Case Studies in Storage Access by Loosely Coupled Petascale Applications*

Justin M. Wozniak
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL USA
wozniak@mcs.anl.gov

Michael Wilde
Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL USA
wilde@mcs.anl.gov

ABSTRACT

A large number of real-world scientific applications can be characterized as loosely coupled: the communication among tasks is infrequent and can be performed by using file operations. While these applications may be ported to large scale machines designed for tightly coupled, massively parallel jobs, direct implementations do not perform well because of the large number of small, latency-bound file accesses. This problem may be overcome through the use of a variety of custom, hand-coded strategies applied at various subsystems of modern near-petascale computers- but is a labor intensive process that will become increasingly difficult at the petascale and beyond. This work profiles the essential operations in the I/O workload for five loosely coupled scientific applications. We characterize the I/O workload induced by these applications and offer an analysis to motivate and aid the development of programming tools, I/O subsystems, and filesystems.

1. INTRODUCTION

Many modern scientific applications are structured as large arrangements of software units glued together by scripting languages such as Perl, Python, Tcl, or shell scripts [20]. This framework allows developers to quickly combine multiple tools together. For example, a simple case might involve performing a computation on a high-performance cluster, gathering the output and passing it through a plotting package for data visualization. More complex constructions perform metacomputations, such as selecting input parameters for future computations or obtaining resources. Scripting has become a prevalent model for scientific application development but faces particular challenges posed by the I/O mechanisms and filesystems on petascale computers. In this paper, we provide a coarse characterization of the I/O workload produced by five applications built on the Swift language [31].

A feature of software produced with scripting toolkits is that it is highly portable, promoting code reuse and providing flexibility of choice for resources. The portability strengths of scripting have brought it to the recently available near-petascale and petascale machines. Scripting languages typically provide an interprocess communication (IPC) mechanism for communication less complex [16] than provided by MPI [11]. While such scripted programs can achieve the massive parallelism available on these machines, portability comes by performing IPC through the filesystem. The filesystem thus becomes a bottleneck. For example, the 160,000-core BlueGene/P at Argonne National Laboratory offers a GPFS [24] filesystem with total bandwidth of 65 GB/s, yet only 400 KB/s is available per core [21], and a file creation rate within a single directory of 40/s gives about 1/hour per core [22]. Clearly, a simple application consisting of many small accesses to the file system would not be efficient, yet due to the complexity of the applications, it is not immediately obvious how to improve the situation in the general case.

We propose that aggregating case studies will help formulate Collective Data Management (CDM) strategies that can be provided to users through scripting language constructs and systems software. First, the set of primitive characteristics must be determined. This involves capturing essential filesystem operations and use cases that impact performance, such as number of file and directory creations and accesses, file size, the balance of read and write operations on files, the size of those accesses, the sequentiality of those accesses, and the long-term disk usage patterns for resulting output data. Next, a set of potential optimizations may be captured and categorized. These steps involve a *careful look at real applications*, performed herein. Future work will involve the construction of language-level solutions to the data management challenges in scripted applications. The output of this work could also improve filesystems themselves by providing new features such as performance optimizations for the small access patterns described here.

I/O has long been identified as a component of the petascale challenge [10]. Many advances have been made in the development of parallel I/O [7] for tightly coupled applications, but the study of I/O performed by large batches of small independent tasks is relatively new. We describe the I/O patterns of five applications that can consume the computing power of petascale machines. Our objective is to characterize important application features that can improve the development of scripting tools, I/O systems, and filesystems.

The remainder of this work is organized as follows. In the next section, we provide background on loosely coupled applications and describe relevant I/O and storage technologies. In Section 3, we describe the applications studied here in detail and extract their performance-critical data access operations. Section 4 contains our analysis of the application characteristics, and we conclude in Section 5 with a brief summary.

*

Supercomputing PDSW '09 Portland, OR USA

2. RELATED WORK

Scientific applications that are composed of large numbers of tasks coupled by filesystem operations have been well studied [26]. Such workflows have proven to be portable [12], running on opportunistic systems [28] such as desktop grids, and scaling up to large production systems such as the TeraGrid [5]. Parallel scripting has been brought to large-scale job submission systems through the Swift language [31], and the use of massively parallel machines has been aided by the efficiency brought by the Falkon scheduler [23].

Once a user has issued hundreds of thousands of tasks communicating through the filesystem, one must consider the effect of the large number of small, latency-bound filesystem operations involved, or the rereading of the same data sets by large numbers of apparently independent processes. Collective I/O operations were proposed [30] to aggregate many relatively small reads and writes into larger operations. This method relies on an intermediate cache to perform aggregation. BAD-FS [3] and data diffusion [21] use data-aware scheduling and caching on large scale production systems to increase data locality.

