A Checkpoint of Research on Parallel I/O for High-Performance Computing

FRANCIELI ZANON BOITO, Institute of Informatics, Federal University of Rio Grande do Sul
EDUARDO C. INACIO, Department of Informatics and Statistics, Federal University of Santa Catarina
JEAN LUCA BEZ and PHILIPPE O. A. NAVAUX, Institute of Informatics, Federal University of Rio Grande do Sul
MARIO A. R. DANTAS, Department of Informatics and Statistics, Federal University of Santa Catarina
YVES DENNEULIN, INRIA, University of Grenoble Alpes

We present a comprehensive survey on parallel I/O in the high-performance computing (HPC) context. This is an important field for HPC because of the historic gap between processing power and storage latency, which causes application performance to be impaired when accessing or generating large amounts of data. As the available processing power and amount of data increase, I/O remains a central issue for the scientific community. In this survey article, we focus on a traditional I/O stack, with a POSIX parallel file system. We present background concepts everyone could benefit from. Moreover, through the comprehensive study of publications from the most important conferences and journals in a 5-year time window, we discuss the state of the art in I/O optimization approaches, access pattern extraction techniques, and performance modeling, in addition to general aspects of parallel I/O research. With this approach, we aim at identifying the general characteristics of the field and the main current and future research topics.

CCS Concepts: • **Computing methodologies** → *Parallel computing methodologies*; *Modeling and simulation*; • **Computer systems organization** → *Client-server architectures*; • **General and reference** → Surveys and overviews;

Additional Key Words and Phrases: Parallel file systems, high-performance computing, storage systems

ACM Reference format:
Francieli Zanon Boito, Eduardo C. Inacio, Jean Luca Bez, Philippe O. A. Navaux, Mario A. R. Dantas, and Yves Denneulin. 2018. A Checkpoint of Research on Parallel I/O for High-Performance Computing. *ACM Comput. Surv.* 51, 2, Article 23 (March 2018), 35 pages.
https://doi.org/10.1145/3152891

This research was accomplished in the context of the LICIA International Joint Laboratory. It received funding from the EU H2020 Programme and from MCTI/RNP-Brazil under the HPC4E Project, Grant Agreement No. 689772.
Authors' addresses: F. Z. Boito, Inria GIANT, Minatec Campus, 17 Avenue des Martyrs, 38000, Grenoble, France; email: francieli.zanon-boito@inria.fr; E. C. Inacio and M. A. R. Dantas, INE, Campus Reitor João D. F. Lima, Florianópolis, 88040-900, Brazil; emails: eduardo.camilo@posgrad.ufsc.br, dantas@ufsc.br; J. L. Bez and P. O. A. Navaux, Institute of Informatics, Av. Bento Gonçalves 9500, Porto Alegre, 90650-001, Brazil; emails: {jlbez, navaux}@inf.ufrgs.br; Y. Denneulin, IMAG, 700 Avenue Centrale, 38401, Saint Martin dHères, France; email: yves.denneulin@grenoble-inp.fr.

https://doi.org/10.1145/3152891

1 INTRODUCTION

Computing systems have a memory hierarchy where programs' data is stored and from where instructions are fetched for processing in the CPU. At the last level of this memory hierarchy resides a non-volatile storage device—such as a hard disk drive (HDD) or a solid state drive (SSD). File systems abstract the physical storage devices, allowing applications to make input and output (I/O) requests for portions of files.

Large scientific applications, such as weather forecast and seismic simulations, typically execute in cluster or massively parallel processing (MPP) architectures. In these supercomputers, the application's workload is separated across multiple processes, executing on different computing nodes. These processes often need access to shared files. *Parallel file systems* (PFSs) allow them to access shared files transparently—i.e., without knowledge of where these files are actually stored. Another key characteristic of PFSs is the use of multiple machines (servers) to store data. With this approach, data can be retrieved from the servers in parallel, an important concept for performance.

Regarding performance, the high-performance computing (HPC) field is today in its petascale era—a processing power of one quadrillion floating point operations per second. Nonetheless, there is a historic gap between processing and I/O, since the latter depends on slower devices such as memory, disks, and network (Patterson and Hennessy 2013). Because of that, applications that access large amounts of data often have their performance impaired by I/O.

For instance, recording every collision at the Large Hadron Collider (LHC) would require generating approximately one petabyte of data to the storage system per second. Even if filters are used to decrease the output volume, by the 2020s the LHC is expected to be dealing with exabytes of data (DOE 2013). Despite decades of research effort into providing high-performance parallel I/O, it continues to be a central issue on the path to exascale (DOE/NNSA 2014).

In this article, we present a survey of the parallel I/O research field in the HPC context. Our focus is the traditional I/O stack, which includes a POSIX PFS. Therefore, we do not discuss storage tools for grid and cloud environments, or for big data processing, such as HDFS (Tantisiriroj et al. 2011). Nonetheless, many of the presented techniques and discussions could be expanded to other storage systems.

