Managing Variability in the IO Performance of Petascale Storage Systems

Jay Lofstead[1], Fang Zheng[1], Qing Liu[2], Scott Klasky[2],
Ron Oldfield[2], Todd Kordenbrock[4], Karsten Schwan[1], and Matthew Wolf[1,2]

[1]College of Computing, Georgia Institute of Technology, Atlanta, Georgia
[2]Oak Ridge National Laboratory, Oak Ridge, Tennessee
[3]Sandia National Laboratories, Albuquerque, New Mexico
[4]Hewlett-Packard Company, Nashville, Tennessee

***Abstract*—Significant challenges exist for achieving peak or even consistent levels of performance when using IO systems at scale. They stem from sharing IO system resources across the processes of single large-scale applications and/or multiple simultaneous programs causing internal and external interference, which in turn, causes substantial reductions in IO performance. This paper presents interference effects measurements for two different file systems at multiple supercomputing sites. These measurements motivate developing a 'managed' IO approach using adaptive algorithms varying the IO system workload based on current levels and use areas. An implementation of these methods deployed for the shared, general scratch storage system on Oak Ridge National Laboratory machines achieves higher overall performance and less variability in both a typical usage environment and with artificially introduced levels of 'noise'. The latter serving to clearly delineate and illustrate potential problems arising from shared system usage and the advantages derived from actively managing it.**

I. Introduction

To meet the performance demands of petascale applications and science, HPC file systems continue to grow in both extent and capacity. For example, the new file system at Oak Ridge National Laboratory supporting the petascale Jaguar machine has 672 individual storage targets (OSTs) and over 10 petabytes of storage. Storage targets can be used in parallel, resulting in a theoretical peak of generally around 60 GB/sec aggregate performance (as much as 90 GB/sec with optimal network organization) and it is clear that such performance levels are needed when up to 225,000 compute cores can concurrently generate output. Additional performance requirements are due to file system sharing across multiple machines, as is the case at both ORNL and NERSC, where IO systems are used simultaneously by petascale machine applications that generate output data and by analysis or visualization codes that consume it.

Extensive prior work is focused on the performance of shared file systems used by enterprise applications that generate rich and varying mixes of read/write accesses to large numbers of files. Topics range from driver-level work on efficient algorithms for disk access to system-level strategies for effective buffering to alternative file organizations [47] used in file systems to diverse methods for content distribution across multiple OSTs and/or machines such as file striping, etc. The parallel file systems used at ORNL, NERSC, and other supercomputing sites, in fact, use many of the sophisticated techniques developed in such research. In addition, HPC researchers have developed novel methods in support of high performance IO, which include data staging [1], [41], the use of alternative file formats or organizations [8], [34], [29], [30], and better ways to organize and update file metadata [42], [16], [20], [57], [31].

Despite their use of state of the art approaches like those described above, the large parallel file system installations at sites like ORNL or NERSC continue to face significant challenges when they are used 'at scale'. This is due to several facts. First, in contrast to most enterprise applications, an HPC application can demand instantaneous and sole access to a large fraction of the parallel file system's resources. An example is a petascale code that outputs restart data. If IO resources are insufficient, this code will block and waste CPU cycles on compute nodes waiting for output rather than making positive progress for the ongoing scientific simulation. Such latency sensitive behavior is characterized by periodic output patterns, with little or no IO activity for the 15 or 30 minutes of duration of alternating computation and output steps, thereby providing distinct deadlines for IO completion. Second, the resource demands imposed by single large-

 SC10 November 2010, New Orleans, Louisiana, USA 978-1-4244-7558-2/10/$26.00

scale codes are magnified by the simultaneous use of the petascale machine by multiple batch-scheduled applications, each desiring a substantial portion of IO system resources and each demanding low latency service. Third, when file systems are shared, like those at ORNL and NERSC, it is not just the petascale codes that demand IO system resources, but there are also additional requests that stem from the analysis or visualization codes running on select petascale machine nodes and/or on attached cluster machines with shared file system access.

The facts listed above all contribute to an important phenomenon observed in the IO systems used with petascale machines, which is that of high levels of variability in IO performance. Measurable sources of such variability include the following:

- *Internal interference* occurs when too many processes within a single application attempt to write to a single storage target at the same time. This causes write caches to be exceeded leading to the application blocking until buffers clear.
- *External interference* can occur even if an application takes great pains to properly use storage resources, since it is caused by 'shared' access to the file system, an example being analysis codes running on an attached cluster machine that attempt to read data stored in the shared scratch space at the same time as the petascale machine is writing its output data. Another example is simultaneous file system use by multiple applications running simultaneously on the petascale machine.

An additional issue is lack of scalability in metadata operations, which has been considered in extensive past research. The LWFS file system, for example, decouples metadata from data operations and postpones them, when possible [42], and the partial serialization approach described in our own previous work with Jaguar [35], [32] reduces intra-application sources of contention experienced by the metadata server.

Prior work in the enterprise domain does not adequately address the internal or external interference effects observed on petascale machines. This is because in enterprise systems, the principal concern has been to properly sequence and batch read vs. write operations on large numbers of files, in ways that leverage processes' sequential read behavior to reduce disk head movement, while also effectively using available buffer space [6], [22]. These solutions may help with interference effects on single storage targets, but they do not address the load balancing or uneven usage across the multiple OSTs seen in HPC storage systems.

We have developed a new set of dynamic and proactive methods for managing IO interference. These *adaptive IO* methods improve IO performance by dynamically shifting work from heavily used areas of the storage system (i.e., storage targets – OSTs) to those that are more lightly loaded. Adaptive IO is complemented by additional techniques that stagger file open (i.e., metadata) operations to manage performance impacts on the metadata server. By using adaptive IO, we have been able to substantially improve the IO performance of petascale codes, including that of fusion simulations like GTC [24], XGC1 [12], GTS [56], and Pixie3D [11]. These codes generate restart and analysis data every 15 or 30 minutes, with full scale, production data sizes generally between 64 MB and 200 MB per process. For a typical petascale run of around 150,000 processes, 200 MB per process yields 3 TB to be written every 30 minutes. Staying within a generally acceptable 5% of wall clock time spent in IO limit, this requires a minimum sustained speed of 35 GB/sec. With the current Lustre limit of a maximum of 160 storage targets for a single file, and a per storage target theoretical maximum performance of around 180 MB/sec, a maximum of only 28 GB/sec can be achieved in theory, assuming perfectly tuned IO routines and an otherwise quiet system. Removing this limit can address internal interference, of course, but it does not help with external interference in a busy system. In response, adaptive IO is designed so as to cope with both internal and external interference effects, the goal being to consistently achieve $> 50\%$ of theoretical peak IO performance.

