File System Workload Analysis For Large Scale Scientific Computing Applications

Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long
Storage Systems Research Center
University of California, Santa Cruz
Santa Cruz, CA 95064
{cyclonew, qxin, hongbo, sbrandt, elm, darrell}@cs.ucsc.edu
Tel +1 831-459-4458
Fax +1 831-459-4829

Tyce T. McLarty
Development Environment Group/Integrated Computing and Communications
Lawrence Livermore National Laboratory
Livermore, CA 94551
{tmclarty@llnl.gov}
Tel +1 925-424-6975
Fax +1 925-423-8719

Abstract

Parallel scientific applications require high-performance I/O support from underlying file systems. A comprehensive understanding of the expected workload is therefore essential for the design of high-performance parallel file systems. We re-examine the workload characteristics in parallel computing environments in the light of recent technology advances and new applications. We analyze application traces from a cluster with hundreds of nodes. On average, each application has only one or two typical request sizes. Large requests from several hundred kilobytes to several megabytes are very common. Although in some applications small requests account for more than 90% of all requests, almost all of the I/O data are transferred by large requests. All of these applications show bursty access patterns. More than 65% of write requests have inter-arrival times within one millisecond in most applications. By running the same benchmark on different file models, we also find that the write throughput of using an individual output file for each node exceeds that of using a shared file for all nodes by a factor of 5. This indicates that current file systems are not well optimized for file sharing.

1. Introduction

Parallel scientific applications impose great challenges on not only the computational speeds but also the data-transfer bandwidths and capacities of I/O subsystems. The U.S. Department of Energy Ac-

celerated Strategic Computing Initiative (ASCI) projects computers with 100 TeraFLOPS, I/O rates of 50–200 gigabytes/second, and storage system capacities of 0.5–20 PB in 2005. The projected computing and storage requirements are estimated to 400 TeraFLOPS, 80–500 gigabytes/second, and 3–20 PB in 2008 [2]. The observed widening disparity in the performance of I/O devices, processors, and communication links results in a growing imbalance between computational performance and the I/O subsystem performance. To reduce or even eliminate this growing I/O performance bottleneck, the design of high-performance parallel file systems needs to be improved to meet the I/O requirements of parallel scientific applications.

The success of file system designs comes from a comprehensive understanding of I/O workloads generated by targeted applications. In the early and middle 1990s, significant research efforts were focused on characterizing parallel I/O workload patterns and providing insights on parallel system designs [1, 4, 7, 14]. The following decade has witnessed significant improvements in computer hardware, including processors, memory, communication links, and I/O devices. At the same time, systems are scaling up to match the increasing demands of computing capability and storage capacity. This advance in technologies also enables new scientific applications. Together these changes motivate us to re-examine the characteristics of parallel I/O workloads a decade later.

In our research, we traced the system I/O activities under three typical parallel scientific applications: the benchmark *ior2* [6], a physics simulation, *f1*, running on 343 nodes, and another physics simulation, *m1*, running on 1620 nodes. We study both static file system and dynamic I/O workload characteristics. We use the results to address the following questions:

- What were the file sizes? How old were they?
- How many files were opened, read, and written? What were their sizes?
- How frequent were typical file system operations?
- How often did nodes send I/O requests? What were the request sizes?
- What forms of locality were there? How might caching be useful?
- Did nodes share data often? What were the file sharing patterns?
- How well did nodes utilize the I/O bandwidth?

The remainder of this paper is organized as follows: a brief overview of the related work is given in Section 2. We then describe the tracing methodology in Section 3 and present our results in Section 4. Finally, we conclude our paper in Section 5.

2. Related Work

The I/O subsystem has been a system performance bottleneck for a long time. In parallel scientific computing environments, the high I/O demands make the I/O bottleneck problem even more severe. Kotz and Jain [3] surveyed impacts of I/O bottlenecks in major areas of parallel and distributed systems and pointed out that I/O subsystem performance should be considered at all levels of system design.

Previous research showed that the I/O behavior of scientific applications is regular and predictable [7, 9]. Users have also made attempts to adjust access patterns to improve performance of parallel file systems [13].

There are several studies on file system workload characterizations in scientific environments [1, 4, 7, 8, 11]. They have shown that file access patterns share common properties such as large file sizes, sequential accesses, bursty program accesses, and strong file sharing among processes within a job. A more recent study [14] showed that applications use a combination of both sequential and interleaved access patterns and all I/O requests are channeled through a single node when applications require concurrent accesses; we observe similar phenomena in one of the applications under our examinations.

Pasquale and Polyzos [9] found that the data transfer rates ranges from 4.66 to 131 megabytes/sec in fifty long-running large-scale scientific applications. They also demonstrated that the the I/O request burstiness is periodic and regular [10].

Baylor and Wu [1] showed that the I/O request rate is on the order of hundreds of requests per second; this is similar to our results. They also found that a large majority of requests are on the order of kilobytes and a few requests are on the order of megabytes; our results differ in this regard.

Previous research has mainly investigated scientific workloads in the 1990's, although technology has evolved very quickly since then. We observed changes in large-scale scientific workloads, in our study, and provided guidelines for future file system designs based on a thorough understanding of current requirements of large-scale scientific computing.

