I/O Characterization on a Parallel File System

Sumit Narayan and John A. Chandy
Department of Electrical and Computer Engineering, University of Connecticut
371 Fairfield Way U-2157, Storrs, CT 06269-2157
Email: {sumit.narayan,john.chandy}@uconn.edu
tel +1 (860) 486-5047, fax +1 (860) 486-2447

***Abstract*—In this paper we present a study of I/O access patterns of scientific and general applications on a parallel file system. Understanding I/O access patterns is an essential condition to effectively designing a file system. Supercomputing applications running on these parallel systems make extensive use of parallel file systems taking advantage of faster data access by requesting information from multiple nodes simultaneously. However, parallel file systems can become a bottleneck if the file distribution parameters do not fit the access scheme of the applications. In our work, we examine a variety of such applications, providing measurement of inter-arrival times, I/O request size and burstiness demanded from a parallel file system. Our tests were conducted on the open source PVFS parallel file system with different configurations of metadata servers and I/O nodes. Among the findings are that the standard assumption of Poisson or random interarrival times is not justified and that access sizes are smaller than would be expected for a parallel application.**

I. Introduction

Rapidly increasing processor speeds and the ease with which they can be connected and converted to form a clustered computer environment has given rise to a new world of parallel computing. These parallel machines run applications which operate on the principle that large problems can always be divided into several smaller ones, and then each smaller problem can be solved concurrently.

However, disk I/O on these parallel machines continues to be a challenge. According to Amdahl's system balance rule [1], each MIPS (million instructions per second) must be accompanied by one megabit per second of I/O. However, today's disks are not able to meet this rule especially in light of the modern cluster computing or supercomputing capabilities. To overcome this, data is typically striped across multiple disks or storage nodes and applications take advantage of this by requesting data from different disks simultaneously. Parallel file systems such as GPFS [2], Lustre [3] and PVFS [4], have been designed to handle data in a clustered environment. In these file systems, striping refers to distributing file data blockwise (or objectwise) over multiple I/O devices or nodes. The application can then concurrently access the data through different connections. Scientific computing often requires large applications doing several noncontiguous access of small regions of data [5–8]. However, the frequency and size of request varies across applications. Thus, a sophisticated design of the parallel file system is required to minimize serialization of requests and facilitate parallelism in client accesses.

Some of the proposed schemes to improve the I/O delays include caching, prefetching and delayed writes. For example, in Sprite [9], data is held in cache for 30 to 60 seconds before being written to disk. This allows the short life temporary files which exist for less than 30 seconds to be deleted before being written to the disk. The distributed file system thus never sees the file. It further helps the operating system to combine write requests into a single larger request. These characteristics, however, are application dependent and require changes made to the file system to reflect differences between smaller and larger applications. Prefetching data into a cache reduces demand for immediate disk I/Os, but with increasing volume of input data, they become another parameter dependent on the application, thus requiring changes made to the file system before being used.

Nevertheless, due to varying computational and data access patterns of different applications, it is essential that file system designers understand how an application accesses data. The standard approach of striping data in a parallel file system can become a bottleneck if the file distribution parameters do not fit the actual access patterns of the applications. Workload characterization, thus, plays an important role in systems design. It is one of the necessary and most important steps in recognizing the fundamentals of the parallel file system. It allows us to understand the state of the system under different applications. In this paper, we have collected I/O access data for a variety of parallel applications and provided an analysis that can characterize these applications.

The rest of this paper is organized as follows: we first provide some background and a list of related work done in Section II. In Section III, we give an overview of PVFS. Section IV provides details of the applications which were run on PVFS and in Section V we present details on our experiments and results. We close with conclusions and summaries.

II. Background & Related Work

There are three primary reasons that any high-performance application conducts I/O operations. They can be broadly classified as compulsory, checkpoint and out-of-core [10]. Compulsory accesses are generally due to reading initialization files, input files or writing intermediate/final outputs (application data or visualizations). These I/Os are necessary for the application to run and cannot be eliminated [6]. Checkpoints are intermediate data which record necessary information to restart the system after a failure. Checkpoint data is flushed at

specified intervals in terms of units of work, or of processing time. They are not necessary in order for the application to correctly execute. However, checkpointing is usually recommended because these large applications typically run for several hours or days and are subject to down times due to failures or maintenance. Checkpointing data is also occasionally used for other parametric studies. The frequency and size of a checkpoint is application dependent, but a high-performance file system can reduce the cost of checkpointing by exploiting knowledge of checkpoint I/O characteristics. Finally, out-of-core input/output is a consequence of limited primary memory. Larger primary memories can reduce the number and size of the out-of-core scratch files, but not obviate their need. With disk I/Os being the slowest component, it is important for the file system designers to know the nature of I/O requirements from these high-performance scientific applications.

There has been significant work on I/O analysis of parallel applications [6, 10–14]. For the most part, these studies have been done on message passing distributed memory parallel systems such as the Intel Paragon, Intel iPSC and Thinking Machines. Much of this work is several years old and may not represent current usage of parallel file systems. In [15, 16] Corbett et al. provided mechanisms in their Vesta file system to dynamically change the partitioning of data based on the intended access pattern to enhance the performance. Their work was based on access patterns of applications existing at that time and may not match with recent developments in the field. Smirni et al. in [17] studied the evolution of I/O access patterns of scientific applications and identified patterns that belonged to the application and not artifacts of optimizations made by the developers.