Experimental results presented in this paper first assess and diagnose the presence and effects of internal and external interference in petascale storage systems. Based on the insights gained from these evaluations, adaptive IO methods are implemented in the context of the ADIOS IO middleware now widely deployed for petascale codes [28]. The outcome is a substantial improvement in IO performance, ranging from around 2x the average performance for a 16384 process run of XGC1 to more than 4.8x for the 16384 process run of Pixie3D with 16 TB output per IO, all with less variability in the time spent performing IO.

The remainder of this paper is structured as follows. Section II experimentally establishes the existence of both internal and external interference for multiple large-scale parallel file systems. We then describe the design, software architecture and implementation details of adaptive IO in Section III. Section IV presents experimental evaluations, using both actual petascale applications and synthetic benchmarks, the latter to better characterize certain performance properties and behaviors. Results are discussed in Section IV-C followed by an outline of related work in Section V.

Conclusions and future work appear in SectionVI.

II. Problem and Motivation

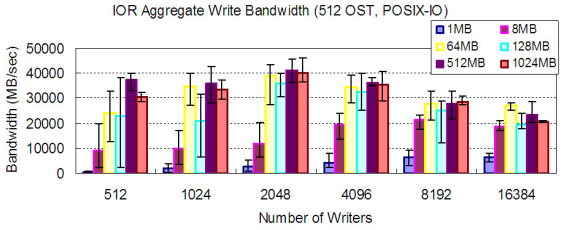
Variability in file system performance due to concurrent use has existed since multi-user operating systems were developed, causing parallel file systems to employ rich caching and other performance management techniques for their internal storage targets. The *internal* and *external* interference effects seen in parallel file systems, however, are not adequately addressed by these techniques, as validated by the performance measurements taken on multiple machines and file systems presented below.

The first set of measurements use the petaflop partition of the Jaguar machine at Oak Ridge National Laboratory. This is a Cray XT5 machine with 18,680 nodes, each with dual, hex-core AMD Opteron processors (224,160 cores) and with 16 GB of RAM per node. The scratch file system is a 672 storage target Lustre 1.6 system with 10 PB total storage shared across multiple machines at ORNL. Second are measurements on the XTP machine at Sandia National Laboratories, which is a Cray XT5 with 160 nodes, each with dual, hex core AMD Opteron processors (1,920 cores) with a Panasas file system (PanFS) configured with 40 StorageBlades for a total of 61 TB of storage. Third are results attained on the Franklin Cray XT4 MPP at NERSC. Franklin has 38,128 Opteron compute cores, and its scratch file system is Lustre with 96 storage targets and 436TB storage. Experimental data concerning Jaguar and XTP are collected by the authors of this paper; performance data on Franklin is obtained from NERSC's online performance monitoring data repository [38].

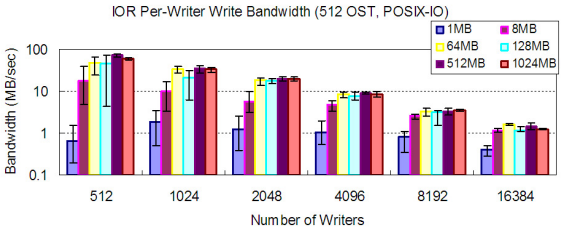
Measurements reported below first document the existence of internal interference and its impact on aggregate write bandwidth. External interference and its impacts are shown second. The section concludes with a summary of results and insights. To strictly isolate interference effects, all reported measurements specifically omit file open and close times.

1) Internal Interference: Using Jaguar/Lustre and the IOR benchmark [51], we demonstrate internal interference by writing data of differing sizes via different ratios of processes to storage targets (OSTs). In all such tests, the IOR program is configured to use 512 OSTs, where each process writes data to a separate file and to some fixed OST using POSIX-IO. Writers are split evenly across the 512 OSTs.

Figure 1(a) depicts the scaling of IOR POSIX-IO aggregate write bandwidth on Jaguar with different numbers of writers and different per-writer data sizes. Figure 1(b) shows the corresponding average per-writer bandwidth values at different scales. In both figures, each bar represents the average value among 40 samples with error bars depicting maximum and minimum values. The ratio of processes per storage target ranges from 1 to 32, and the data sizes range from 1 MB per process to 1024 MB per process with weak scaling.



(a) Scaling of Aggregate Write Bandwidth on Jaguar/Lustre.



(b) Scaling of Per-Writer Write Bandwidth on Jaguar/Lustre.

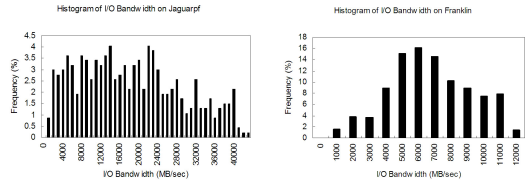
Fig. 1. Illustration of Internal Interference Effect

Measurements clearly demonstrate the performance effects of internal interference. In Figure 1(b), per-writer write bandwidth consistently decreases with an increasing number of writers, and Figure 1(a) reveals that eventually, the increase in aggregate performance due to an increased total number of writers is dwarfed by the losses in individual writer performance caused by contention. This holds for all cases other than those in which output benefit from the caches associated with storage targets, i.e., with 1 MB writes. Aggregate bandwidth peaks with a per-writer data size of 8 MB, then begins to decrease at the scale of 8192 writers (the ratio of writer vs. OST being 16:1); for all other data sizes, aggregate write bandwidth begins to decrease at the scale of 2048 writers (4 writers per OST). The effects are amplified at large scales. With per-writer data size equal or larger than 128MB, the aggregate write bandwidth degrades by 16%-28% when scaling from 8912 to 16384 writers. For Sandia's XTP, we did not observe substantial bandwidth degradation except that there is a $< 5\%$ reduction in write bandwidth for the large data sizes (512MB or 1024MB per writer) when scaling IOR from 512 to 1024 writers. This can be attributed to the XTP machine's relatively small size limiting the contention among concurrent writers and/or the design of PanFS.