3. Tracing Methodology

All the trace data in this study was collected from a large Linux cluster with more than 800 dual processor nodes at the Lawrence Livermore National Laboratory (LLNL). A development version of Lustre Lite [12] is employed as the parallel file system and the Linux kernel in use is a variant of 2.4.18.

3.1. Data Collection

Tracing I/O activities in large scale distributed file systems is challenging. One of the most critical issues is minimizing the disturbance of tracing on the system behaviors. A commonly-used method is to develop a trace module that intercepts specific I/O system calls—a dedicated node in the cluster collects all trace data and stores them to local disks.

However, due to time limits, we chose a simpler approach: we employed the *strace* utility with parameters tuned for tracing file-related system calls. The trace data are written to local files. We rely on the local host file systems to buffer trace data.

This approach has two shortcomings: first, strace intercepts all I/O-related activities, including parallel file system, local file system, and standard input/output activities. This results in relatively large data footprint. Second, the strace utility relies on the local file system to buffer traced data. This buffer scheme works poorly when the host file system is under heavy I/O workloads. In such a scenario, the host system performance might be affected by the frequent I/Os of the traced data.

However, the strace utility greatly simplifies the tedious data collection process to a simple shell script. More importantly, the shortcomings mentioned above were not significant in our trace col-

Table 1. The ASCI Linux Cluster Parameters

Total Nodes (IBM x355)	960
Compute Nodes	924
Login Nodes	2
Gateway Nodes	32
Metadata Server Nodes	2
Processor per Nodes (Pentium 4 Prestonia)	2
Total Number of Processors	1920
Processor Speed (GHz)	2.4
Theoretical Peak System Performance (TFlops)	9.2
Memory per Node (GB)	4
Total Memory (TB)	3.8
Total Local Disk Space (TB)	115
Nodes Interconnection	**Quadrics Switch**

lection because of the large I/O requests and the relatively short tracing periods. As we discuss in Section 4, I/O requests in such a large system are usually around several hundred kilobytes to several megabytes. Even in the most bursty I/O period, the total number of I/Os per node is still around tens of requests per second. Up to one hundred trace records will be generated on each node per second on average. Buffering and storing these data only has a slight impact on the system performance. Moreover, instead of tracing the whole cluster, we only study several typical scientific applications. Those applications are usually composed of two stages: the computation phase and the I/O phase. The typical I/O stage ranges from several minutes to several hours. During this period, each node usually generates several hundred kilobytes of trace data, which can be easily buffered in memory.

3.2. Applications and Traces

All of the trace data were collected from the ASCI Linux Cluster in Lawrence Livermore National Laboratory. This machine is currently in limited-access mode for science runs and file system testing. It has 960 dual-processor nodes connected through a Quadrics Switch. Two of the nodes are dedicated metadata servers and another 32 nodes are used as the gateways for accessing a global parallel file system. The detailed configuration of this machine is provided in table 1 [5]. We traced three typical parallel scientific applications during July, 2003. The total size of the traces is more than 800 megabytes.

The first application is a parallel file system benchmark, *ior2* [6], developed by LLNL. It is used for benchmarking parallel file systems using POSIX, MPIIO, or HDF5 interfaces. Basically it writes a large amount of data to one or more files and then reads them back to verify the correctness of the data. The data set is large enough to minimize the operating system caching effect. Based on different file usages, we collected three different benchmark traces, named *ior2-fileproc*, *ior2-shared*, and *ior2-stride*, respectively. All of them ran on a 512-node cluster. *ior2-fileproc* assigns an individual output file for each node, while *ior2-shared* and *ior2-stride* use a shared file for all the nodes. The difference between the last two traces is that *ior2-shared* allocates a contiguous region in the shared file for each node, while *ior2-stride* strides the blocks from different nodes into the shared file.

The second application is a physics simulation run on 343 processes. In this application, a single node gathers a large amount of data in small pieces from the others nodes. A small set of nodes then write these data to a shared file. Reads are executed from a single file independently by each node. This application has two I/O-intensive phases: the restart phase, in which read is dominant; and the result-dump phase, in which write is dominant. The corresponding traces are named *f1-restart* and *f1-write*, respectively.

The last application is another physics simulation which runs on 1620 nodes. This application use an individual output file for each node. Like the previous application, it also has a restart phase and a result-dump phase. The corresponding traces are referred as *m1-restart* and *m1-write*, respectively.

3.3. Analysis

The raw trace files required some processing before they could be easily analyzed. Some unrelated system calls and signals were filtered out. Since each node maintained its own trace records, the raw trace for each application is composed of hundreds of individual files. We merged those individual files in chronological order. Thanks to the Quadrics switch, which has a common clock, the traced time in those individual trace files was globally synchronized. Our analysis work, such as request inter-arrival time, have been greatly simplified by sorting all requests into a chronologically sorted trace file.

A good understanding of file metadata operation characteristics is important, however, our traces are not large enough to capture general metadata access patterns. Therefore, we focus more on file data I/O characterization in the following section.

4. Workload Characteristics

We present the characteristics of the workloads, including file distributions and I/O request properties. We study the distributions of file size and lifetimes and show the uniqueness of large-scale scientific workloads. We focus on three typical applications as described in Section 3.2 and examine the characteristics of I/O requests, such as the size and number of read and write requests and the burst and the distribution of I/O requests on various nodes.