Enhanced filesystem features have been proposed to address the problem at file server component or filesystem client component. Small file and metadata operations were improved in the Chirp file system [27] by hybridizing the protocol between RPC and streaming techniques, as well as adding new, nontraditional filesystem calls for commonly performed operations. Similarly, small file and metadata operations were improved for the Parallel Virtual Filesystem (PVFS) [4] by precreating data objects for files, utilizing locality for small file data and metadata, and using eager messages for small data movement. New technologies such as object storage devices may be tapped to improve the performance of directory operations [1]. Contrarily, the BlueFS system [19] increases performance for applications with latency-bound operations by performing speculative execution in the client kernel, reducing latency for predictable functions.

Augmenting established standards is another route to improving performance for the applications studied here. Extending the commonly implemented POSIX operating system interface for high-end computing systems has been proposed [13] to improve performance for a wide range of highly concurrent applications. For example, the `readdirplus()` extension has been implemented in the Chirp and PVFS systems, and its use could benefit applications that perform large numbers of directory queries. Additionally, NFSv4 [25] extends NFS in ways that could improve the scalability of metadata-intensive applications, including the use of compound operations.

The application script itself may contain information that can be tapped to improve the application-visible performance of the I/O system. Job submission scripts may be annotated with directions to the storage system regarding intended file accesses [17]. MapReduce [8] and All-Pairs [18] are programming models that provide complete information about the application data access pattern.

3. CASE STUDIES

3.1 Applications

3.1.1 OOPS

OOPS [9] is a protein folding software package based around the Protein Library, a protein structure toolkit. Using a model that reduces interactions through a coarse-grained statistical potential, OOPS-based simulated annealing produces reliable structures with minimal side-chain and nearest neighbor complexities. Our OOPS script evaluates many potential protein structures in parallel, then performs postprocessing and visualization on the resulting output. This process repeats until an acceptable structure has been detected, signaling convergence.

3.1.2 DOCK

DOCK [15] is a molecular program that quickly analyzes the docking potential of large numbers of molecules against a set of target sites. The model employed by this software places each molecule in the binding site at the target and evaluates the conformational space at that interaction. Our DOCK script pairs large numbers of target sites against a database of ligand molecules, selecting those that fit.

3.1.3 BLAST

BLAST [2] is a DNA and amino acid search tool to detect alignments of two sequences that are minimal in variation. BLAST uses a heuristic method to assign mutation scores to sequence pairs to quickly obtain probable sequence similarities. Our BLAST script performs large numbers of sequence analysis computations in parallel and reduces the results into output indicating the matches.

3.1.4 PTMap

PTMap [6] is a software package designed to match mass spectroscopy data against a database of protein sites. To avoid the generation of large numbers of false positives, PTMap uses several algorithmic enhancements that reduce false positives, extracting relevant signal peaks from noise. Our script scores PTMap results for pairs of spectroscopy data sets against proteins in parallel, followed by analysis and summarization.

3.1.5 fMRI

The fMRI application [14] considered here analyzes brain regions for response to experimental stimuli. A relational database of responses for a given subject may be queried for analysis, providing statistical connections to be made between MRI data and brain function. Our fMRI script pulls records from the MRI database, performing statistical tests on each brain region using the statistical analysis language R and writes the result.

3.2 System Architecture

As diagrammed in Figure 1, our target petascale system architecture consists of several components of interest to small-task I/O. The infrastructure consists of three major hardware sections: the file servers (FS), the intermediate servers (IS), and the application compute nodes (APP). Compute nodes are assumed to be connected by a high-performance (possibly specialized) interconnect, ideal for low-latency, high-bandwidth messages required by typical high-performance computing applications. Compute nodes are connected to intermediate services, which are connected to each other and to file services via a commodity network. File servers are assumed to be participating in a large-scale deployment of a parallel file system. Intermediate nodes cache and aggregate I/O operations to reduce the impact of small, latency-bound file operations on the file system.

3.3 Application Profiles

Table 1 profiles the five applications covered in this report by diagramming the essential operations performed by the workflow model, as well as quantifies essential usage statistics. Additionally, certain easily optimized data (file) movement operations may be characterized by one of the following patterns:

- Ⓑ Broadcast: The same data set is obtained by multiple receivers.