Our contributions are twofold: first, we provide the basic concepts so readers who are unfamiliar with the subject can understand the concepts of parallel I/O, the main components involved, the common problems, and the techniques typically applied to achieve high performance. We believe this knowledge is useful to the whole scientific community, as applications often observe poor performance due to poor I/O design. Second, we aim at identifying current and future research topics in parallel I/O. We have focused on 5 years of publications from the field's most important conferences and journals. By exploring the research activity of the last 5 years, we aim at answering questions such as:

- —Has the amount of research effort put into the field grown over the past few years?
- —What are the main techniques HPC researchers resort to when working to improve I/O performance?
- —Which are the research topics this community is expected to put more effort in the next years?

Additionally, we work to characterize the research on parallel I/O for HPC: the most used systems and tools, what are the most active countries and institutions, general characteristics of publications in the field, and so on.

The remainder of this article is organized as follows: Section 2 explains the survey methodology, lists the considered conferences and journals, and presents some numbers about them. Section 3

provides a background for parallel I/O in the HPC context, discussing the main components and factors involved in achieving high performance. By the end of Section 3, surveyed data is presented to identify the most used tools for research. The state of the art in techniques to improve performance of PFS access is discussed in Section 4, followed by the state of the art in application access pattern extraction and performance modeling in Section 5. Section 6 discusses practical aspects of parallel I/O research, identifying general characteristics of the studied publications. Finally, Section 7 concludes this article by summarizing the presented information and listing the main topics for future research.

2 A SURVEY ON PARALLEL I/O FOR HPC

To get a representative picture of the state of the art in parallel I/O for HPC, we have made a selection of widely known, leading quality conferences and journals:

- —ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)
- —IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid);
- —IEEE International Conference on Cluster Computing (CLUSTER)
- —International European Conference on Parallel and Distributed Computing (Euro-Par)
- —USENIX Conference on File and Storage Technologies (FAST)
- —ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)
- —ACM International Conference on Supercomputing (ICS)
- —IEEE International Parallel & Distributed Processing Symposium (IPDPS)
- —IEEE Symposium on Massive Storage Systems and Technology (MSST)
- —The IEEE/ACM International Conference for High-Performance Computing, Networking, Storage and Analysis (SC)
- —*ACM Computing Surveys* (CSUR)
- —*Elsevier Journal of Parallel and Distributed Computing* (JPDC)
- —*Elsevier Parallel Computing* (ParCo)
- —*IEEE Transactions on Computers* (TOC)
- —*ACM Transactions on Computer Systems* (TOCS)
- —*ACM Transactions on Storage* (TOS)
- —*IEEE Transactions on Parallel and Distributed Systems* (TPDC)

We have defined a 5-year window for this analysis, covering publications between 2010 and 2014.[1] We went through all proceedings and issues inside the time window—a total of 5,629 publications—to identify relevant work by looking at title and abstract. Only full papers were considered. This activity aimed at avoiding "false negatives," i.e., a paper would be selected at the slight suspicion of relevance so no relevant work would be lost. This process resulted in 140 selected papers for further analysis (2.5%).

The analysis consisted of reading each article and answering a set of questions. Most of the prepared questions were answered by marking it with predefined "tags," such as "new file system," "I/O scheduling," or "simulation." Because of the selection approach, some of the selected papers were not in fact relevant for this survey. Therefore, they were later excluded from the analysis. In the end, 99 articles remained (1.7%).

[1] 2015 was not included because most of the analysis was conducted in the middle of the year, when not all proceedings and issues were available.

Table 1. Number of Selected Publications

	Conferences			Journals	
	Publications	Selected (%)		Publications	Selected (%)
ASPLOS	182	1 (0.5%)	**CSUR**	198	0 (0%)
CCGrid	300	9 (3%)	**JPDC**	630	1 (0.1%)
CLUSTER	179	11 (6.1%)	**ParCo**	257	1 (0.4%)
Euro-Par	306	3 (1%)	**TOC**	869	2 (0.2%)
FAST	115	5 (4.3%)	**TOCS**	54	0 (0%)
HPDC	109	10 (9.2%)	**TOS**	72	1 (1.4%)
ICS	179	6 (3.3%)	**TPDC**	1069	7 (0.6%)
IPDPS	579	12 (2.1%)			
MSST	135	5 (3.7%)			
SC	396	25 (6.3%)			
TOTAL	2480	87 (3.5%)	**TOTAL**	3149	12 (0.4%)