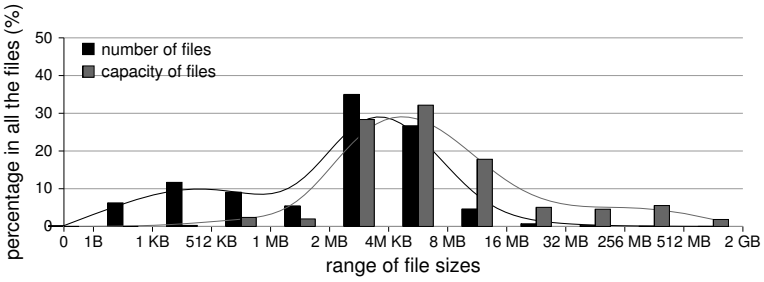
4.1. File Distributions

We collected file distributions from thirty-two file servers that were in use for the ASCI Linux cluster during the science runs phase. Each file server has storage capacity of 1.4 terabytes. The file servers were dedicated to a small number of large-scale scientific applications, which provides a good model of data storage patterns. On average, the number of files on each file server was 350,250, and each server stored 1.04 terabytes of data, more than 70% of their capacity. On most of the file servers, the number and capacity of files are similar except for five file servers. Table 2 displays statistic values of the number and capacity of files on these servers, including mean, standard deviation (std. dev.), median, minimum (min) and maximum (max).

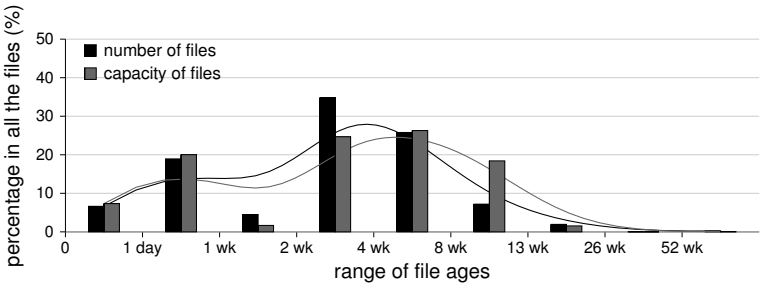
Figure 1(a) presents file size distributions by number and file capacity. The ranges of file sizes are sampled from 0–1 Byte to 1–2 Gigabytes. Some of the partitions were merged due to space limitations. We observed that over 80% of the files are between 512 kilobytes and 16 megabytes in

Table 2. File Numbers and Capacity of the 32 File Servers

	Number	Capacity
mean	305,200	1044.33 GB
standard deviation	75,760	139.66 GB
median	305,680	1072.88 GB
minimum	67,276	557.39 GB
maximum	605,230	1207.37 GB



(a) By File Sizes



(b) By File Ages

Figure 1. Distribution of Files

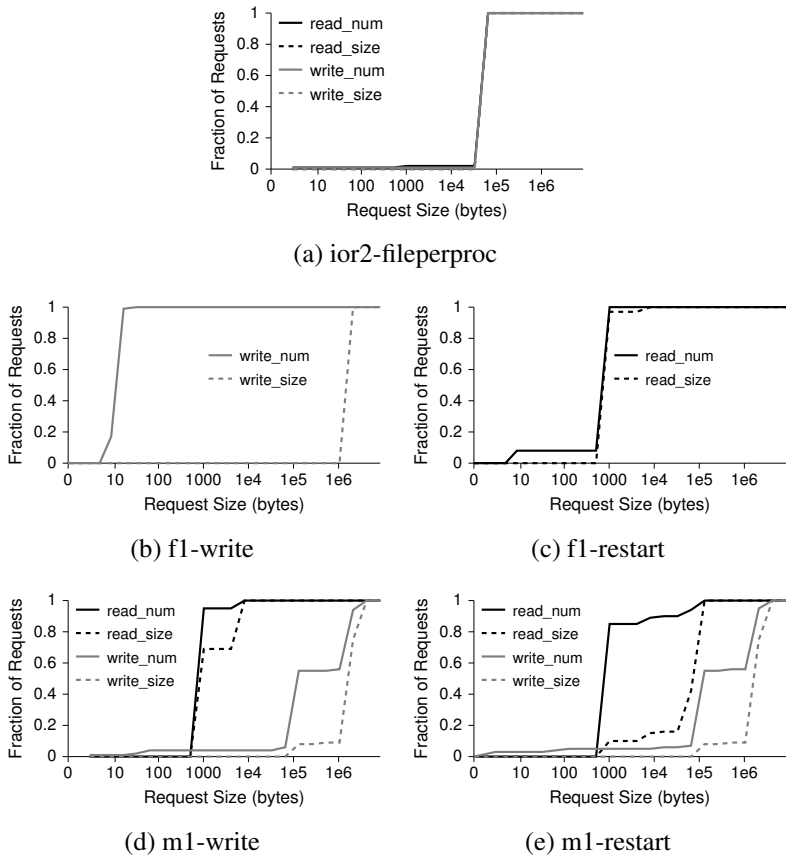
size and these files accounted for over 80% of the total capacity. Among various file size ranges, the most noticeable one is from 2 megabytes to 8 megabytes: about 61.7% of all files and 60.5% of all bytes are in this range.