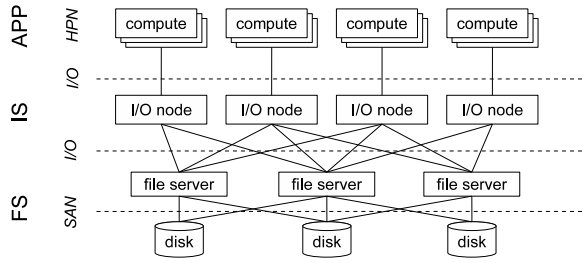


Figure 1: Coarse schematic of a petascale architecture.

Ⓢ Scatter: A collection of data sets is split up and sent to multiple receivers.

Ⓖ Gather: A distributed data set is aggregated by a single receiver. A typical use case of a gather operation is a data reduction or selection, which could involve performing an operation on a set of results, aggregating results into a compact data set, or culling unnecessary results.

These operations may be optimized through special uses of the network such as multicast, or by using tree-based algorithms.

Data objects are represented by cylinders. Non-persistent data objects are represented by dashed cylinders; these data sets are not required by the user in the final output.

These two notations indicate the potential for optimization by transforming the portable file system calls used by the application into message-oriented operations. For example, if multiple application invocations read the same data set, the load on the file system can be reduced by performing a single read and employing an efficient broadcast. Similarly, data written by a process and re-read by a successive, dependent process may avoid using the filesystem altogether by forwarding the data set directly from the writer to the reader.

The column in Table 1 labeled **Statistics** indicates the **total** I/O *as performed by the tasks*. **I/O Reduction** indicates the theoretical fraction of I/O that may be eliminated through the application of CDM strategies:

$$\text{reduction} = 100\% - \frac{\text{I/O seen by FS}}{\text{I/O seen by APP}}.$$

Additionally, some application characteristics are denoted for discussion below.

3.4 Application Scale

Each script consists of a variable number of sequential tasks, symbolized by N and M. An invocation of each application is capable of consuming much or all of the parallelism on a near-petascale machine, i.e., 50,000 concurrent tasks or more; individual task run times are short (5-10 minutes).

OOPS:
$N \approx 5 - 10$
$M \approx 10,000$

DOCK:
$N \leq 1,000,000$
$M \approx 20$

BLAST:
$N \approx 1,000,000$

PTMap:
$N \approx 50$
$M \approx 1000$

fMRI:
$N \approx 100,000$

4. ANALYSIS

4.1 Reducing I/O and Application Patterns

The results from Table 1 indicate that a great deal of the I/O workload may be reduced by applying the CDM strategies. In the first four cases, the I/O seen by the FS may be reduced by more than 99%. The actual result requested by the user is often relatively small; the total I/O is primarily used to pass intermediate results from one component task to another. In an MPI application, this would not be described as I/O at all; however, when scripting, the application writer does not specify the nature of the I/O operation. Tools to automate the application of CDM strategies must be developed to maintain the ease of scripting while ensuring efficiency.

Each application gains an I/O reduction through caching. An example is shown in the OOPS diagram, where a 10 MB file is written and then reread at the next iteration. This data should be cached to prevent accessing the FS; however, large runs could exceed the size of the IS; and if the IS is used as an LRU cache, additional FS accesses could be necessary. Thus, in order to ensure the locality of the intermediate data sets, a data-aware scheduler must be used.

Two applications, BLAST and fMRI, show the MapReduce pattern of data distribution, computation, and output reduction. Notably, a straight forward MapReduce port would still not be efficient if it did not recognize the large broadcast in the BLAST case. (The MapReduce pattern in BLAST workflows was previously noted [16].) The DOCK and PTMap applications use the All-Pairs [18] pattern.

4.2 Parallelism and Contention

As is typical in scripted workflows, all application data operations read or write whole files. This approach eliminates the need for the FS to manage write consistency under contention within a file or manage shared file pointers. Contention for modifying a directory, however, is a constraint. Additionally, none of the component application tasks are parallel applications, so they cannot benefit from MPI-IO [29] optimizations. As noted in the introduction, modifying a directory introduces write contention in the FS. Currently, the cost is reduced by manually distributing file creation across multiple directories. The PTMap application generates an index of Unix links to structure the selected data sets, a process made tolerable by limiting the concurrency of directory accesses.

4.3 Post-petascale Developments

The road ahead for post-petascale parallel scripting applications faces I/O challenges. We assume near-term machines in the 20-100 petaflop/s range will contain 1-2 million processor cores. The run time of individual tasks is not expected to change substantially as MIPS rate gains are expected to be modest. Additionally, memory per node is not expected to increase. This situation has two implications for CDM strategies. First, the number of files will increase with N and M as used in Table 1, increasing the importance of efficient filesystem metadata processing. Second, caching will become more complex as the number of cores may grow faster than size of the IS cache space, necessitating data-aware scheduling.