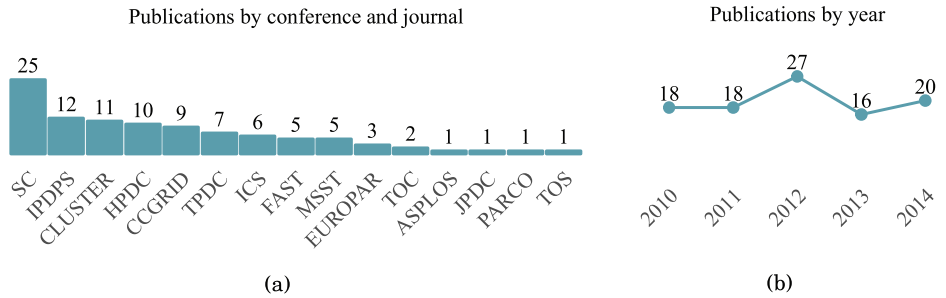

Fig. 1. Publications on parallel I/O separated by year and by form of publication.

Papers were considered relevant when discussing traditional parallel I/O in an HPC context, even if they do not discuss parallel I/O experiments. Moreover, techniques to characterize parallel application access patterns were also considered relevant, since the goal is often to provide information to parallel I/O optimization techniques. Table 1 presents the number of published and selected papers by conference and journal. It is important to notice that we have not found relevant articles in the *ACM Computing Surveys* journal, what motivates the present work. In fact, in our analysis, no survey was found.

From 99 surveyed papers, 87 (88%) are from conferences and 12 (12%) from journals. These numbers are further explored in Figure 1, which shows the number of selected publications by conference and journal [Figure 1(a)] and the number of publications by year [Figure 1(b)]. These numbers indicate that research on parallel I/O is more often published in conferences than in journals. This holds true even if we exclude the generic journals on computer science (CSUR, TOC, and TOCS), considering only the ones specialized in parallel and distributed systems and storage: TOS and TPDC, for instance, publish proportionally fewer pieces on the subject than most conferences.

Regarding field representativeness, i.e., how many of the papers published in a conference or journal are about parallel I/O for HPC, the most relevant conferences are, in this order: HPDC, SC, and CLUSTER. However, considering the absolute number of relevant publications, the most relevant conferences are SC and IPDPS.

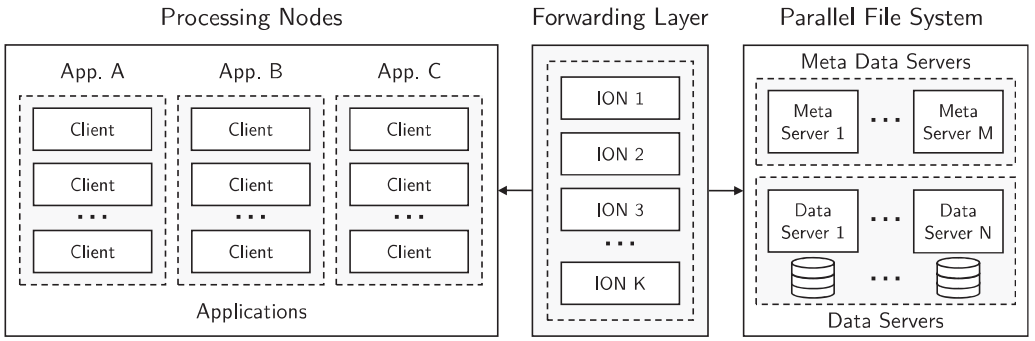

Fig. 2. Logical components involved when performing I/O to parallel file systems.

We can also notice from Figure 1(b) that 2012 was particularly productive regarding parallel I/O research, with approximately 50% more publications than the other studied years. In 2012, SC had ten relevant papers (the conference's average for the other years is approximately four). CLUSTER also presented a "parallel I/O peak" in that year. The next sections classify the surveyed articles according to their focus and proposed techniques.

3 THE PARALLEL I/O STACK

Files are accessed by applications through an interface that defines I/O operations like *open*, *write*, *read*, and *close*. These operations generate *requests* treated by the file system. On large-scale architectures, PFSs provide a shared storage infrastructure so applications can access remote files as if they were stored in a local file system. We call processes that access a PFS its *clients*.

Figure 2 shows an overview of the main components that affect I/O performance when using PFSs. They are discussed in Sections 3.1 to 3.4, providing the base concepts needed for the rest of this document. Section 3.5 presents some data gathered during our analysis, and concludes this part of the discussion.

3.1 Storage Devices

We start our discussion with storage devices, the last level of the considered I/O stack. Actually, storage in tapes is often the final level for data in a supercomputer. Nonetheless, archiving is a postmortem activity, and we focus on the main levels that affect performance perceived by applications when accessing a PFS.

HDDs have been the main storage device available for many years. They are composed of magnetic-surfaced rotating platters. Accessing data requires moving the head to the proper location, an operation known as *seek*. HDDs are known for presenting the best performance when accesses are done to sequentially positioned blocks instead of randomly because it minimizes seek time (Patterson and Hennessy 2013).