We divided file lifetimes into 9 categories: from 0–1 day to 52 weeks and older. As illustrated in figure 1(b), 60% of the files and 50% of the bytes lived from 2 weeks to 8 weeks, while 6.6% of the files and 7.3% of the bytes lived less than one day. The lifetime of the traced system is about 1 year so no files lived longer than 52 weeks.

4.2. I/O Request Sizes

Figure 2 shows the cumulative distribution function of request sizes and request numbers. Since all three *ior2* benchmarks have identical request size distributions, we only show one of them. As shown in Figure 2(a), *ior2* has only an unique request size of around 64 kilobytes.

Figure 2(b) shows the write request size distribution of the result-dump stage in the physics simulation, *f1*. Almost all the write requests are smaller than 16 bytes, while almost all the I/O data are



(a) ior2-fileperproc

(b) f1-write

(c) f1-restart

(d) m1-write

(e) m1-restart

Figure 2. Cumulative Distribution Functions (CDF) of the Size and the Number of I/O Requests (X axis-logscale). The *read_num* and *write_num* curves indicate the fraction of all requests that is smaller than the size given in X axis. The *read_size* and *write_size* curves indicate the fraction of all transferred data that live in requests with size smaller than the value given in the X axis.

transferred in the requests with sizes larger than one megabyte. This turns out to be a common I/O pattern of scientific applications: a master node collects small pieces of data from all computing nodes and writes them to data files, which results in a huge number of small writes. Other nodes read and write these data files in very large chunks then. There are so few read requests in the result-dump stage and write requests in the restart stage that we actually ignore the read request curves in figure 2(b) and the write request curves in figure 2(c).

Figure 2(d) and figure 2(e) show the same write request distribution in the restart and result-dump stages of the physics simulation, *m1*. The two spikes in the *write_num* curves indicate two major write sizes: 64 kilobytes and 1.75 megabytes, respectively. Each of them accounts for 50% of all write requests. More than 95% of the data are transfered by large requests, which is also shown in Figures 2(d) and 2(e). Reads in *m1* are dominated by small requests less than 1 kilobytes. However, a small faction (less than 3%) of 8 kilobyte requests still accounts for 30% of all read data transfer. This is similar to the read distribution in Figure 2(e): only 5% of the read requests contribute to 90% of all data read.

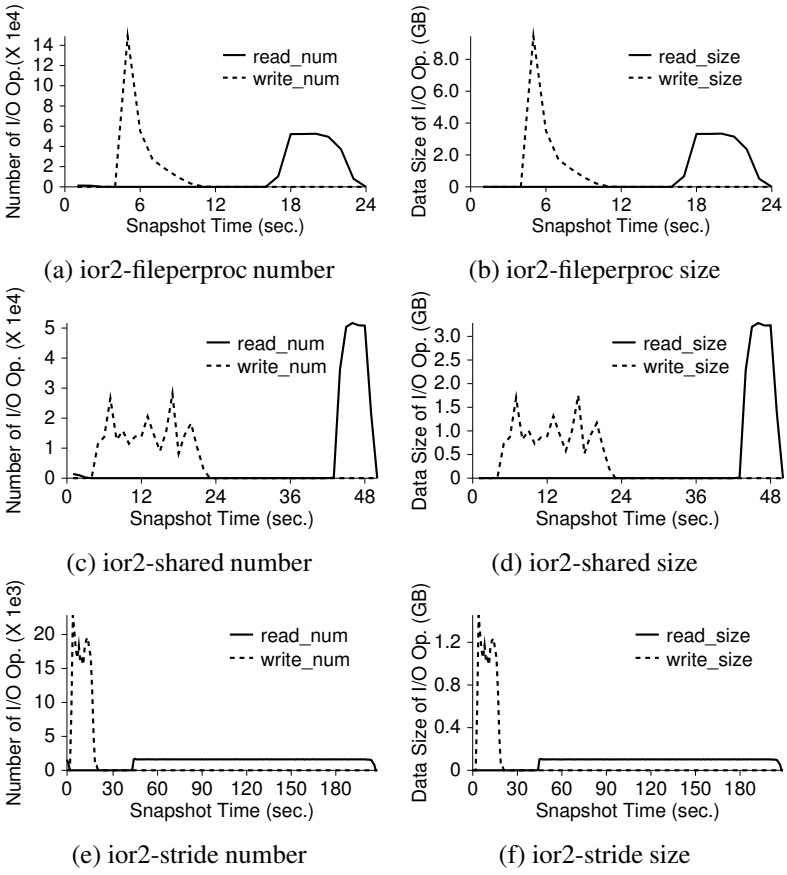
4.3. I/O Accesses Characteristics

Figure 3–5 show I/O accesses characteristics over time. The resolution for these figures is 1 second except figure 4(a), which uses a resolution of 50 seconds. Figure 3 shows that the request number distribution and the request size distribution are almost identical in *ior2* due to the fixed size requests used in those benchmarks. The *ior2-fileproc* benchmark, using the one-file-per-node model, presents the best write performance. Up to 150,000 write requests per second, totaling 9 gigabytes per second, are generated by the 512 nodes. However, the *ior2-shared* and *ior2-stride* benchmarks can only achieve 25,000 write requests per second, totaling 2 gigabytes per second. These two benchmarks use the shared-region and the shared-stride file model, respectively. We believe that the performance degradation is caused by the underlying file consistency protocol. This result is somewhat counterintuitive. The shared-region file model appears to be similar to the one-file-per-node model because the contiguous regions in the former can be analogous to the separate files in the latter. Therefore, their performance should be comparable as well. The severe performance degradation implies that the shared-file model is not optimized for this scenario.