2) External Interference: Tests are run on all three machines to demonstrate the effects of external inter-

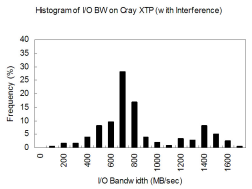
TABLE I
IO PERFORMANCE VARIABILITY DUE TO EXTERNAL INTERFERENCE

Machine	Number of Samples	Avg. IO Bandwidth (MB/sec)	Std. Deviation	Covariance
Jaguar	469	1.78e+4	1.07e+4	60.09%
Franklin	2581	6.22e+3	2.50e+3	40.22%
XTP(with Int.)	400	7.89e+2	3.44e+2	43.68%
XTP(without Int.)	320	1.44e+3	1.28e+2	8.86%

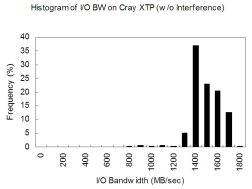


(a) Jaguar/Lustre

(b) Franklin/Lustre

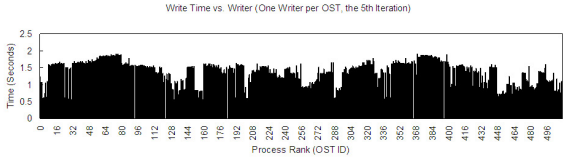


(c) XTP/PanFS (with Int.)

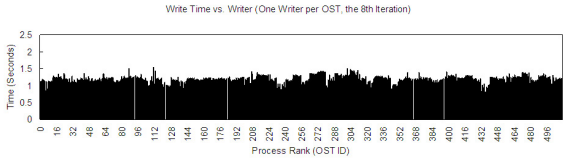


(d) XTP/PanFS (without Int.)

Fig. 2. IO Performance Variability due to External Interference



(a) Test 1



(b) Test 2

Fig. 3. Illustration of Imbalanced Concurrent Writers

ference on IO performance. Specifically, hourly IOR tests are launched where each test is configured with 512 writers using POSIX-IO, one file per writer, and one process per storage target. Performance results for these tests have a total of 469 samples of IO actions. Over the past two years, similar experiments have been conducted at NERSC on Franklin using 80 writers, with results from those tests accessible through NERSC's online performance monitoring data repository. The experiments we conduct on Sandia's Cray XTP differ because XTP is not a production machine. Here, tests are run in two controlled ways: the first runs a single IOR program with 512 writers using POSIX-IO and one file per writer (referred to as "XTP without Int."); the second launches two IOR programs at the same time, thereby emulating the presence of multiple simultaneous workloads(referred to as "XTP with Int.").

Table I summarizes experimental results, and Figure II-2 shows the histograms of IO bandwidth based on the performance data collected. It is clear that in busy production environments like Jaguar and Franklin, IO variability can be substantial, ranging from 40%-60%. On Sandia's Cray XTP, even a moderate degree of sharing (i.e., two simultaneous IOR jobs) can cause IO performance variations of up to 43%.

To better characterize the extent of interference, we define the *imbalance factor* of each IO action to be the ratio of the slowest vs. fastest write times across all writers. Consider two separate samples from the external interference tests for 128 MB per process on Jaguar. Figures 3(a) and 3(b) show the individual write times for each process for these two tests, respectively. Test 2 took place only 3 minutes later than Test 1. Apparent from these tests is the dynamic and potentially transient nature of external interference, resulting in write times that are much more evenly distributed among all concurrent writers in Test 2 than those in Test 1. In Test 1, an imbalance factor of 3.44 separates the minimum and maximum time spent performing IO. For Test 2, this factor is reduced to 1.86. Interestingly, even for the latter relatively smaller imbalance factor, nearly twice as much data could be written to the faster storage target than to the slower one.

To summarize, we observe a significant imbalance in terms of fastest vs. slowest writes in all IO tests run in our experiments with an overall average imbalance factor of 7.12. Since overall write time is determined by the slowest writer, the purpose of the adaptive IO methods presented in this paper, then, is to mitigate the performance impact of these 'slow' writers.

3) Alternatives to Adaptive IO: One possible way to reduce the effects on applications of IO performance variability is to decouple IO from application actions through the use of asynchronous IO. Unfortunately, given the large volumes of output generated by typical petascale applications, asynchronicity is limited by the total and limited amounts of buffer space available on the machine, which typically extends to only one or at most a few simulation output steps. Such 'near-synchronous' IO, therefore, still causes applications to block on IO when IO performance is consistently too low. Unfortunately and as evident from the experi-

mental evidence presented above, consistently low performance is a natural outcome of internal or external interference.

Data staging [1], a second potential solution to IO performance variability, also has limited applicability. To explain, data staging moves output from a large number of compute nodes to a smaller number of staging nodes before writing it to disk. However, the total buffer space available in the staging area is limited, thereby limiting the achievable degree of asynchronicity. Further, large staging areas and/or multiple staging areas concurrently used by multiple applications will still lead to internal or external interference. Data staging, therefore, can help with interference issues, but does not directly address them. In fact, our ongoing work is integrating adaptive IO even into the data staging software we are deploying on Jaguar.

Another approach to reducing internal interference is to split output into a collection of files to 'match' the parallel file system being used. In the case of Jaguar and its Lustre FS, for instance, splitting output into 5 parts would enable an application to take full advantage of the entire file system's resources, thereby providing at least a reasonable guarantee of achieving required performance during some normal, productive, busy time. This helps alleviate internal interference, but does not solve it nor does it address external interference.

In summary, the use of asynchronous IO, data staging, and/or target-specific mitigation methods may reduce the effects of IO performance variability on applications, but does not address its root problems. Because of these facts and the substantial performance variability in the storage system, adaptive IO continuously observes the storage system's performance to configure output in a way that transparently 'best' matches its static *and* dynamic characteristics.

4) Summary and Discussion: Experimental results shown in this section demonstrate the existence of internal and external interference on three different machines and with two different file systems. Interference (1) negatively impacts the scaling of IO performance, and perhaps more importantly, (2) introduces substantial IO performance variations that make it difficult to accurately predict and then properly allocate the amounts of time needed for performing IO. IO performance variations are shown to be the common rather than the uncommon case, particularly in production environments. This holds both the for POSIX-IO measurements reported above and for tests that use MPI-IO (not reported, for brevity), where for all cases, MPI-IO results show the same trends, but with inferior performance. We conclude, therefore, that internal and external interference are inherent and performance-limiting properties of petascale file systems.

III. Software Architecture and Implementation of Adaptive IO

Adaptive IO is implemented in the context of the ADIOS IO middleware [33], [28]. Specifically, adaptive IO is realized as an optional set of techniques bundled into a new IO method.