A popular solution for storage in HPC systems is the use of *redundant array of independent disks* (RAIDs), which combine multiple hard disks into a virtual unit for performance and reliability purposes. Data is distributed among the disks in fixed-size portions called "stripes," and can be retrieved in parallel for high performance. RAID performance is affected by the combination of stripe size and access size.

SSDs are a recent flash-based alternative to hard disks. Their advantages include higher bandwidth and less sensitivity to access sequentiality. Due to their internal organization, which allows for some parallelism, SSDs typically present better performance for large requests (Kim et al. 2012).

There is a growing adoption of SSDs, although their larger cost per byte still causes many PFS deployments in clusters to store data in hard disks.

In addition to the storage device's physical characteristics, the observed performance behavior also reflects characteristics from higher levels of the server's local I/O stack. Most HDDs and SSDs contain a small cache in hardware. Additionally, the operating system kernel has a cache to mask device access costs. Both caches typically perform *prefetching* and *read-ahead*, techniques that try to predict data that will be accessed by applications in the future to anticipate them. Therefore, random read accesses may perform worse than sequential ones because they do not fully exploit these mechanisms. These approaches can also be applied to PFS client caches, both for data and metadata. The next section discusses PFSs.

3.2 Parallel File Systems

PFSs are composed of two specialized servers: the *data server* and the *metadata server*. The latter is responsible for metadata, which is information about data such as size, permissions, and location among the data servers. To access data, clients must first obtain layout information from metadata servers.

As all basic file system operations involve metadata operations, metadata access scalability impacts the whole system. Some systems cache metadata on clients to accelerate this access. However, this technique brings the complexity of maintaining cache coherence, especially when a large number of clients is concurrently accessing the file system. Another way of improving metadata access performance is to distribute metadata among multiple servers, in a similar way to what is done with data itself. This is done on systems like PVFS[2] (Latham et al. 2004). Other systems, like Lustre (CFS 2002), decide not to distribute metadata to keep its management simple.

Files are distributed across data servers in an operation called *data striping*. Each file is divided into fixed size portions, called *stripes*, and the stripes are given to servers following a round-robin approach. A PFS's main characteristic is the possibility of retrieving stripes from different servers in parallel, increasing throughput. The striping configuration depends on the system's target applications. The retired Google File System (Ghemawat et al. 2003) employed a 64MB stripe size because its target applications performed very large sequential accesses only. PVFS, Lustre, and GPFS (Schmuck and Haskin 2002) have defaults between 64KB and 1MB.

Some systems apply *locking* on the servers to keep consistency in the presence of concurrent accesses. This is done by Lustre using stripe granularity, i.e., multiple clients are not allowed to access the same stripe concurrently. Other systems, like PVFS, leave the consistency to be treated by users for simplicity and performance.

To provide fault tolerance, some systems support replication of data and metadata. This is usually done by keeping mirrored servers and may have a performance impact to keep copies synchronized. On the other hand, having the same data on more than one server allows parallel access, potentially improving performance.

The abstraction of *objects* is often used to store PFS servers' portions of data, supporting object-based storage solutions. One example of such a case is Lustre, where data servers are called "object storage servers" (OSS). Another alternative is storing data in files through a local file system on the servers. When object-based storage is not available, objects are usually also stored as local files. Section 3.2.1 gives an overview of popular PFSs.

3.2.1 Popular Parallel File Systems. PVFS is an open-source PFS where servers run at the user level. PVFS provides two client interfaces: the UNIX API as presented by the client operating

[2]In this article, we use "PVFS" referring to both PVFS2 and OrangeFS, a recent branch of PVFS2.

system and MPI-IO, which is linked to a low-level PVFS API for access. PVFS supports distributed metadata, and metadata servers may be collocated with data servers.

The Lustre file system is another open-source PFS, implemented as Linux kernel modules. It has three main functional units: metadata servers (MDS), OSS, and clients. MDS nodes have one or more metadata target devices that store metadata in a local file system. OSS nodes store file data in one or more object storage target (OST) devices. The PFS capacity is the sum of the capacity of its OSTs.

The IBM GPFS is a commercial PFS designed for HPC and data-intensive applications. GPFS is founded upon a shared storage model, distributing and managing files in a shared storage system while providing a single namespace to all nodes.

The Panasas file system (Nagle et al. 2004) is divided into storage nodes and manager nodes. The former provide access to object storage devices and are accessed directly from clients during I/O operations. The latter manage the overall storage cluster, implement the distributed file system semantics, handle recovery of storage node failures, and provide an exported view of the file system via Network File System and Common Internet File System.