Recent work has looked at parallel I/O on the Cray XT [18]. Alam et al. in [19] characterized scientific workloads on multi-core processors, but without considering the effects of the file system. Their work evaluated the performance of the system based on the bandwidth and latency of the operation when run in different configurations. Kunkel and Ludwig evaluated the performance of PVFS architecture under synthetic workloads concentrating on software layers of PVFS instead of the disk I/Os [20]. In 2007, Ahmad published results which included disk I/O performance, block size and locality details, however they used synthetic workloads over ZFS and UFS file systems [21].

Our work differs from previous work in that we evaluate the workloads from the parallel file system point of view, specifically looking at the effect on individual I/O nodes under both synthetic and actual applications. We present details on how different configuration of servers on storage nodes effects the frequency of request on the file system, and how much variation we get on those parameters by changing the number of processors.

III. Parallel Virtual File System (PVFS)

There are many parallel storage file systems which have been developed recently and are in common use [3, 22–25]. Most of these file systems are based on the idea of separating the metadata from the data. By separating the metadata, storage management functionalities are kept away from the real data access, thus giving the user direct access to the data. These file systems can achieve high throughput by striping the data across many storage servers. Metadata is spread across one or more storage servers with clients performing file system activities over a shared network. In order to operate on a file, the client must first obtain the file metadata from a metadata server. This metadata information will include the location of the file data, i.e. information about which storage server stores which part of the file.

PVFS [26] is one such parallel file system and is installed on several high performance clusters today. PVFS is a high-performance parallel and distributed file system that utilizes two types of servers – metadata servers to handle file system's metadata and I/O servers to handle storage of file data. The node serving as metadata server can be a dedicated node or one of the I/O nodes or clients. On both servers, information is stored on top of the node's local file system. This is done in order to reuse existing solutions for the local storage task. By doing this, PVFS does not need to implement a proprietary storage mechanism for storing its data and can rely on sophisticated techniques of existing file systems. With the metadata separated from the actual data and data striped across several nodes, PVFS can easily obtain very high throughput.

PVFS provides multiple application programming interfaces including UNIX/POSIX and MPI-IO. MPI-IO is part of the standard programming interface for parallel applications, the message passing interface MPI. The MPI layer calls PVFS's functions directly to access data. Using MPI-IO shows a great benefit in comparison to Unix's POSIX interface. MPICH2 [27], which is the most commonly used implementation of MPI can be easily configured to support PVFS using ROMIO [28], a high-performance input/output implementation for MPI. ROMIO libraries, which are present within MPICH2 and run on a variety of parallel architectures, provide the abstract I/O device (ADIO) layer for PVFS. This ADIO layer provides the link between MPI and PVFS. All I/O requests made using MPI libraries are sent directly to the PVFS file system. PVFS also provides a kernel module for integration with Linux's VFS to run applications which require a POSIX interface to the parallel file system.

IV. Applications

Input/output characterization of an application code ideally includes access patterns and performance data from the application, its input/output libraries, file system, and device drivers [6]. Some of these are handled by cache at different levels of the system. Physical disk access patterns provide the ultimate evaluation of system's response. Disk accesses made by the application generally depend on its input/output modules, size of the data being handled, temporal spacing and spatial patterns generated from its computation libraries and certain file system's optimizations (eg. prefetching or caching). Tracing the I/O patterns gives us the real response of the system under that application and gives the file system

designers an insight into the application's requirements. At the same time, it is also a good resource for application developers to maximize the use of the file system policies.

Most parallel applications use one of the five major models of I/O as described below.

- Single output file shared by multiple nodes by ranges
- Large sequential read by single node at beginning of computation and large sequential write by single node at end of computation
- Checkpointing of state
- Metadata and read intensive - small data I/O, frequent directory lookups for reads
- Each node outputs to its own file

There are other I/O access patterns, but most applications use one or more of these patterns for the majority of their I/O. In this study, we selected a sample of parallel applications that use one of these forms of I/O in order to represent a wide range of parallel I/O applications. We don't investigate the last I/O class because in this case, the application can write its own file to local storage before flushing to a parallel file system. The popular benchmarking tool NPB-BMI uses the single output file model. The scientific application OpenAtom is largely distinguished by its I/O on several input states files during the computation. FLASH, another common scientific application uses frequent checkpointing of state to an output file, and finally, web server usage is characterized by heavy metadata usage. We shall briefly discuss the properties of each of these applications in this section.

A. NPB-BMI Benchmarking Tool

The NAS Parallel Benchmark (NPB) (BMI version 3.3) is a parallel benchmarking tool developed by the NASA Advanced Supercomputing Division [29, 30]. This tool was formerly known as BTIO. It presents a block-tridiagonal (BT) partitioning pattern on a three-dimensional array across a square number of compute nodes. Each processor is responsible for multiple Cartesian subsets of the entire data set, whose number increases with the square root of the number of processors participating in the computation. The I/O requirements and verification tests for the tools are as follows. After every five time steps, the entire solution field, consisting of five double-precision words per mesh point, must be written to one or more files. After all time steps are finished, all data belonging to a single time step must be stored in the same file, and must be sorted by vector component x-coordinate, y-coordinate and z-coordinate respectively. It uses MPI for communication and MPI-IO for I/Os. In our experiments, we ran NPB-BMI's 'C' class of "full_mpio" application which uses collective I/O to combine data accesses of multiple processes into large, regular I/O requests. This application provides an example of a parallel application in which each processor writes into a shared output file the data for which it is responsible, thus potentially contributing to the file system's fragmentation.