After a write, each node reads back another node's data as soon as it is available. The gaps between the write and read curves in each sub-figure reflect the actual I/O times. Obviously, the *ior2-fileproc* benchmark demonstrates much better performance: only 10 seconds are used in this model, while more than 20 seconds are needed to dump the same amount of data when using the shared file model. Since reads must be synchronous, we can easily figure out the file system read bandwidth from the *read_size* curve. The *ior2-fileproc* and *ior2-shared* benchmarks have comparable read performance. However, the *ior2-stride* has the worst read performance, which is only 100 megabytes per second for 512 nodes. This result is not surprising: the stride data layout in shared files limits the chances of large sequential reads.

Figure 4 shows the I/O access pattern of the application *f1*. As we mentioned before, *f1-write* has very few reads and *f1-restart* has very few writes. Therefore, we can ignore those requests in the corresponding figures. In Figure 4(a), we chose a resolution of 50 seconds because it becomes unreadable if we use finer time resolutions. The spike of the *write-num* curve is caused by the activities of the master node to collect small pieces of data from other computing nodes. At its peak time, nearly 1 million file system requests are issued per second. However, due to the very small request size (8 to 16 bytes), this intensive write phase contributes negligable amounts of data to the overall data size. In the rest of the application, large write requests from 48 nodes dominate the I/O activities. Requests are issued in a very bursty manner. Figure 4(b) zooms in a small region of Figure 4(a) by 1 second resolution. It shows that sharp activity spikes are separated by long idle periods. At the peak time, up to 120 megabytes per second of data are generated by 48 nodes. In the restart phase of *f1*, read requests become dominant. However, both the number and the data size of read requests are small compared to those in the write phase.

Figure 5 presents the I/O access pattern of the physics application *m1*. It demonstrates very good read performance: nearly 28 gigabytes per second bandwidth can be achieved by 1620 nodes, thanks to the large read size (1.6 megabytes – 16 megabytes). Like *f1*, its write activities are also bursty. We observed that the write curves have similar shapes in figure 5. They all begin with a sharp spike followed by several less intensive spikes. One possible explanation is that the file system buffer cache absorbs the coming write requests at the begin of the writes. However, as soon as the buffer is filled up, the I/O rate drops to what can be served by the persistent storage.

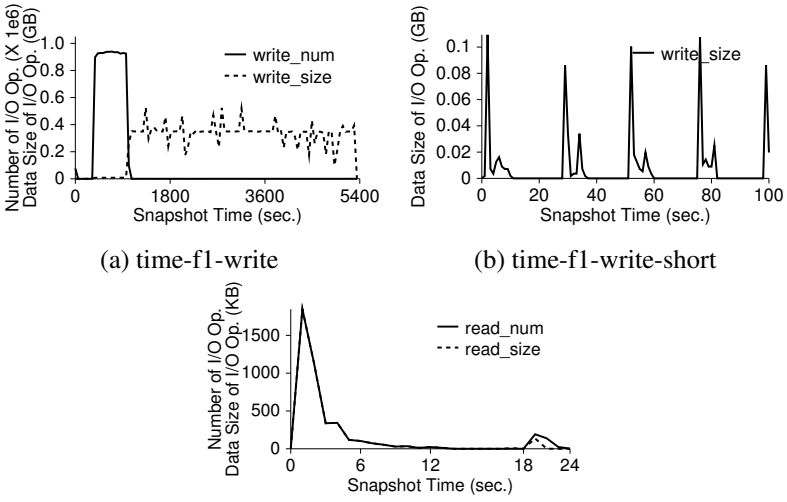


(a) ior2-fileperproc number (b) ior2-fileperproc size

(c) ior2-shared number (d) ior2-shared size

(e) ior2-stride number (f) ior2-stride size

Figure 3. I/O Requests over Time for *ior2* Benchmarks



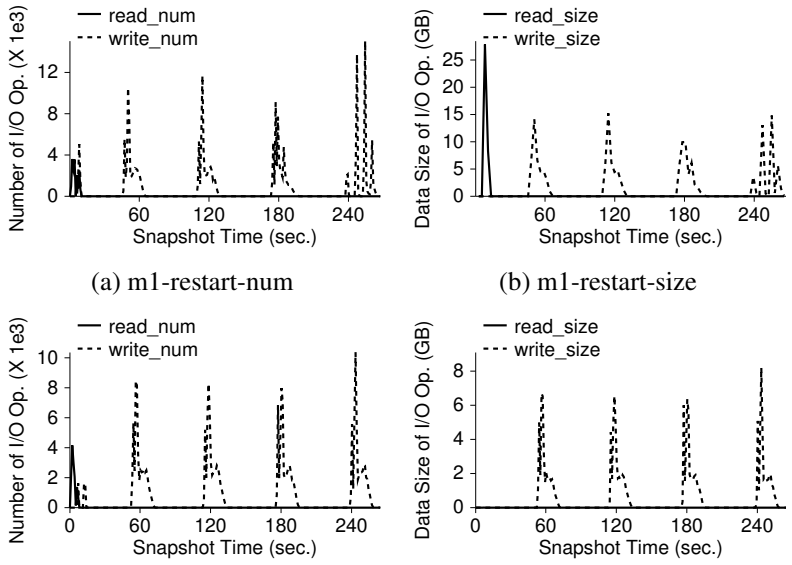
(a) time-f1-write (b) time-f1-write-short