A. MPI-IO

The MPI-IO transport method was developed as one of the first options offered by ADIOS. Its common use has resulted in several optimizations, leading to excellent peak IO performance seen on Jaguar and its Lustre file system [35]. Substantial performance advantages are derived from limited asynchronicity, by buffering all output data on compute nodes before writing it (if possible). Additional optimizations in certain variants of the base ADIOS transport are tied to the Lustre file system used by many HPC codes. As a result, ADIOS and its MPI-IO base transport constitutes a high performance, well-tuned set of IO abstractions against which adaptive IO can be tested and evaluated.

B. Adaptive IO

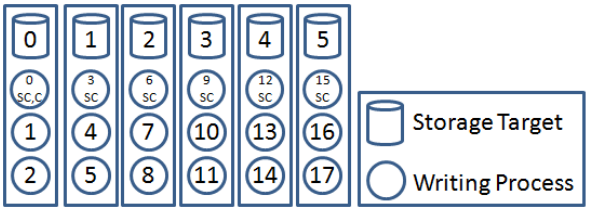


Fig. 4. Adaptive IO Organization

Figure 4 depicts a sample configuration. When using the adaptive IO method, middleware enhances the output actions taken by these processes in ways that ascribe to them three different roles, as illustrated in the figure: (1) the numbered circles represent process IDs of writers participating in the output, (2) some of these processes carry out additional actions, acting as sub-coordinators (SC) for a set of writers and a storage target, represented by the vertical boxes in the figure, and (3) one process plays the distinct coordinator (C) role for the entire set of writers. The coordinator and writers only communicate with the sub coordinators, never directly with each other. This isolates the messaging reducing the message load on any particular part of the system. We note that different sub coordinators write to different files, since this is how we can control mappings to certain storage targets. We purposely enhance writers with roles rather than implementing coordinators and sub-coordinators separately from writers. This avoids using additional processes and having to tightly

synchronize their coordination actions with the writing actions of separated writer processes. Since process IDs are typically assigned sequentially to cores in a node, grouping them as illustrated reduces the network contention on the node due to simultaneous writing from the same node, but different cores. Finally, by placing the coordination/sub coordination roles into the first process in each group, they can each focus on management after completing their writes instead of possibly being reassigned adaptively to a different target (file). This choice also avoids any delays in messaging due to the writer role for the process being busy while the coordinator is attempting to start an adaptive writer for this group.

The software architecture chosen for adaptive IO scales to the numbers of writers present in petascale machines like Jaguar and beyond. Specifically, based on the current state of the XT5 partition of Jaguar with approximately 225,000 processing cores and with the current 672 storage targets in the attached Lustre scratch system, this means each sub-coordinator is responsible for, at most, 335 processes. The coordinator is only responsible for the sub-coordinators, giving it 672 processes to manage. Even at the extreme scale of the Jaguar machine, these numbers are manageable and leave room for growth. An additional layer of coordination or distributed or partial coordination would further improve scalability, at the costs of additional messaging and thus, coordination overheads. Some insights on these tradeoffs are present in prior work on larger-scale management architectures for the enterprise domain [26], [27].

We next explain in more detail the precise actions taken by processes in different roles.

1) Writers: The details of this role are described in Algorithm 1. To ensure a consistent flow of data to storage, file indexing information is transferred separately and after writing is complete, so that this additional metadata transfer can take place concurrently with another process writing to storage.

Algorithm 1 Writer Process

```
1: Wait for message (target, offset)
2: Build local index based on offset
3: Write data
4: Send WRITE_COMPLETE to triggering SC
5: if triggering SC ≠ target SC then
6:    Send WRITE_COMPLETE to target SC
7: end if
8: Send local index to target SC
```

2) Sub-Coordinator (SC): Communications between the sub-coordinator(s) and coordinator constitute the major elements of the adaptive IO implementation. The details are described in Algorithm 2.

Algorithm 2 Sub-Coordinator Process (SC)

```
   while not done and missing_indices ≠ 0 do
2:   Signal next waiting writer to write
     Wait for message
4:   if message = WRITE_COMPLETE then
       if source is one of mine, but target is not me
       then
6:       Send adaptive WRITE_COMPLETE to C
       end if
8:     if source is one of mine and target is me then
         Save index size for index message
10:      missing_indices++
       end if
12:    if all writers completed then
         Send WRITE_COMPLETE to C
14:    end if
     end if
16:  if message = INDEX_BODY then
       Save for index for local file
18:    missing_indices–
     end if
20:  if message = ADAPTIVE_WRITE_START
     then
       if no waiting writers then
22:      Send WRITERS_BUSY to C
       else
24:      Signal writer with new target and offset
       end if
26:  end if
     if message = OVERALL_WRITE_COMPLETE
     then
28:    done = true
     end if
30: end while
   Sort and merge the index pieces for file index
32: Write the index
   Send the index to C
```

3) Coordinator (C): The coordinator role is generally idle until the late stages of IO when sub coordinators message their completion. As completions arrive, the coordinator begins to obtain a global view of the relative performance of storage targets. Given this view, it then attempts to shift work from busy (i.e., slower) to less loaded (i.e., faster) storage targets. This continues until all work has been completed, at which point it signals the completion of the composite write operation so that the local indices can be created and a global, master index formed. Adaptive writing requests are spread evenly among the sub coordinators to spread

out the accelerated completion of the write rather than pushing sub coordinators to completion one at a time. The sub coordinators are tracked as either *writing*, the initial state for the output operation, *busy*, indicating all processes have been scheduled so no adaptive writes are possible, or *complete* indicating that all writers have completed and this file is available for adaptive writing use. The details are described in Algorithm 3.