5. SUMMARY

In this report we have provided a coarse-grained description of the data access workloads produced by five scripted scientific applications. We have identified common I/O patterns that may be captured and exploited to improve the performance of the I/O system as well as to reduce the responsibilities of the script writer. We intend that the contribution of this work will enhance the usability and efficiency of petascale computers.

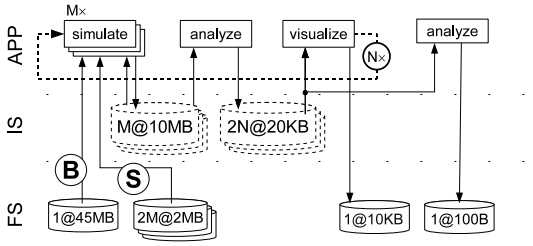
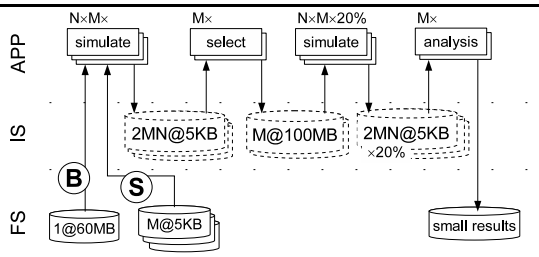
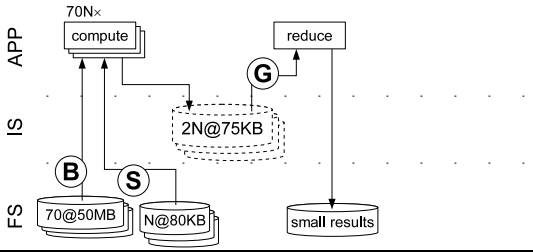
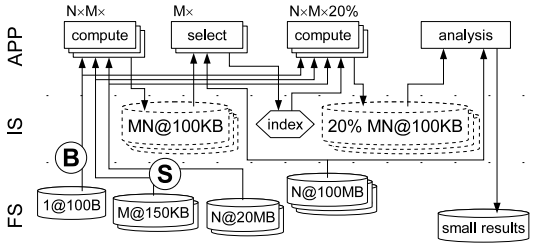
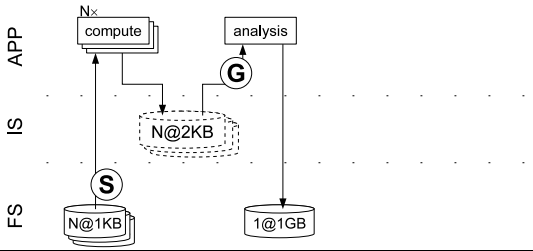
Application	Diagram	Statistics	I/O Reduction
OOPS		read: 5.7TB write: 1TB *iteration* (§ 4.1) *overwrites* (§ 4.2)	input: 99% output: 99%
DOCK		read: 3.2PB write: 2PB *all-pairs* (§ 4.1)	input: 99% output: 99%
BLAST		read: 3.5PB write: 150GB *map-reduce* (§ 4.1)	input: 99% output: 99%
PTMap		read: 1.1TB write: 6GB *directory ops* (§ 4.2) *all-pairs* (§ 4.1)	input: 99% output: 99%
fMRI		read: 18MB write: 1GB *map-reduce* (§ 4.1)	input: 66% output: 17%

Table 1: Application profiles.
All file sizes represent one of many possible use cases and are approximations.
Task dependencies are denoted with arrows; execution generally flows from left to right.

6. ACKNOWLEDGMENTS

We would like to thank application collaborators for their input when conducting this study, including Aashish Adhikari and Sarah Kenny. We also would like to thank Robert Ross for feedback on the project. This research is supported in part by NSF grant OCI-721939, NIH grants DC08638 and DA024304-02, the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy under Contracts DE-AC02-06CH11357 and DE-AC02-06CH11357. Work is also supported by DOE with agreement number DE-FC02-06ER25777.

7. REFERENCES

[1] N. Ali, A. Devulapalli, D. Dalessandro, P. Wyckoff, and P. Sadayappan. An OSD-based approach to managing directory operations in parallel file systems. In *Proc. CLUSTER*, 2008.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Molecular Biology*, 215(3), 1990.

[3] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Explicit control in a batch-aware distributed file system. In *Proc. USENIX Symposium on Networked Systems Design and Implementation*, 2004.

[4] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig. Small-file access in parallel file systems. In *Proc. International Parallel and Distributed Processing Symposium*, 2009.