(c) time-f1-restart

Figure 4. I/O Requests over Time for *f1* Application

4.4. I/O Burstiness

To study I/O burstiness, we measure I/O request inter-arrival times. Figure 6 shows the cumulative distribution functions (CDF) of I/O request inter-arrival times. Note that the x-axis is in the logarithmic scale. Write activities are very bursty in the *ior2* benchmarks and the *f1* application: over



(a) m1-restart-num (b) m1-restart-size

(c) m1-write-num (d) m1-write-size

Figure 5. I/O Requests over Time for *m1* Application

65–100% of write requests have inter-arrival times within 1 millisecond. In *ior2* and *f1*, most of the write activities are due to memory dump and I/O nodes can issue write requests quickly. However, write activities on *m1* are less intensive than those on *ior2* and *f1*

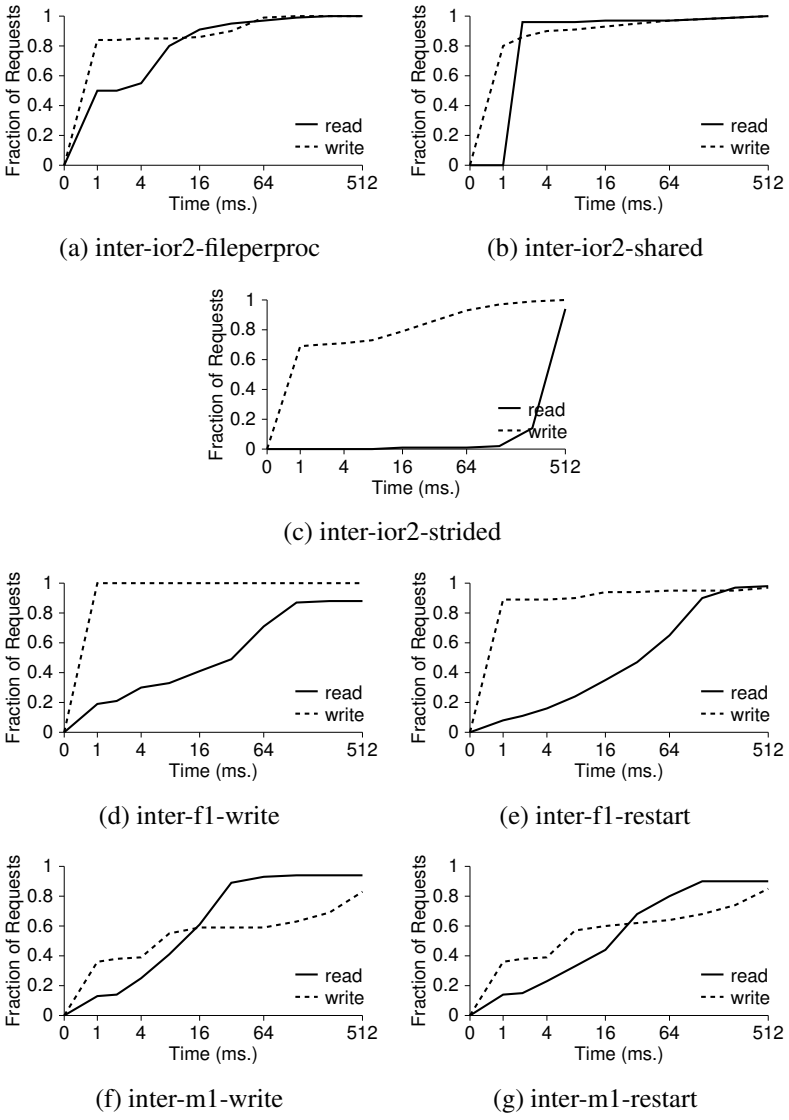
On the other hand, read requests are generally less intensive than write requests because reads are synchronous. In particular, Figure 6(c) indicates that *ior2* under shared-strided files suffers low read performance, as described in Section 4.3. In this scenario, data are interleaved in the shared file and read accesses are not sequential.

4.5. I/O Nodes

In this section, we study the distributions of I/O request sizes and numbers over nodes, as shown in Figure 7. For the *ior2* benchmarks, read and writes are distributed evenly among nodes, as shown in Figures 7(a) and 7(b), because each node executes the same sequence of operations in these benchmarks.

In the physics application *f1*, a small set of nodes write gathered simulated data to a shared file. Therefore, only a few nodes have significant I/O activity in their write phase and most of the transfered data are from large write requests (14% of the write requests), as shown in Figures 7(c) and 7(d). There is little read activity in the write phase. However, read requests are evenly distributed among nodes in the restart phase and their sizes are around 1 megabyte, as shown in Figures 7(e) and 7(f). There is little write activity in the restart phase.