Algorithm 3 Coordinator Process (C)

```
    while any SC state ≠ complete or adaptive write
    request outstanding do
      Wait for message
 3:   if message = WRITE_COMPLETE then
        if this was an adaptive write then
          Request adaptive write by next writing SC
 6:     end if
        if this is an SC completing then
          Set SC state to complete
 9:       Note final offset
          Request adaptive write by next writing SC
        end if
12:   end if
      if message = WRITERS_BUSY then
        Set SC state to busy
15:     Request adaptive write by next writing SC
      end if
    end while
18: Send OVERALL_WRITE_COMPLETE to all SC
    Gather index pieces
    Merge into global index with local file information
21: Write global index file
```

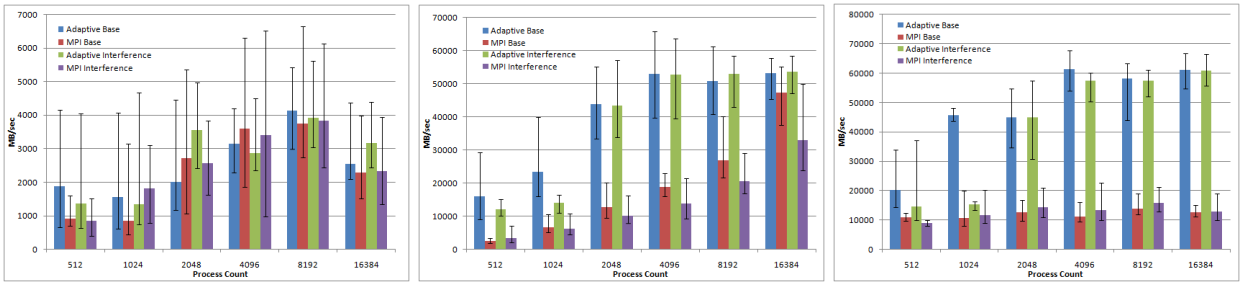
This adaptive mechanism scales according to the number of storage targets rather than the number of writers. The coordinator is only involved in the process once the bulk of writers are complete. Then, the largest number of simultaneous adaptive requests is strictly limited to $SCcount - 1$ as at most one write will be active for any file at one time. A larger pool of writers will only serve to keep the distributed, independent sub coordinators busy longer without affecting the coordinator with any additional simultaneous work. Adaptive IO has been fully implemented and tested, with the exception of the global indexing phase. In the interim, we use a automatic, systematic search of the index in each file for particular data of interest. The inclusion of the data characteristics [35] aid this search by enabling quickly searching for both the content as well as the logical 'location' of the data of interest. Also note that the Adaptive IO configuration shown in this section can be generalized, at the consequent cost of additional code complexity. For instance, one might use 2 or 3 simultaneous writers per storage location and/or multiple storage locations per sub coordinator. We have not experimented with these generalizations.

IV. Experimental Evaluation

To evaluate the performance of adaptive IO, all tests are performed on the XT5 partition of the Jaguar system at Oak Ridge National Laboratory (see Section II for detailed machine configuration). Tests aim to understand how different per process sizes of data perform with adaptive vs. non-adaptive IO, using two production petascale codes: (1) an IO kernel for the Pixie3D MHD simulation is run at 3 different per process data sizes; (2) the full XGC1 fusion code is run using a single per process data size. The ADIOS [33] layer is used to switch between the MPI-IO and the adaptive transport methods described in Section III.

The four output size sets of tests demonstrate the performance ranging from 2 MB/process up to 1024 MB/process. Tests are run with different process counts from 512 to 16384 processes against 160 OSTs for MPI, the maximum allowed for 1 file, or 512 OSTs for adaptive. The 512 OST selection for adaptive is chosen to simplify the discussion of ratios of writers to storage targets. The adaptive approach has been successfully tested with 672 storage targets with no penalties compared with the 512 storage targets measurements presented here. The tests are first run under normal system conditions with whatever other simultaneous jobs happen to be running. A second set of runs are performed with artificial interference introduced in an attempt to show the performance under a more heavily loaded file system. These results are then analyzed to show the performance of the different IO approaches. To ensure accurate measurements, an explicit 'flush' is introduced prior to the file close operation for both the MPI and the Adaptive transport methods. For all cases, at least five samples are generated and included. Where possible, additional samples are included as well to strengthen the numbers. In all cases, the times reported only include the actual write, flush, and file close operations to remove the variability due to the metadata server.

External interference is introduced through a separate program that continuously writes to a file striped across 8 storage targets during the runtime of the interference test cases. A stripe count of 8 is selected to reflect two applications writing using the default stripe count of 4 configured for the file system. Three processes each write 1 GB continuously to a single storage target, for a total of 24 processes.



(a) Small Data (2 MB/process) (b) Large Data (128 MB/process) (c) Extra Large Data (1024 MB/process)

Fig. 5. Pixie3D IO Performance

A. *Pixie3D*

Pixie3D [11] is a 3-Dimensional extended MHD (Magneto Hydro-Dynamics) code that solves the extended MHD equations in 3D arbitrary geometries using fully implicit Newton-Krylov algorithms. Pixie3D employs multigrid methods in computation and adopts a 3D domain decomposition. The output data of Pixie3D consists of eight double-precision, 3D arrays. The tested configuration consists of three different sized runs, named *small*, *large*, and *extra large*. The small run uses 32-cubes, large uses 128-cubes, while extra large uses 256-cubes. These cubes represent the per process, per variable size of the data. Overall, the small run generates 2 MB/process, large generates 128 MB/process, and extra large generates 1 GB/process. Weak scaling is employed.

The first set, shown in Figure 5(a), use the small data model for Pixie3D. With this model, the 2 MB/process, even at the 16384 process level, never comes close to the 2 GB cache size for the storage target (32×2 MB). Given that, in general, the adaptive approach does well. For example, at both 8192 and 16384 processes, the adaptive approach is 10% better on average for base performance. For the interference tests, 8192 processes for adaptive is 3% better on average while the 16384 processes test came to about 35% better. This small data model is maybe 10% of a typical data size for an application like the S3D [13] combustion simulation or the Chimera [36] astrophysics code. Interestingly, although these data volumes are small, as process counts increase, the adaptive approach can still pay off.

The second set, shown in Figure 5(b), use the large data model. This model consists of 128 MB/process and it quickly overcomes any caching advantage the storage targets may provide. It has consistently better performance both on average and at a maximum. The improvements range from 1% to more than 350% for the base case and 62% to more than 430% for the interference case. This 128 MB/process data size is comparable to what many of the fusion codes generate on a per process basis, such as GTC [24]. Another way to look at this data is considering a hybrid MPI/OpenMP setup. In this case, we divide the 128 MB by the number of OpenMP threads to find out what the per process data size would be. For Jaguar's 12 cores per node, this yields approximately 10 MB, or about the size of smaller S3D and Chimera runs.