3.3 The I/O Forwarding Layer

The I/O forwarding technique aims at decreasing the number of clients concurrently accessing the file system servers by having some special nodes (often called *I/O nodes*) to receive the processing nodes' requests and forward them to the file system. The processing nodes may then be powered with a simplified local I/O stack to avoid its interference with performance. In this schema, the number of I/O nodes is typically larger than the number of file system servers, and smaller than the number of processing nodes (Vishwanath et al. 2010).

The I/O forwarding technique is applied in many of the current Top500[3] supercomputers. For instance, it was employed in the storage infrastructure of Tianhe-2, ranked as second in the Top500 list (November 2016). Tianhe-2's 16,000 computing nodes do not have a local I/O stack, but instead, all I/O operations are transferred to the 256 available intermediate I/O nodes. These nodes are powered with high-speed SSDs, and configurations for each file determine when data is transferred from the I/O nodes to the PFS (Xu et al. 2014).

The I/O forwarding idea has the advantage of providing a layer between application and file system. This layer can work to keep compatibility between both sides and apply optimizations such as requests reordering and aggregation.

3.4 PFS Clients

Applications may start their execution by reading data from previous executions or previous steps of the analysis. It is usual for simulations to have their execution organized as a series of timesteps. Each timestep evolves the simulated space in time. The execution often finishes by writing the obtained results, but I/O operations may also be generated at every given number of timesteps. Another common reason for applications to generate I/O operations is checkpointing. Some applications periodically write their state to files so execution can be easily resumed after interruptions.

We call the description of the I/O operations performed by an application its *access pattern*. There is no globally accepted taxonomy for patterns. In the literature, papers that provide some classification usually do it in the context of specific optimizations, considering only the aspects that are relevant to the proposed techniques.

[3]Top500: https://www.top500.org/lists/2016/11.

The most usual access pattern aspect is *spatiality*. It tells the location of requests into files: contiguous, distant by a fixed value, randomly positioned, and the like. Spatiality is an important aspect because of its direct impact on performance. This happens because, as discussed in Section 3.1, the storage infrastructure where servers store data has its performance affected by access sequentiality. The *temporal pattern* of the application is also usually considered. It describes how often the application generates requests, and especially if there are *bursts*, i.e., periods of the execution where the workload is heavier. Other important aspects usually considered are request size, number of generated/accessed files, intra-node concurrency, and operation (write or read).

3.4.1 I/O Libraries. PFSs usually deploy a client module in client nodes so they can view the remote folders as local and access them through the *POSIX* API, which defines standard I/O operations. Depending on the complex interaction between the different levels and on the system design choices, performance will be better for some access patterns than for others. Hence, achieving good performance depends on how well application accesses suit the used system. Nevertheless, this tuning between applications and systems is not easily achieved. First, PFSs do not have enough information to adapt to applications, as this information is usually lost through the I/O stack. On the other hand, tuning applications would require them to be developed considering a specific system, which would compromise their portability. Moreover, developers would need to know details about the target system performance behavior. Given these systems' complexity, this behavior is not easily analyzed.

One solution is the use of I/O libraries, the most popular being MPI-IO (Corbett et al. 1996). These libraries take charge of application I/O operations and have the power to perform optimizations to adapt their access pattern. High-level I/O libraries like HDF5 (The HDF Group 2016) and netCDF[4] (Li et al. 2003) also abstract I/O operations by allowing the definition of complex data types and file formats. These formats can be freely mapped to real files by these libraries, providing optimization opportunities. Section 4 discusses techniques to improve I/O performance.

3.5 Surveyed Data

There are already a few well-accepted PFSs, so research papers proposing new file systems are rare. From all the papers from this analysis, only one of them does so. Yi et al. (2014) propose a block-based PFS where metadata is stored physically separated from data—and clients are kept from accessing the storage device containing metadata—for reliability purposes.

As expected, most of the selected articles use a PFS (92 out of 99). Figure 3 presents some numbers on PFS usage. Figure 3(a) presents the number of papers that use each file system. The last bar—"other"—represents file systems that were not listed. Figure 3(b) to (e) detail the usage among papers presenting techniques for access pattern extraction and I/O optimization (for data or metadata access). In these four graphs, the "other" bars include GPFS and Panasas, as we focus on the two most used systems. The numbers do not add up to 92 because some articles discuss more than one file system, and hence are represented multiple times in the graphs. Similarly, data and metadata optimizations can be discussed in the same publication.

From the 10 most powerful supercomputers in the world, according to the November 2016 edition of the Top500 list, five use Lustre, two use solutions based on Lustre, two use their own custom file system, and one uses GPFS. Lustre importance in the HPC field can be confirmed in the graphs from Figure 3. Nonetheless, we can see that PVFS is also very popular as a research tool. PVFS

[4]In this article, and in the presented surveyed data, we use "netCDF" to refer to both the original library and the parallel version "PnetCDF."