In the restart and write phases of the physics application *m1*, I/O activity is well balanced among nodes, as shown in Figures 7(g)–7(j). We also observe significant write activity in the restart phase.



(a) inter-ior2-fileperproc (b) inter-ior2-shared

(c) inter-ior2-strided

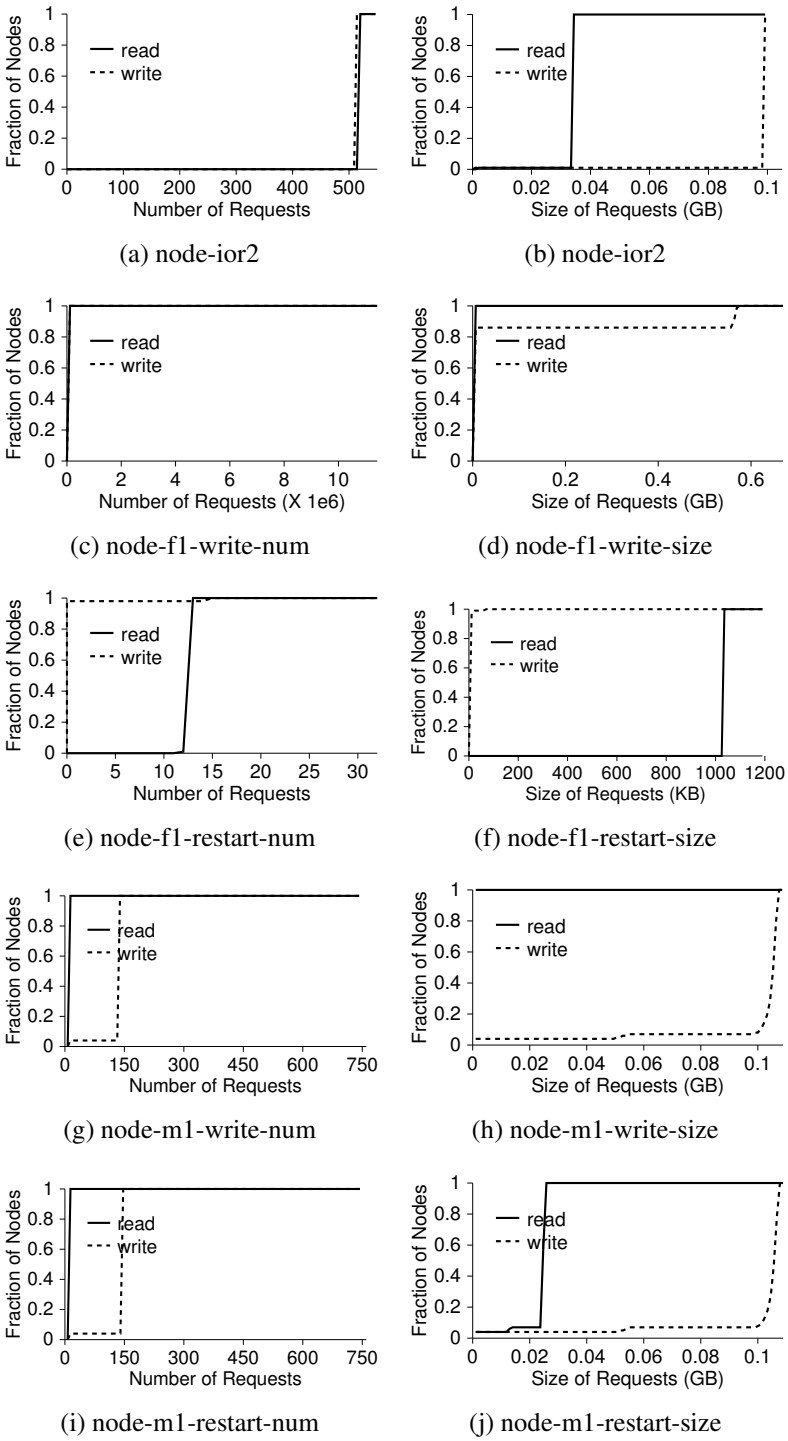
(d) inter-f1-write (e) inter-f1-restart

(f) inter-m1-write (g) inter-m1-restart

Figure 6. Cumulative Distribution Functions (CDF) of Inter-arrival Time of I/O Requests (X axis-logscale)

Table 3. File Open Statistics

Applicatons	Overall Number of File Opens			Number of Data File Opens		
	Read/Write	Read	Write	Read/Write	Read	Write
ior2	6,656	5,121	0	1,024	0	0
f1-write	3,871	6,870	718	98	10	34
f1-restart	3,773	6,179	0	0	343	0
m1-restart	17,824	22,681	12,940	0	1,620	12,960
m1-write	17,824	21,061	12,960	0	0	12,960



(a) node-ior2 (b) node-ior2

(c) node-f1-write-num (d) node-f1-write-size

(e) node-f1-restart-num (f) node-f1-restart-size

(g) node-m1-write-num (h) node-m1-write-size

(i) node-m1-restart-num (j) node-m1-restart-size

Figure 7. Cumulative Distribution Functions (CDF) of the Size of I/O Requests over Nodes