B. OpenAtom

OpenAtom is a highly scalable and portable parallel application for molecular dynamics simulations at the quantum level [31]. It is written in Charm++ [32] which is a parallel object-oriented programming language based on C++ and was designed with the goal of enhancing programmer productivity by giving a high-level abstraction of a parallel program to deliver good performance on the underlying system. OpenAtom attempts to solve important problems in material science and chemistry. Their approach uses Car-Parrinello ab initio molecular dynamics (CPAIMD) which involves a large number of inter-dependent phases with high-communication overhead including sparse 3D Fast Fourier Transforms (3D-FFTs), non-square matrix multiplies and dense 3D-FFTs. OpenAtom's I/O requirement consists of a large sequential read at the beginning of simulation to read the input data. At the end of simulation, the output is flushed as a large file thus initiating a sequential write. In addition, there are several intermediate checkpoint flushes which can be controlled using pre-defined parameters. OpenAtom requires POSIX access to file system and hence uses PVFS's virtual file system interface.

C. FLASH

FLASH is a block-structured adaptive mesh hydrodynamics code that solves fully compressible, reactive hydrodynamic equations, developed mainly for the study of nuclear flashes on neutron stars and white dwarfs [33]. The "Flash" problem is centered on simulating the accretion of matter onto a compact star, and the subsequent stellar evolution including nuclear burning either on the surface of compact star, or in its interior. The computational domain is divided into blocks which are distributed across the MPI processes. A block is a three dimensional array with an additional four elements as guard cells in each dimension on both sides to hold information from its neighbors. The I/O uses HDF5 [34], a higher level data abstraction which allows data to be stored along with its metadata in the same file. HDF5 is built on top of MPI-IO. FLASH flushes its data at checkpoint intervals over multiple checkpoint files and is a completely write-dominated workload. It hence provides us a different perspective of scientific applications.

D. Web Server

Since the metadata server is a prime component of a parallel file system such as PVFS, we chose a metadata heavy workload for our final application. We used the real web server traces of the 1998 Soccer World Cup website [35]. The traces were collected on 33 different web servers at four geographic locations. The real measurement was done over a period of 88 days during which it received 1.35 billion requests [36]. However, for our experiments, we ran the trace of 4 peak hours on one of the busiest days observed by the web server. The access rate averaged roughly 800 HTTP requests per second. Unlike the scientific applications that we have discussed above, the web server is very metadata intensive as it requires numerous directory lookups and file opens and

TABLE I
BASIC APPLICATION CHARACTERISTICS

Application	Number of Files Accessed	Number of Data Accesses	Data Read Percentage	Number of Metadata Accesses	Run Time (sec)
NPB-BMI	1	30,736	50	1	59
OpenAtom	135	174	91.8	135	297
FLASH	10	480	0	10	934
Web Server	4,965	11,469,846	100	2,433,391	14,524

closes. Thus, running these web server traces will give us an insight into the impact of real-world heavy metadata workloads on parallel file systems.

V. APPLICATION CHARACTERIZATION

We conducted our experiments on 16 dual-core AMD Opteron dual-processor machines each with 2GB RAM and a 80GB SATA disk drive. EXT3 served as the base file system on each of these machines. We modified PVFS (ver. 2.7.0) to collect traces about each request sent by the client and received by the server. The information which was logged included time-stamp, nature of request, size of request (in case of read/write), server number and file ID. This information which was added as a configurable option to PVFS would dump the trace to a separate log file, in a temporary directory. In our experiments, apart from tracing, PVFS was mounted with default options using TCP protocol.

For the Web server experiment, we took the World Cup trace and partitioned the trace so that the workload was load balanced across the available Apache servers. We configured from 2 to 16 Apache servers and split the requests among them. PVFS was configured with multiple metadata servers with each MDS colocated with an Apache server.

We analyze our results based on three important parameters. They are inter-arrival time, request size and long-range dependence of requests. Table I also shows other basic characteristics of the applications including number of files accessed, number of data accesses, percentages of data accesses that are reads, number of metadata accesses, and run time for 16 processors using 16 I/O nodes configuration.

A. Inter-arrival Time

Inter-arrival time is defined as the time difference between successive I/O requests on a node. It is a useful measure to help identify the load on the storage nodes and determines the required service from them. It can be used to determine the intensity of traffic observed by the node and hence can help design and configure both hardware and software components of the node.

Each application was run with 2 to 16 compute nodes and 2 to 16 I/O nodes, except for NPB-BMI where the number of compute nodes were 4, 9 and 16 since for NPB-BMI the number of processors must be a square. In the graphs in Figures 1–4, we show the cumulative interarrival time distribution seen by all I/O nodes.

As seen in Figure 1, NPB-BMI has very frequent accesses – more than 90% of the requests are within 2 milliseconds of the previous request in all cases. We see heavy load on storage nodes in case of 16 I/O nodes with over 90% of requests having inter-arrival time of less than a millisecond. However, increasing the number of storage nodes reduces the frequency of requests. This is because the data is shared between more nodes. It can also be seen that increasing the number of compute nodes increases the frequency of requests, which is because the new processors are sharing the I/Os and hence the storage node sees requests from more compute nodes.

The FLASH application (Figure 3) and the web server (Figure 4) show similar characteristics to NPB-BMI. For FLASH and the web server, it can be seen that with fewer processors, the frequency is reduced, which is similar to what was seen with NPB-BMI. Thus, as we scale up in processors we must be careful to scale up I/O capacity to match with the increase in computation. But scaling up I/O does not simply mean increasing the number of I/O nodes. As seen in the figures, distributing the I/O across more nodes does not decrease the interarrival times because the files are striped across all nodes which causes any read or write to access all nodes. If the latency is being affected because of the frequent accesses, it may be appropriate to change the striping so that files are distributed across only different subsets of the I/O nodes.