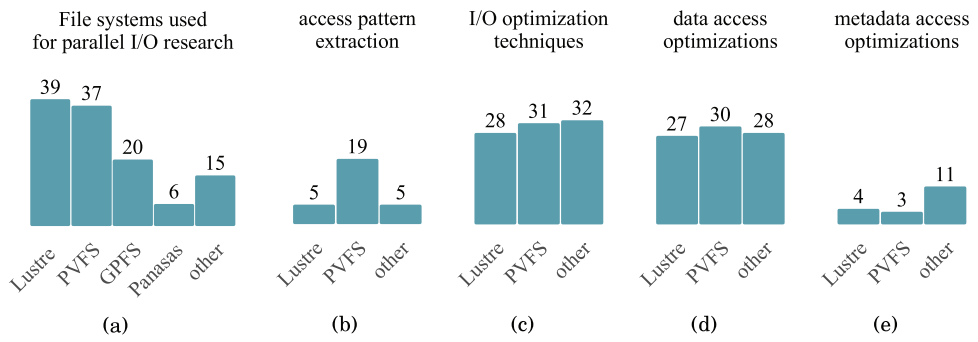

Fig. 3. Parallel file system usage among the surveyed publications—in general and separated by technique.

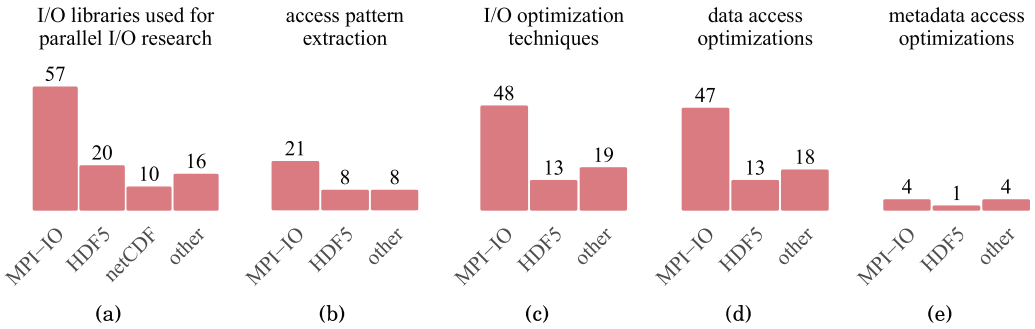

Fig. 4. I/O library usage among the surveyed publications—in general and separated by technique.

is the most used PFS to prototype and evaluate access pattern extraction and I/O optimization techniques.

Among metadata access optimization efforts, diverse and less known file systems are used. For this kind of work, researchers often choose the most convenient one in terms of ease to modify (which may be a "research toy" or even a simulator). This happens because changing the way a file system manages metadata access is a very critical modification, which may require adjustments in the whole system implementation. Hence, it makes sense to validate the optimization techniques before putting the effort to implement them in a more complex, but more popular, system.

From the analyzed publications, 70 (71%) use I/O libraries for their experiments. Figure 4 shows I/O library usage among the surveyed papers. Figure 4(a) shows the number of publications using each library, and Figure 4(b) to (e) focus on the two most used libraries while detailing usage among papers discussing access pattern extraction and optimizations. Similarly to the previous graphs, the "other" bar includes libraries that were not listed, and publications may count to multiple bars as they may use multiple libraries. We can see MPI-IO was used in most of the papers (57 out of 70 that use some library—81%). Over half of the I/O optimization implementations or evaluations have used MPI-IO (48 out of 79).

Among articles that present techniques for access pattern extraction (29), most of them (26) use some I/O library, 21 of them MPI-IO. I/O libraries are popular among these papers because their high-level abstraction provides more information about applications, and this information can be easily and transparently obtained.

When evaluating their techniques, researchers often use benchmarks to represent the many possible access patterns applications may present. From the research efforts considered in this survey,

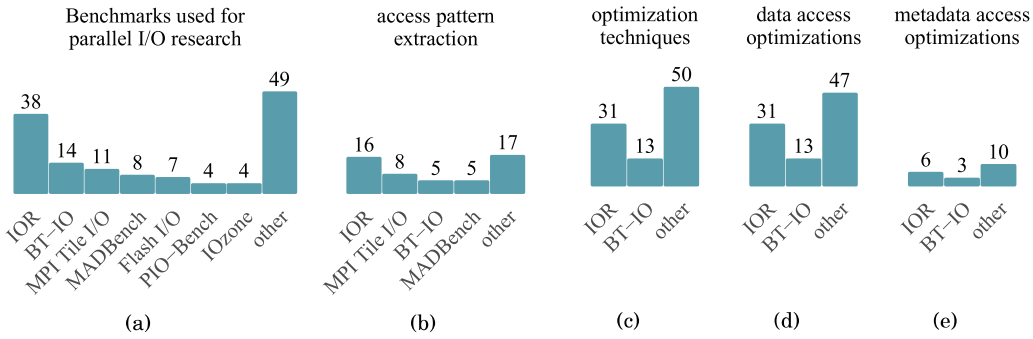

Fig. 5. Benchmark usage among the surveyed publications—in general and separated by technique.