The last set, shown in Figure 5(c), use the extra large data model. Although there are 3.2x more storage targets used for the adaptive approach, it is about 4.8x faster than the non-adaptive one! Once the adaptation can play a role, i.e., there are a few more processes than storage targets, there is a consistently $> 300\%$ performance improvement for both the base and interference tests. We note, however, that this data model is large even by fusion simulation standards, but we use it because of the growth in per node core counts on future platforms, likely resulting in hybrid MPI/OpenMP codes with larger per-node output.

B. *XGC1*

The XGC1 [12] code is a fusion gyrokinetic Particle In Cell code that uses realistic geometry to understand the physics on the edge of the plasma in a fusion reactor, such as ITER. These tests are performed using a configuration that generates 38 MB per process and weak scaling is used. While 38 MB per process is smaller than the largest runs for XGC1, it is still a representative size for a production run.

The performance of XGC1, shown in Figure 6, falls between that of the Pixie3D small and large data models, as would be expected. In this case, 38 MB/process is not uncommon for many scientific codes beyond XGC1, such as larger S3D runs. Adaptive IO shows clear advantages. For example for all of the tests, the performance improvement ranges from 30% to greater than 224%.

C. *Additional Insights and Discussion*

Adaptive IO benefits from 'locality-awareness', referring to the fact that when outputs are written, there are less vs. more 'busy' areas of the file system, due

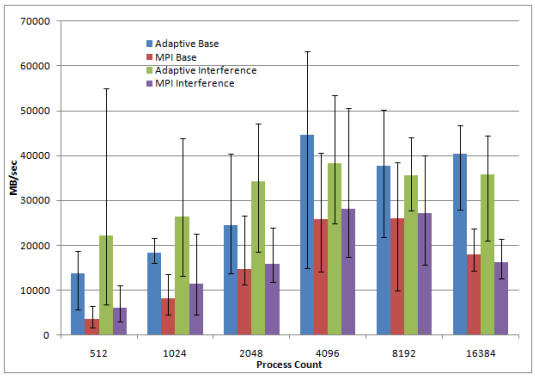
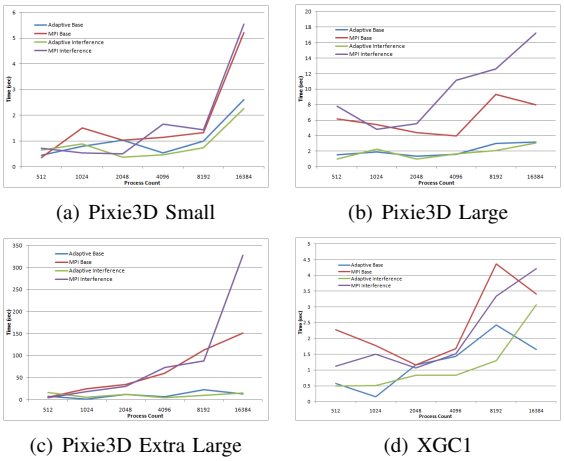


Fig. 6. XGC1 IO Performance (38 MB/process)



(a) Pixie3D Small (b) Pixie3D Large

(c) Pixie3D Extra Large (d) XGC1

Fig. 7. Standard Deviation of Write Time

to external and/or internal interference. Measurements and evaluations appearing in this paper substantiate that fact, and they also refine our earlier reports in which we note variability in IO performance [32]. We further substantiate these statements by next showing that adaptive IO typically reduces the IO performance variability experienced by applications.

For both Pixie3D and XGC1, once the process count reaches some small multiple of the storage target count, e.g., 4, the adaptive approach offers higher and more consistent performance. The graphs in Figure 7 show the standard deviation of the write times for each of the 4 cases measured. Here, the absolute numbers are less important than the fact that for all cases, once the caches on the storage targets start to be taxed, adaptive IO reduces variability. In some cases, such as in Figure 7(c), the difference is quite large.

Adaptive IO manages these variations by taking advantage of the imbalance factor noted in Section II to dynamically shift work from slower areas of the file system to faster ones. A potential issue with using adaptive IO, however, is that it requires additional files for output. Specifically, the number of output files is a function of the file system size rather than the process count in the run. By using the global index, access to any data can be performed using a single lookup into the index and then a direct read of the value(s) from the appropriate data file(s), sometimes resulting in improved performance [45] compared to the use of single storage formats. Considering that output sets are generally treated as a unit and that the number of files is a function of the number of storage targets rather than the number of processes, we believe the use of additional files does not strongly impact the ability of the scientist to manage the generated data.

V. Related Work

Parallel file systems offer high levels of performance for HPC applications, including Panasas [44], PVFS [48], Lustre [10], and GPFS [49], all of which provide POSIX-compliant interfaces. As stated earlier, there remain certain performance challenges, however. These systems aim to provide general purpose, multi-user file services, which as a goal, is somewhat orthogonal with a single user's desire to receive substantial IO resources and then, to optimize how these resources are used on behalf of that user. Adaptive IO provides such complementary functionality.

Current work on log-based file systems [46], [8] has improved write performance for checkpoints, but at the potential cost of reduced read performance. PLFS [45] has demonstrated that read performance does not suffer when performing a restart-style read of all of the data, but interference effects have not yet been addressed.

LWFS [42] breaks POSIX requirements in order to better suit client needs, but does not address the internal nor external sources of interference created by the shared file system. The Google File System [17] is focused on high aggregate throughput, but is not concerned with maximizing per client performance. Closer to our work is that of Gulati and Varman [18], who provide for scheduling IO operations, but their focus is on using caches to improve read performance, and they do not address the cases where the data is far larger than total cache space.

Using middleware to manage IO to improve performance is not new to HPC. MPI-IO introduced the ADIO layer [21] as a way to install system-specific optimizations of the general MPI-IO implementation. Many optimizations are possible and have been handled at this layer, such as custom drivers for Lustre or PVFS. Collective IO, also handled in this middleware layer, attempts to perform a level of data-size driven adaption by aggregating small writes into single, larger writes to obtain greater performance, but it does not address the issue of large writes from all of the processes, nor does it address interference problems.

At the slightly higher layer of IO APIs, the issue has largely been sidestepped. Both HDF-5 [19], and therefore NetCDF4 [54], and PnetCDF [39] have ceded control of this detail to the underlying IO layer, typically using MPI-IO. Some work has been done by the PnetCDF team on 'subfiling' [14] to try to address the need to decompose the output to gain greater parallelism. They also did not address the transient performance issues of external interference.