OpenAtom, on the other hand, does not show any significant difference in the interarrival rates as we change the number of processors (Figure 2). Increasing the number of processors reduces the interarrival time only by a small factor. This is because OpenAtom does most of its I/O through a single compute node. Although OpenAtom dumps checkpoint data at regular intervals, it is also done through a single compute node. Thus, the number of processors does not change the frequency of the I/O. From Figure 2 it is also clear that increasing the number of I/O nodes lengthens the interarrival times. This is simply because the data is distributed sequentially across more I/O nodes. As a result, striping across all nodes is a preferred option to increase throughput at the start and end, and latency is not likely to be an issue during the actual run of the application.

Figure 5 shows the average interarrival time of each metadata server using the web server application. For relatively few HTTP servers, the frequency of access to the metadata servers is not that great, but as we increase the number of HTTP servers the interarrival time decreases significantly. With 16 HTTP servers, the interarrival time distribution is nearly the same as the data I/O distribution. Even with the load distributed across 16 metadata servers, the each server can see significantly increased activity.

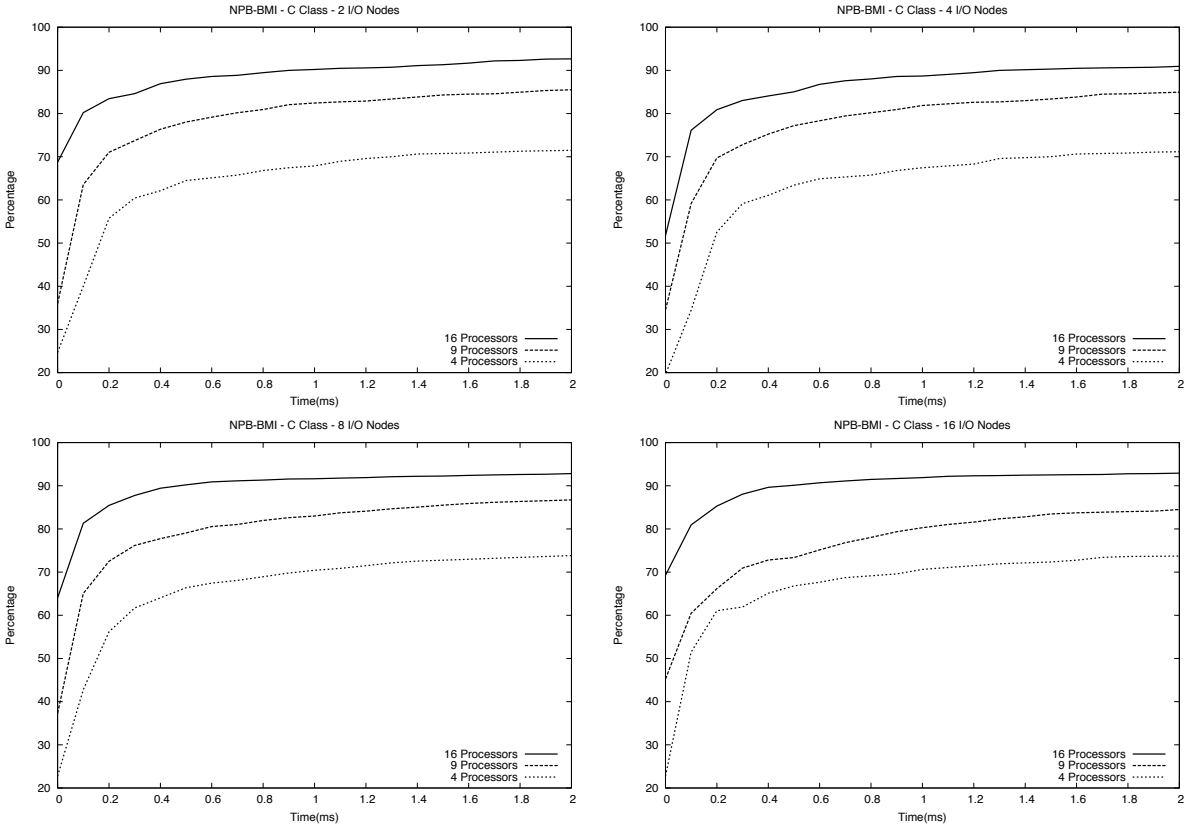

Fig. 1. Interarrival Time on NPB-BMI 'C' Class

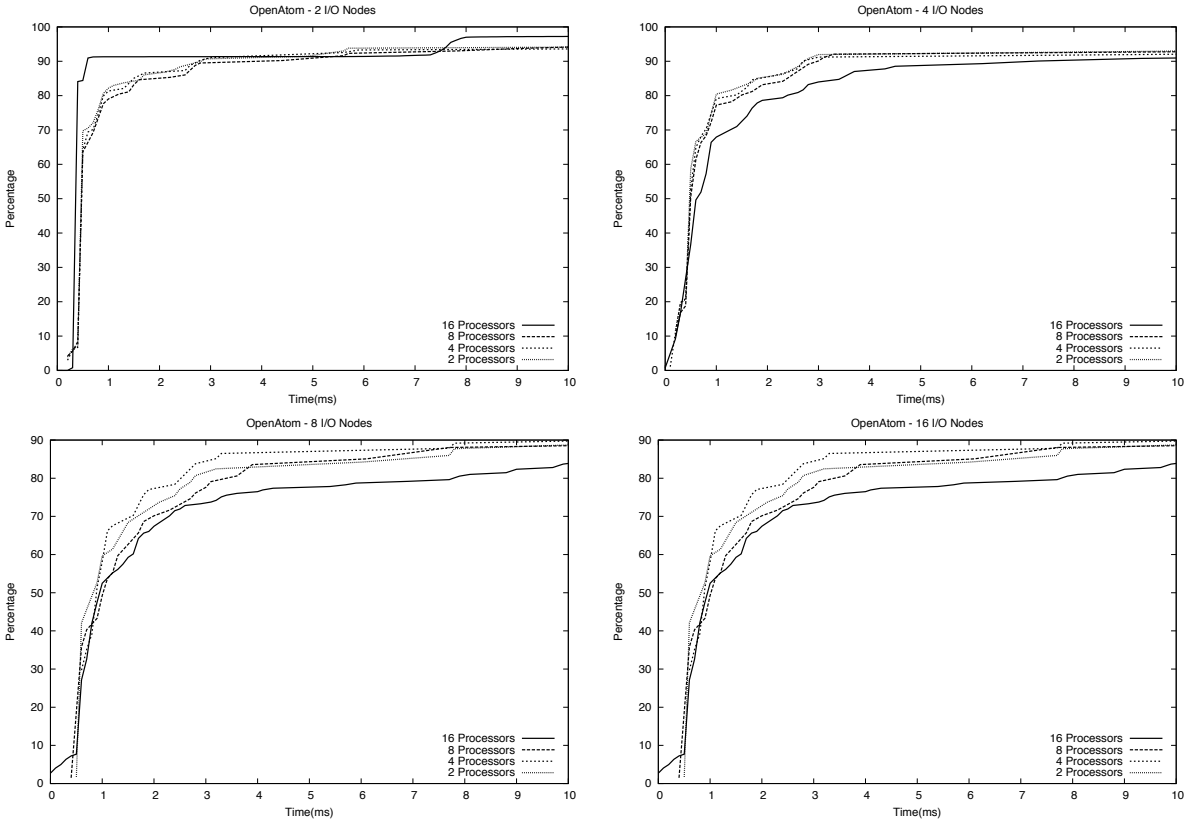

Fig. 2. Interarrival Time on OpenAtom

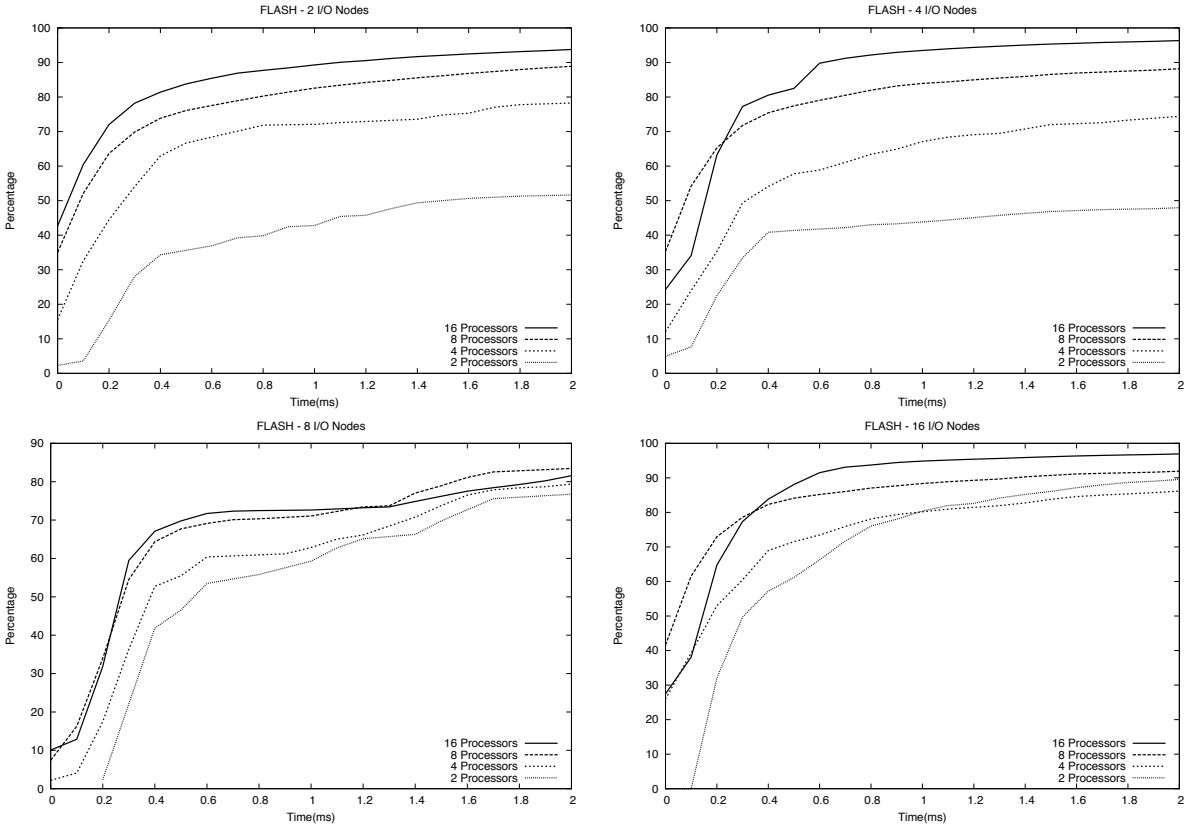

Fig. 3. Interarrival Time on FLASH

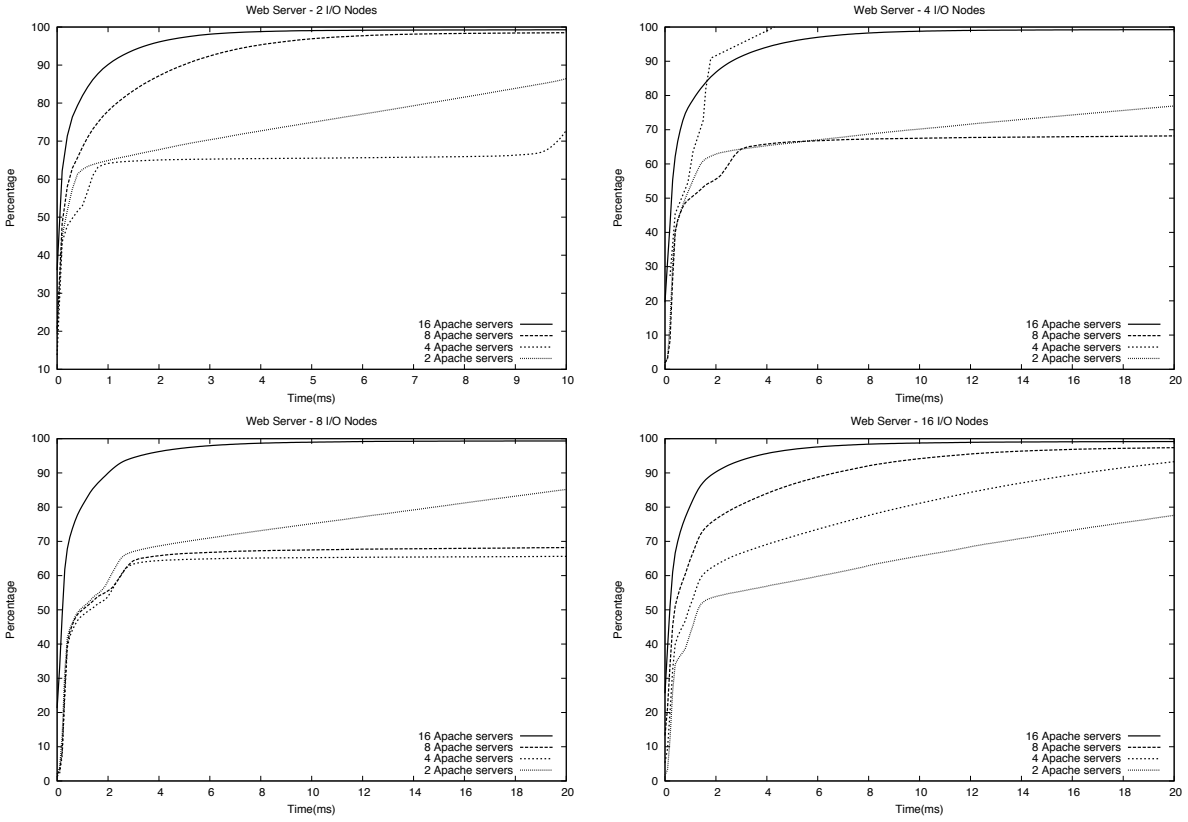

Fig. 4. Interarrival Time on Web server

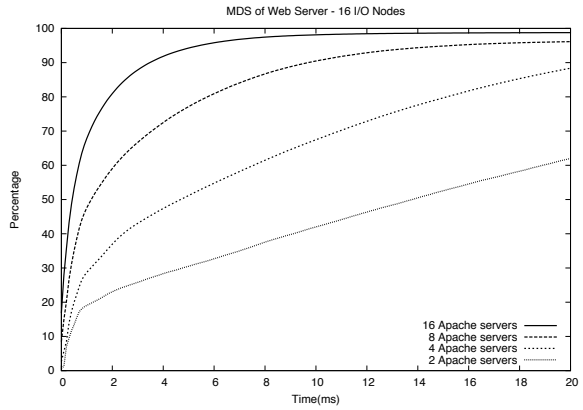

Fig. 5. MDS Interarrival time with 16 I/O nodes on Web server

TABLE II
REQUEST SIZE FOR THE NPB-BMI APPLICATION (IN MB)

Application	Number of Processors	I/O Nodes 2	4	8	16
NPB-BMI	4	15.4	15.4	15.4	15.4
	9	7.05	7.05	7.05	7.05
	16	3.34	3.34	3.34	3.34

B. Request Size

In this section, we examine the request size distribution, a measure of the number of blocks in each request sent to the disk (Table II). The larger the request the more likely that the underlying file system can deliver the parallelism required for performance.

In the case of NPB-BMI, we observed a fixed request size across different storage node counts. The request size however decreased as the processor count increased. This implies that the new processors are sharing the I/Os.

The request size for the FLASH, OpenAtom and web server workloads were fixed at 4K in size. This could be easily explained since all three of them do sequential reads of data through a single processor, they do not do very large I/O. Further, OpenAtom and the web server submit requests using the POSIX interface which uses Linux kernel's VFS and thus limiting the request size to 4K. An increase in VFS's page size limit could possibly help storage nodes see larger requests and at a slightly lower frequency thereby improving overall I/O performance.

C. Long Range Dependence

Many analytical studies of I/O assume a Poisson model for I/O requests. Poisson models presume that there is no dependence between a request and subsequent requests. However, real I/O does exhibit some long range dependence - i.e. request patterns that do not behave like the request patterns generated by a Poisson process. Markov I/O models that assume a probabilistic dependence also do not accurately reflect real I/O behavior. In our work, we calculate the Hurst parameter to describe self-similarity or long-range dependencies on the traffic generated by the workload as seen on the storage nodes. Self-similar traffic behaves the same when viewed at different degrees of traffic. This information could be used to understand the "burstiness" of requests. We draw on techniques to estimate the Hurst parameter as described in [37].