75 (76%) use benchmarks for their experiments. Numbers on benchmark usage are presented in Figure 5. Figure 5(a) shows the number of publications using each benchmark, Figure 5(b) counts publications that discuss an access pattern extraction technique using each tool, focusing on the four most used ones. Figure 5(c) to (e) present the number of publications proposing data and metadata access optimization techniques, focusing on the two most used benchmarks. Again, the "other" bar includes benchmarks that were not listed, and publications may be represented multiple times in each graph as they may use multiple ones. The graphs show IOR was the most used I/O benchmark. One of the main reasons for its popularity in the field is the possibility of easily producing different access patterns by adjusting its parameters.

Nonetheless, "others" account for most publications (49 out of 75). During our analysis, benchmarks were classified among "others" mainly when they were customized microbenchmarks, designed to present a specific behavior or stress some component. It is usual for researchers to use well-known tools in addition to customized ones, tailored to mimic target applications characteristics. Customization is often needed because, despite the large number of available benchmarks, most of them present a regular behavior, well-structured spatial access pattern (mostly 1D strided), and synchronous operations. These characteristics, although usual, are not always enough to represent target scenarios.

Moreover, one problem that arises from the large number of available tools, in addition to customized ones, is the lack of standardization. A standard set of benchmarks and applications, as the role played by the NAS parallel benchmarks[5] in other fields of HPC, would facilitate the comparison between techniques and strengthen the presented validations. Such an effort towards standardization was made by the Parallel I/O Benchmarking Consortium[6] in the early 2000s. Nevertheless, our results do not indicate progress in this aspect.

Finally, 24 (24%) of the surveyed papers used checkpointing as a motivation for their work on parallel I/O.

4 TECHNIQUES TO IMPROVE PARALLEL I/O PERFORMANCE

Since performance depends on application access patterns, and some patterns are known to perform better than others, strategies to improve application performance often involve changing its access pattern to make it more suitable for the used system. This can be done on the server side,

[5] http://www.nas.nasa.gov/publications/npb.html.
[6] http://www.mcs.anl.gov/research/projects/pio-benchmark.

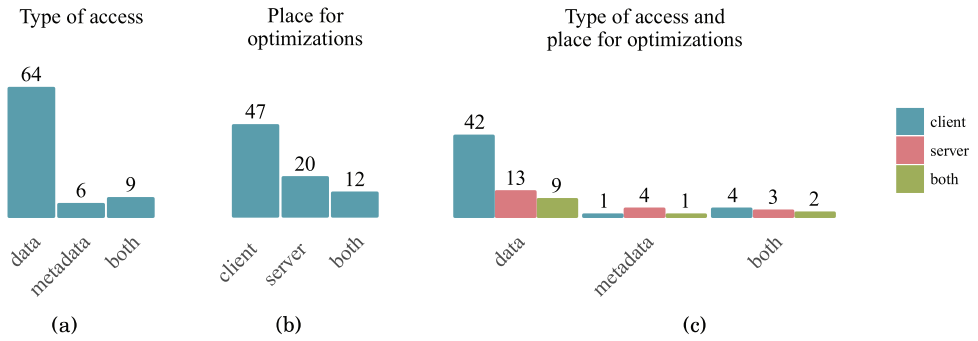

Fig. 6. Surveyed data on optimizations—type of access and place for optimizations.

mainly by modifying the file system; or on the client side, by changing applications, I/O libraries, APIs, and so forth.

From the studied publications, 79 (80%) propose optimizations for parallel I/O. Figure 6 provides some surveyed data on these techniques. Figure 6(a) presents the number of publications separated by the optimization focus (data or metadata access), Figure 6(b) presents publications separated by optimization place (server or client side), and Figure 6(c) integrates both aspects.

Among papers proposing optimization techniques, most focus on data access (64, 81%). Moreover, most techniques work on the client side only (47, 59%). We can see data access optimization happens mostly on the client side, while the most usual place for metadata access optimizations is the server side. This happens because metadata access is usually triggered by a simple request from the client, and the rest happens in the file system and depends on its metadata management approach. On the other hand, data access depends on applications' access patterns, and the access pattern can be adapted still on the client side, as previously discussed.

This section describes typical I/O optimizations techniques. Sections 4.1 to 4.5 discuss these optimizations, Section 4.6 presents more survey results about them, and Section 4.7 concludes this part of the paper.

4.1 Optimizations for Metadata Access and Small Files

Section 3.2 stated that distributing metadata storage among multiple servers is a strategy to improve performance and scalability for metadata access. This design choice brings the extra cost of managing this distribution to keep consistency.