Data staging efforts have primarily focused on reducing data read times in HPC [37], in grid environments [15], [43], and for mobile applications [50]. More recent work has used data staging for enhancing write performance [40], but at a cost of additional compute resources. Demonstrated on a BlueGene/P with dedicated IO forwarding nodes, the IO Forwarding Scalability Layer [4] aggregates requests to reduce contention on the IO system to manage internal interference for writing but does nothing to manage external interference effects. To soften the cost of the additional resources for data staging, DataTap [3] provides data staging-like functionality, but provides much more significant in transit data processing features. DataTap has worked extensively to manage the IO effectively using several scheduling techniques [2].

Some results for the ADIOS *stagger* IO approach were reported at the 2009 Cray User's Group [32]. Stagger addressed internal interference and exposed the magnitude of the transient external interference. Since these results were presented, the number of cores was increased by 50% and the connection to the file system was adjusted because it is now being shared as the primary scratch space across most HPC resources at ORNL. These changes in system configuration make managing IO performance vairability even more challenging and motivate us to explore adaptive IO mechanism in this paper.

Adaptive approaches have been applied to IO systems in the past. Shared use of enterprise shared storage systems is considered in [55]. The goal is to maintain some level of quality of service for all competing applications, but there is no consideration of balanced IO loads for single applications across multiple storage targets. More recent work investigates pre-fetching and job scheduling [53], integrated with the job scheduler for an HPC resource. The goal is to reduce application load and start up times by pre-staging input data for read-intensive workloads. Also related to our work is the OPAL ADIO library [58] for MPI-IO. It attempts to manage IO based on the disk system itself, but it does not dynamically adjust where data is written. CA-NFS [7] pursues goals similar to ours, but it does not actively manage different storage areas, relies on asynchronous IO, and is limited to the mechanisms of NFS. Most closely related to our work is [52], which dynamically changes disk striping based on data sizes and on information about past usage. Its focus on repeat IO events means that it does not dynamically adjust file system usage across a single large output file, as done in our work.

Observations about the performance variability of shared HPC storage systems appear in [5], where NERSC researchers report that a small number of slow storage targets greatly increased total IO time. System logs and dedicated benchmarks [25] have been used to identify a variety of performance variations in HPC environment. Black box approaches [23] have been used to identify sources of performance problems related to storage or the network. Network sources for contention [9] have also been documented. Our observations about IO performance variability comply with these work, and our work explores active management to better handle IO performance variability.

VI. Conclusions and Future Work

Interference effects cause variable IO performance on both the shared file systems present at NERSC and ORNL, but also on machines with non-shared file systems, like Sandia's XTP. The adaptive IO methods presented in this paper mitigate such variability by continuously observing the storage system's performance and then balancing the workload being imposed. This substantially improves the IO performance seen by petascale codes, as demonstrated with numerous measurements and on multiple machines.

Our future work will examine the benefits of adaptive IO on systems beyond Lustre at ORNL, including Franklin at NERSC, PanFS on Sandia's XTP, and perhaps, GPFS on a BlueGene/P machine. Also of interest are other sources of variability, including that of metadata operations like file opens. Finally, there are likely more complex and/or state-rich methods for system adaptation, including those that take into account past usage data.

VII. Acknowledgements

This work was funded in part by Sandia National Laboratories under contract DE-AC04-94AL85000, the Department of Energy under Contract No. DEAC05-00OR22725 at Oak Ridge National Laboratory, the resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, a grant from NSF as part of the HECURA program, a grant from the Department of Defense, a grant from the Office of Science through the SciDAC program, and the SDM center in the OSCR office.

REFERENCES

[1] H. Abbasi, G. Eisenhauer, M. Wolf, and K. Schwan. Datastager: scalable data staging services for petascale applications. In *HPDC '09: Proceedings of the 18th international symposium on High performance distributed computing*, New York, NY, USA, 2009. ACM.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 39–48, New York, NY, USA, 2009. ACM.

[3] H. Abbasi, M. Wolf, and K. Schwan. Live data workspace: A flexible, dynamic and extensible platform for petascale applications. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 341–348, Washington, DC, USA, 2007. IEEE Computer Society.

[4] N. Ali, P. H. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. B. Ross, L. Ward, and P. Sadayappan. Scalable i/o forwarding framework for high-performance computing systems. In *CLUSTER*, pages 1–10, 2009.

[5] K. Antypas and A. Uselton. Mpi-i/o on franklin xt4 system at nersc. In *Cray User's Group 2009*. Cray User's Group, 2009.

[6] J. Axboe. Linux block io–present and future. *Proceedings of the Ottawa Linux Symposium 2004*, 2004.

[7] A. Batsakis, R. C. Burns, A. Kanevsky, J. Lentini, and T. Talpey. Ca-nfs: A congestion-aware network file system. In *FAST*, pages 99–110, 2009.

[8] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. Plfs: A checkpoint filesystem for parallel applications. *SC Conference*, 0, 2009.

[9] A. Bhatelé and L. V. Kalé. Quantifying Network Contention on Large Parallel Machines. *Parallel Processing Letters (Special Issue on Large-Scale Parallel Processing)*, 19(4):553–572, 2009.

[10] P. J. Braam. Lustre: a scalable high-performance file system, Nov. 2002.

[11] L. Chacón. A non-staggered, conservative, $\nabla \dot{s} B \rightarrow = 0$, finite-volume scheme for 3D implicit extended magnetohydrodynamics in curvilinear geometries. *Computer Physics Communications*, 163:143–171, Nov. 2004.

[12] C. S. Chang and S. Ku. Spontaneous rotation sources in a quiescent tokamak edge plasma. *Physics of Plasmas*, 15(6):062510, 2008.

[13] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. *Computational Science & Discovery*, 2(1):015001 (31pp), 2009.

[14] A. Choudhary, W. keng Liao, K. Gao, A. Nisar, R. Ross, R. Thakur, and R. Latham. Scalable i/o and analytics, 2009.

[15] H. Dail, H. Casanova, and F. Berman. A decoupled scheduling approach for the grads program development environment, 2002.

[16] W. Frings, F. Wolf, and V. Petkov. Scalable massively parallel i/o to task-local files. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, New York, NY, USA, 2009. ACM.

[17] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[18] A. Gulati and P. J. Varman. Scheduling multiple flows on parallel disks. In *HiPC*, pages 477–487, 2005.

[19] HDF-5. http://hdf.ncsa.uiuc.edu/products/hdf5/.