TABLE III
HURST PARAMETER FOR DIFFERENT WORKLOADS

Application	Processor Count 2	4	8	9	16
NPB-BMI	-	0.8850	-	0.8700	0.8245
OpenAtom	0.7831	0.7932	0.8628	-	0.8552
FLASH	0.6474	0.7660	0.7871	-	0.6013
Web Server	0.9439	0.9831	0.7707	-	0.9438

Let $X = (X_t : t = 0, 1, 2 \ldots)$ be a covariance stationary stochastic process with mean μ, variance σ^2 and autocorrelation function $r(k), k \gg 0$. Assume $r(k)$ of the form

$$r(k) \sim k^{-\beta} \quad , \quad k \to \infty$$

where $0 < \beta < 1$. The process X is called (exactly) second-order self-similar if for all $m = 1, 2, 3 \ldots$, $var(X^{(m)}) = \sigma^2 m^{-\beta}$ and

$$r^{(m)}(k) \sim r(k), k \geq 0.$$

and called (asymptotically) second-order self-similar if for k large enough,

$$r^{(m)}(k) \to r(k), m \to \infty$$

The Hurst parameter, H is characterization of long range dependence. To calculate the Hurst parameter and determine self-similarity, we first plot $var(X^{(m)})$ as a function of m. The variance-time (variance vs. m) plot is made on log-log scale. The relationship between the Hurst parameter and β is given by $H = 1 - \beta/2$, where $\beta = -\rho$ from the equation of the line $y = \rho x + b$ for each processor configuration. ρ is the slope of the line. Any process characterized by a slope less than 0 and greater than the reference line exhibits long range dependency and has an H parameter value of $1/2 < H < 1$.

Figure 6 shows the variance-time plot for OpenAtom with 16 I/O nodes. The data for each processor configuration series is fitted to determine the $y = \rho x + b$ line. The graph shows the slowly decaying variance of a self-similar series. It can be observed that the slope for all processor configurations has a slope between 0 and -1. Thus, for OpenAtom, the requests on the storage nodes exhibit Long Range Dependence. We found similar behavior for the other applications and the Hurst parameter for these applications is shown in Table III.

The Hurst parameter, H, takes on values from 0.5 to 1. A value of 0.5 indicates the data is uncorrelated and purely random, while values closer to 1 indicate high degree of persistence or long-range dependence. In our work, we obtained the Hurst parameter by calculating the slope of the best-fit line on each configuration. From this, it can be observed that all four applications have self-similar inter-arrival properties on the storage nodes. Thus, the traditional approach of using Poisson models to characterize I/O behavior is not adequate. NPB-BMI and Flash do I/O at regular intervals and OpenAtom does

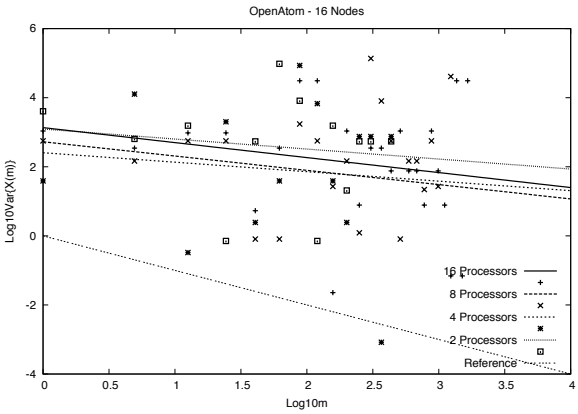

Fig. 6. Variance-time plot for 4 configurations on 16 I/O nodes for OpenAtom

major I/O at the beginning and end as well as checkpointing at regular intervals. Thus, these I/O accesses are clearly repetitively patterned and that is reflected in the high Hurst parameters. Web server I/O shows significant self-similarity because of the popularity of groups of files. A Hurst parameter value closer to 1 for the web server workload indicates that significant performance benefit could be achieved by using additional memory or caching on the storage nodes.

VI. Conclusion

In this paper, we conducted a survey of different applications and their impact on I/Os as seen on storage nodes and metadata servers in a parallel file system. From our study on interarrival times, we see that most parallel applications that do significant I/O during the run increase the I/O frequency as we increase the number of compute nodes. However, scaling I/O nodes alone will cause problems because the increased load is transferred to each I/O storage node. Thus, care must be taken with striping as you increase the number of I/O nodes. Other applications which do significant sequential I/O can and should use striping across as many nodes as possible to increase throughput. Our study on self similarity shows that most parallel applications exhibit significant long range dependencies. This result shows that I/O access models that assume independence or randomness between requests are not valid.

Acknowledgment

This work was supported in part by the National Science Foundation HECURA program under Award Number CCF-0621448. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation. The FLASH software used in this work was in part developed by the Department of Energy - supported ASC/Alliance Center for Astrophysical Thermonuclear Flashes at the University of Chicago, IL.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann Publishers, Inc., May 2002.

[2] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the Conference on File and Storage Technologies*, Monterey, CA, January 2002, pp. 231–244.

[3] P. J. Braam and R. Zahir, "Lustre - a scalable high performance file system," Cluster File Systems, Inc., Mountain View, CA, Tech. Rep., July 2001.

[4] P. H. Carns, W. B. Ligon, R. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in *4th Annual Linux Showcase and Conference*, Atlanta, GA, October 2000, pp. 317–327.

[5] S. Baylor and C. E. Wu, "Parallel I/O workload characteristics using Vesta," in *IPPS Workshop on Input/Output in Parallel and Distributed Systems*, April 1995.