[5] P. A. Cheeseman, M. W. Deem, D. J. Earl, , and W. I. Whitson. Adapting an application for use in a Condor based parameter sweep on TeraGrid. In *Proc. TeraGrid 2007 Conference*, 2007.

[6] Y. Chen, W. Chen, M. H. Cobb, and Y. Zhao. PTMap – A sequence alignment software for unrestricted, accurate, and full-spectrum identification of post-translational modification sites. *Proceedings of the National Academy of Sciences of the USA*, 106(3), 2009.

[7] A. Ching, K. Coloma, J. Li, W. keng Liao, and A. Choudhary. High-performance techniques for parallel I/O. In *Handbook of Parallel Computing: Models, Algorithms and Applications*, chapter 35. 2008.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. Operating Systems Design and Implementation*, 2004.

[9] J. DeBartolo, A. Colubri, A. K. Jha, J. Fitzgerald, and T. R. S. Karl F. Freed. Mimicking the folding pathway to improve homology-free protein structure prediction. *Proc. National Academy of Sciences*, 106(10), 2009.

[10] J. J. Dongarra and D. W. Walker. The quest for petascale computing. *Computing in Science and Engineering*, 3(3), 2001.

[11] M. P. I. Forum. MPI: A message-passing interface standard, 1994.

[12] Y. Gil, P. A. González-Calero, and E. Deelman. On the black art of designing computational workflows. In *Proc. Workshop on Workflows in Support of Large-Scale Science*, 2007.

[13] G. Grider, L. Ward, R. Ross, and G. Gibson. A business case for extensions to the POSIX I/O API for high end, clustered, and highly concurrent computing, 2006. Available at: http://www.opengroup.org/platform/hecewg/uploads/40/10891/POSIXIO-API-Business-case-HEC-ggrider.pdf.

[14] U. Hasson, J. I. Skipper, M. J. Wilde, H. C. Nusbaum, and S. L. Small. Improving the analysis, storage and sharing of neuroimaging data using relational databases and distributed computing. *Neuroimage*, 39(2), 2008.

[15] D. M. Lorber and B. K. Shoichet. Hierarchical docking of databases of multiple ligand conformations. *Current Topics in Medicinal Chemistry*, 5(8), 2005.

[16] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang. Introducing Map-Reduce to high end computing. In *Proc. Petascale Data Storage Workshop*, 2008.

[17] H. M. Monti, A. R. Butt, and S. S. Vazhkudai. /Scratch as a cache: Rethinking HPC center scratch storage. In *Proc. International Conference on Supercomputing*, 2008.

[18] C. Moretti, J. Bulosan, D. Thain, and P. J. Flynn. All-pairs: An abstraction for data-intensive cloud computing. In *Proc. International Parallel and Distributed Processing Symposium*, 2008.

[19] E. B. Nightingale, P. M. Chen, and J. Flinn. Speculative execution in a distributed file system. *ACM Transactions on Computer Systems*, 24(4), 2006.

[20] J. Ousterhout. Scripting: Higher-level programming for the 21st century. *IEEE Computer*, Mar. 1998.

[21] I. Raicu, I. Foster, Y. Zhao, P. Little, C. Moretti, A. Chaudhary, and D. Thain. The quest for scalable support of data-intensive workloads in distributed systems. In *Proc. High Performance Distributed Computing*, 2009.

[22] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, and B. Clifford. Towards loosely-coupled programming on petascale systems. In *Proc. SC'08*, 2008.

[23] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: A Fast and Light-weight tasK executiON framework. In *Proc SC'07*, 2007.

[24] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *Proc. USENIX Conference on File and Storage Technologies*, 2002.

[25] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck. Network File System (NFS) version 4 protocol. RFC 3530, 2003.

[26] I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors. *Workflows for e-Science*. Springer, 2007.

[27] D. Thain and C. Moretti. Efficient access to many small files in a filesystem for grid computing. In *Proc. Conference on Grid Computing*, 2007.

[28] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4), 2005.

[29] R. Thakur, W. Gropp, and E. Lusk. On implementing MPI-IO portably and with high performance. In *Proc. of the Sixth Workshop on I/O in Parallel and Distributed Systems*, May 1999.

[30] Z. Zhang, A. Espinosa, K. Iskra, I. Raicu, I. Foster, and M. Wilde. Design and evaluation of a collective I/O model for loosely-coupled petascale programming. In *Proc. MTAGS Workshop and SC'08*, 2008.

[31] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *Proc. Workshop on Scientific Workflows*, 2007.