Table 4. Operations During File Open

Applications	Avg. open time		Avg. IOs per Open		Avg. IO Size per Open	
	Overall	Data File	Overall	Data File	Overall	Data File
ior2-fileproc	0.4 sec	4.5 sec	44.4	512.0	2.8 MB	32.8 MB
ior2-shared	0.7 sec	5.2 sec	44.4	512.0	2.8 MB	32.8 MB
ior2-stride	7.6 sec	26.57 sec	44.4	512.0	2.8 MB	32.8 MB
f1-write	20.2 sec	504.9 sec	14.8	142161	2.4 MB	3993.5 MB
f1-restart	0.02 sec	0.1 sec	0.5	1	<< 1 MB	<< 1 MB
m1-restart	1.2 sec	3.9 sec	4.2	15.3	3.7 MB	8.5 MB
m1-write	1.2 sec	2.4 sec	4.3	17	3.1 MB	6.5 MB

4.6. File Opens

In this section, we study the file open patterns of the applications. We use the term *data files* to refer to those files that actually store results dumped from applications.

In all applications, files tend to be opened as read/write or read-only. We only observe significant write-only files in the physics application *m1*, as shown in table 3. However, the data files are opened either read-only or write-only except for the benchmark *ior2*. The open operations on the data files only account for a small portion of the overall files opened. Given the fact that the data file operations dominate the overall I/O, the small number of data file opens implies longer open time and more I/O operations during each open. As listed in table 4, the open duration of data files ranges from several seconds to several hundred seconds, which is typically 2 to 20 times longer than overall file open durations. The average number of operations and the size of data files on each open operation are also much larger than those on the overall files. For example, up to 400 MB data are transferred during each data file open in physical application *f1-write*.

5. Conclusion

In this study, we analyze application traces from a cluster with hundreds of processing nodes. On average, each application has only one or two typical request sizes. Large requests from several hundred kilobytes to several megabytes are very common. Although in some applications, small requests account for more than 90% of all requests, almost all of the I/O data are transferred by large requests. All of these applications show bursty access patterns. More than 65% of write requests have inter-arrival times within one millisecond in most applications. By running the same benchmark on different file models, we also find that the write throughput of using an individual output file for each node exceeds that of using a shared file for all nodes by a factor of 5. This indicates that current file systems are not well optimized for file sharing. In all those applications, almost all I/Os are performed on a small set of files containing the intermediate or final computation results. Such files tend to be opened for a relatively long time, from several seconds to several hundred seconds. And a large amount of data are transferred during each open.

Acknowledgments

Feng Wang, Qin Xin, Scott Brandt, Ethan Miller, Darrell Long were supported in part by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714. Bo Hong was supported in part by the National Science Foundation under grant number CCR-073509. Tyce McLarty's effort was under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-Eng-48. This document was reviewed and released as unclassified with unlimited distribution as LLNL-UCRL-CONF-201895.

We are also grateful to our sponsors: National Science Foundation, USENIX Association, Hewlett Packard Laboratories, IBM Research, Intel Corporation, Microsoft Research, ONStor, Overland Storage, and Veritas.

References

[1] S. J. Baylor and C. E. Wu. Parallel I/O workload characteristics using Vesta. In *Proceedings of the IPPS '95 Workshop on Input/Output in Parallel and Distributed Systems (IOPADS '95)*, pages 16–29, Apr. 1995.

[2] DOE National Nuclear Security Administration and the DOE National Security Agency. Proposed statement of work: SGS file system, Apr. 2001.

[3] D. Kotz and R. Jain. I/O in parallel and distributed systems. In A. Kent and J. G. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 40, pages 141–154. Marcel Dekker, Inc., 1999. Supplement 25.

[4] D. F. Kotz and N. Nieuwejaar. File-system workload on a scientific multiprocessor. *IEEE Parallel and Distributed Technology*, 3(1):51–60, 1995.

[5] Lawrence Livermore National Laboratory. ASCI linux cluster. http://www.llnl.gov/linux/alc/, 2003.

[6] Lawrence Livermore National Laboratory. IOR software. http://www.llnl.gov/icc/lc/siop/downloads/download.html, 2003.

[7] E. L. Miller and R. H. Katz. Input/output behavior of supercomputing applications. In *Proceedings of Supercomputing '91*, pages 567–576, Nov. 1991.

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

[9] 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, Portland, OR, 1993. IEEE.

[10] B. K. Pasquale and G. C. Polyzos. Dynamic I/O characterization of I/O-intensive scientific applications. In *Proceedings of Supercomputing '94*, pages 660–669. IEEE, 1994.

[11] A. Purakayastha, C. S. Ellis, D. Kotz, N. Nieuwejaar, and M. Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. In *Proceedings of the 9th International Parallel Processing Symposium (IPPS '95)*, pages 165–172. IEEE Computer Society Press, 1995.

[12] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, July 2003.

[13] E. Smirni, R. A. Aydt, A. A. Chien, and D. A. Reed. I/O requirements of scientific applications: An evolutionary view. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 49–59. IEEE, 1996.

[14] E. Smirni and D. Reed. Lessons from characterizing the input/output behavior of parallel scientific applications. *Performance Evaluation: An International Journal*, 33(1):27–44, June 1998.