[6] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed, "Input/output characteristics of scalable parallel applications," in *Proceedings of SuperComputing*, San Diego, CA, 1995, p. 59.

[7] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. L. Best, "File-access characteristics of parallel scientific workloads," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 10, October 1996, pp. 1075–1089.

[8] R. Thakur, W. Gropp, and E. Lusk, "Data sieving and collective I/O in ROMIO," in *Frontiers of Massively Parallel Computation*, February 1999, pp. 182–189.

[9] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, and B. B. Welch, "The Sprite network operating system," *IEEE Computer*, vol. 21, no. 2, pp. 23–26, February 1988.

[10] E. Miller and R. H. Katz, "Input/output behavior of supercomputing applications," in *Proceedings of SuperComputing*, Albuquerque, NM, Nov. 1991, pp. 567–576.

[11] B. Maron, T. Chen, B. Olszewski, S. Kunkel, and A. Mericas, "Workload characterization for the design of future servers," in *IEEE International Symposium on Workload Characterization*, Austin, TX, October 2005, pp. 129–136.

[12] A. L. N. Reddy and P. Banerjee, "An evaluation of multiple-disk I/O systems," in *IEEE Transactions on Computers*, vol. 38, no. 12, December 1989, pp. 1680–1690.

[13] ——, "A study of I/O behavior of PERFECT benchmarks on a multiprocessor," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, Seattle, WA, May 1990, pp. 312–317.

[14] D. Kotz and N. Nieuwejaar, "Dynamic file-access characteristics of a production parallel scientific workload," in *Proceedings of the 1994 Conferenceon Supercomputing*, Washington, D.C., November 1994, pp. 640–649.

[15] P. F. Corbett and D. G. Feitelson, "The Vesta Parallel File System," in *ACM Transactions on Computer Systems*, vol. 14, no. 3. ACM, 1996, pp. 225–264.

[16] P. F. Corbett, D. G. Feitelson, J.-P. Prost, G. S. Almasi, S. J. Baylor, A. S. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. Herr, J. Kavaky, T. R. Morgan, and A. Zlotek, "Parallel file systems for the IBM SP computers," *IBM Systems Journal*, vol. 34, no. 2, pp. 222–248, January 1995.

[17] E. Smirni, R. A. Aydt, A. A. Chien, and D. A. Reed, "I/O requirements of scientific applications: An evolutionary view," in *5th IEEE International Symposium on High Performance Distributed Computing*, Syracuse, NY, August 1996, pp. 49–59.

[18] W. Yu, J. S. Vetter, and H. S. Oral, "Performance characterization and optimization of parallel I/O on the Cray XT," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Miami, FL, April 2008, pp. 1–11.

[19] S. R. Alam, R. F. Barrett, J. A. Kuehn, P. C. Roth, and J. S. Vetter, "Characterization of scientific workloads on systems with multi-core processors," in *IEEE International Symposium on Workload Characterization*, San Jose, CA, October 2006, pp. 225–236.

[20] J. M. Kunkel and T. Ludwig, "Performance evaluation of the PVFS2 architecture," in *15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing*, Naples, Italy, February 2007, pp. 509–516.

[21] I. Ahmad, "Easy and efficient disk I/O workload characterization in VMware ESX Server," in *IEEE International Symposium on Workload Characterization*, Boston, MA, September 2007, pp. 149–158.

[22] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka, "A cost-effective, high-bandwidth storage architecture," in *Proceedings of Architectural Support for Programming Languages and Operating Systems*, October 1998, pp. 92–103.

[23] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM Symp. on Operation System Principles*, Lake George, NY, Oct. 2003, pp. 29–43.

[24] H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu, "The Panasas ActiveScale Storage Cluster - Delivering Scalable High Bandwidth Storage," in *Proceedings of SuperComputing*, Pittsburgh, PA, November 2004, p. 53.

[25] S. A. Weil, S. A. Brandt, E. Miller, and D. D. E. Long, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of Symposium on Operating System Design and Implementation*, Seattle, WA, November 2006, p. 22.

[26] "Parallel Virtual File System (PVFS)." [Online]. Available: http://www.pvfs.org

[27] MPICH2 - High Performance MPI. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpich2

[28] ROMIO: A High-Performance, Portable MPI-IO Implementation. [Online]. Available: http://www-unix.mcs.anl.gov/romio

[29] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS Parallel Bencmarks and its performance," NASA Advanced Supercomputing (NAS) Division, Moffett Field, CA, Tech. Rep. NAS-99-011, October 1999.

[30] P. Wong and R. F. V. der Wijngaart, "NAS Parallel Benchmarks I/O version 2.4," NASA Advanced Supercomputing (NAS) Division, Moffett Field, CA, Tech. Rep. NAS-03-002, January 2003.

[31] OpenAtom. [Online]. Available: http://charm.cs.uiuc.edu/OpenAtom

[32] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on C++," in *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, vol. 28, no. 10, Washington, D.C., September 1993, pp. 91–108.

[33] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, and H. Tufo, "FLASH: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes," *Astrophysical Journal Suppliment*, vol. 131, pp. 273–334, November 2000.

[34] (2007) HDF Group - HDF5. [Online]. Available: http://www.hdfgroup.org/HDF5

[35] Webserver traces - Soccer World Cup 1998. [Online]. Available: http://ita.ee.lbl.gov

[36] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," HP Laboratories, Palo Alto, CA, Tech. Rep. HPL-1999-35(R.1), September 1999.

[37] R. G. Clegg, "A practical guide to measuring the Hurst parameter," in *ArXiv Mathematics*, October 2006, pp. 3–14.