Symposium*, pages 21–37, 1993.

[Smi81] A.J. Smith. Analysis of long term file reference patterns and their applications to file migration algorithms. *IEEE Transactions on Software Engineering*, SE-7(4):403–417, July 1981.

[Thi93a] Thinking Machines Corporation. *CM-5 I/O System Programming Guide Version 7.2*, September 1993.

[Thi93b] Thinking Machines Corporation. *CM5 Techical Summary*, November 1993.

[Thi93c] Thinking Machines Corporation. *CMMD Reference Manual Version 3.0*, May 1993.