[20] Y. Hua, H. Jiang, Y. Zhu, D. Feng, and L. Tian. Smartstore: a new metadata organization paradigm with semantic-awareness for next-generation file systems. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[21] M. P. Interface. Mpi-2: Extensions to the message-passing interface, 1996.

[22] S. Iyer and P. Druschel. Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous i/o. *SIGOPS Oper. Syst. Rev.*, 35(5):117–130, 2001.

[23] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan. Black-box problem diagnosis in parallel file systems. In *FAST*, 2010.

[24] S. Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, and R. Samtaney. Grid -based parallel data streaming implemented for the gyrokinetic toroidal code. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 24, Washington, DC, USA, 2003. IEEE Computer Society.

[25] W. T. C. Kramer and C. Ryan. Performance variability of highly parallel architectures. In *International Conference on Computational Science*, pages 560–569, 2003.

[26] V. Kumar, Z. Cai, B. F. Cooper, G. Eisenhauer, K. Schwan, M. Mansour, B. Seshasayee, and P. Widener. Implementing diverse messaging models with self-managing properties using iflow. In *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 243–252, Washington, DC, USA, 2006. IEEE Computer Society.

[27] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf. Monalytics: Online monitoring and analytics for managing large scale data centers. *International Conference on Autonomic Computing (ICAC)*, June 2010.

[28] O. R. N. Laboratory. http://adiosapi.org/.

[29] A. Leung, I. Adams, and E. L. Miller. Magellan: A searchable metadata architecture for large-scale file systems. Technical Report UCSC-SSRC-09-07, University of California, Santa Cruz, Nov. 2009.

[30] A. Leung, A. Parker-Wood, and E. L. Miller. Copernicus: A scalable, high-performance semantic file system. Technical Report UCSC-SSRC-09-06, University of California, Santa Cruz, Oct. 2009.

[31] A. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller. Spyglass: Fast, scalable metadata search for large-scale storage systems. Feb. 2009.

[32] J. Lofstead, S. Klasky, M. Booth, H. Abbasi, F. Zheng, M. Wolf, and K. Schwan. Petascale io using the adaptable io system. Cray User's Group, 2009.

[33] J. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and C. Jin. Flexible io and integration for scientific codes through the adaptable io system (adios). In *CLADE 2008 at HPDC*, Boston, Massachusetts, June 2008. ACM.

[34] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Input/output apis and data organization for high performance scientific computing. In *In Proceedings of Petascale Data Storage Workshop 2008 at Supercomputing 2008*, 2008.

[35] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich io methods for portable high performance io. In *In Proceedings of IPDPS'09, May 25-29, Rome, Italy*, 2009.

[36] O. E. B. Messer, S. W. Bruenn, J. M. Blondin, W. R. Hix, A. Mezzacappa, and C. J. Dirk. Petascale supernova simulation with CHIMERA. *Journal of Physics Conference Series*, 78(1):012049–+, July 2007.

[37] E. L. Miller, R. H. Katz, and Y. H. Katz. Analyzing the i/o behavior of supercomputer applications. In *Eleventh IEEE Symposium on Mass Storage Systems*, pages 51–55, 1991.

[38] NERSC. http://www.nersc.gov/nusers/systems/franklin/monitor.php. obtained 24 march 2010.

[39] P. netCDF. http://trac.mcs.anl.gov/projects/parallel-netcdf.

[40] A. Nisar, W. keng Liao, and A. N. Choudhary. Scaling parallel i/o performance through i/o delegate and caching system. In *SC*, page 9, 2008.

[41] A. Nisar, W.-k. Liao, and A. Choudhary. Scaling parallel i/o performance through i/o delegate and caching system. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[42] R. Oldfield, L. Ward, R. Riesen, A. Maccabe, P. Widener, and T. Kordenbrock. Lightweight i/o for scientific applications. *Cluster Computing, 2006 IEEE International Conference on*, pages 1–11, 25-28 Sept. 2006.

[43] E. Otoo, F. Olken, and A. Shoshani. Disk cache replacement algorithm for storage resource managers in data grids. In *In: Proc. of the IEEE/ACM SC 2002 Conf. on Supercomputing. Los Alamitos: IEEE Computer Society*, pages 1–15, 2002.

[44] Panasas. http://www.panasas.com/.

[45] M. Polte, J. Lofstead, J. Bent, G. Gibson, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, M. Wingate, and M. Wolf. ...and eat it too: High read performance in write-optimized hpc i/o middleware file formats. In *In Proceedings of Petascale Data Storage Workshop 2009 at Supercomputing 2009*, 2009.

[46] M. Polte, J. Simsa, W. Tantisiriroj, G. Gibson, S. Dayal, M. Chainani, and D. Uppugandla. Fast log-based concurrent writing of checkpoints. In *Petascale Data Storage Workshop, 2008. PDSW '08. 3rd*, pages 1–4, Nov. 2008.

[47] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.*, 10(1):26–52, 1992.

[48] R. Ross, R. Latham, N. Miller, and P. Carns. A next-generation parallel file system for linux clusters. January 2004.

[49] F. Schmuck and R. Haskin. Gpfs: A shared-disk file system for large computing clusters. In *In Proceedings of the 2002 Conference on File and Storage Technologies (FAST*, pages 231–244, 2002.

[50] J. F. Shafeeq, J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanarayanan. Data staging on untrusted surrogates.

[51] H. Shan, K. Antypas, and J. Shalf. Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[52] H. Simitci and D. A. Reed. Adaptive disk striping for parallel input/output. In *IEEE Symposium on Mass Storage Systems*, pages 88–102, 1999.

[53] N. Tran and D. A. Reed. Automatic arima time series modeling for adaptive i/o prefetching. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):362–377, 2004.

[54] Unidata. http://www.hdfgroup.org/projects/netcdf-4/.

[55] S. Uttamchandani, L. Yin, G. A. Alvarez, J. Palmer, and G. A. Agha. Chameleon: A self-evolving, fully-adaptive resource arbitrator for storage systems. In *USENIX Annual Technical Conference, General Track*, pages 75–88, 2005.

[56] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam. Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments. *Physics of Plasmas*, 13(9):092505, 2006.

[57] J. Xing, J. Xiong, N. Sun, and J. Ma. Adaptive and scalable metadata management to support a trillion files. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11, New York, NY, USA, 2009. ACM.

[58] W. Yu, J. S. Vetter, and R. S. Canon. Opal: An open-source mpi-io library over cray xt. *Storage Network Architecture and Parallel I/Os, IEEE International Workshop on*, 0:41–46, 2007.