A new hierarchical adaptive approach for metadata management is proposed by Hua et al. (2011). Their approach applies Bloom filters to route metadata accesses among the servers. Patil and Ganger (2011) propose a file system directory service called GIGA+, which distributes directories' entries across multiple servers using a decentralized hash-based indexing. Consistency of directory indexes in client caches is eventually maintained. Another metadata distribution design called IndexFS is proposed by Ren et al. (2014), with a table-based architecture that incrementally partitions the namespace on a per-directory basis. Their approach includes bulk namespace insertion for creation-intensive workloads. Xiong et al. (2011) also propose a distribution policy among metadata servers, in addition to a consistency policy. Their consistency policy focuses on allowing fast recovery in the presence of faults.

Metadata access may also become a bottleneck when the file system tree must be traversed for transferring data to permanent storage, visualization, post-processing, and so on. In this context, LaFon et al. (2012) propose three distributed algorithms—with no centralized control—for

traversing PFSs and performing file operations in parallel. Their algorithms are adequate for different scenarios. They use a randomized work-stealing scheduler to efficiently balance workload between the worker processes (idle workers "steal" from other workers' pending queues).

One important concern for metadata management is reliability, since the loss of a file metadata results in the file no longer being reachable. Therefore, a fault in one of the metadata servers may leave the system in a corrupted state, while a fault in a centralized metadata server may cause the loss of the whole file system tree. For this reason, metadata replication is often applied. Additionally, some systems keep metadata updates in a journal, where these modifications are only valid after committed through transactions (with guaranteed atomicity).

Chen et al. (2010b) propose an alternative approach to both replication and journaling, aiming at improving the performance of metadata access. They use the Paxos algorithm to build a coordination mechanism with low synchronization latency, where all replica servers simultaneously provide metadata read access. This mechanism decreases the impact of server failures and avoids the interruption of service.

Oral et al. (2010) improve the performance of *ldiskfs*, a variation of the *ext*3 file system used by Lustre in both data and metadata servers. Two approaches were applied: using external journaling devices to eliminate latency incurred by extra disk head seeks and a software-based optimization to aggregate commits.

In the block-based symmetric storage approach for PFSs, metadata is stored along with data in underlying storage devices, which may compromise reliability. Yi et al. (2014) propose an asymmetric file system where metadata is stored in a dedicated access domain, and clients are not allowed to directly access it. Their approach applies a centralized metadata server. To maintain scalability, they propose some techniques that include message stuffing, block reorganization in the disk (so metadata belonging to the same directory are close to each other), file layout prefetching, and speculative file allocation (to avoid data fragmentation). They also propose an algorithm to improve fault detection on the metadata server.

Situations where a large number of small files are accessed may present poor performance due to contention on the metadata servers. Moreover, when a large file is read or written, the initial cost of accessing its metadata is diluted by the larger read or write time. This does not happen when files are small. Hence metadata access optimizations are often motivated by this kind of workload.

Lu et al. (2012) work to improve performance in such situations. They tackle the ordered writes mechanism, which keeps consistency under distributed writes by ordering involved suboperations and ensuring data is written to storage devices before issuing metadata writes. This mechanism can degrade performance, especially for situations with a large number of small files. To improve performance, they propose a delayed commit protocol to transfer the order keeping to the file system so applications do not have to synchronously wait. They also employ a space delegation technique to cluster the space allocated to each client and increase the chance of merging I/O operations.

4.2 Requests Aggregation and Reordering

Access patterns with small requests, although common between scientific applications, usually achieve poor performance from the PFS. Therefore, many optimization techniques focus on the idea of *requests aggregation*, which means coalescing small accesses, uniting them into larger ones.

Islam et al. (2012) present a checkpoint-restart library that works to coalesce requests from the multiple processes to the PFS. They use information—variable name and type—to place similar data close together. They do so because compression schemes work better with similar data. Vishwanath et al. (2011) aggregate requests from the processing nodes to the I/O forwarding layer, promoting better usage of the target system—an IBM Blue Gene/P—interconnection.

A popular technique is the use of *collective operations*, whose idea consists in transforming multiple non-contiguous accesses from an application's processes into a single contiguous call. The classical collective write implementation is a two-phased strategy: first, all processes send their data to processes that were selected as "aggregators," and then aggregators perform the write operations. Collective read operations follow a similar approach. This is the strategy employed by ROMIO, a popular MPI-IO implementation (Thakur et al. 1999).

The typical two-phase I/O strategy works when the multiple processes access a single file. Kumar et al. (2011) implement two-phase I/O to multiple files in the context of the PIDX library. In a following work, Kumar et al. (2012) discuss a modification in their two-phase I/O strategy. Their new proposal is a three-phase approach with an additional phase at the beginning to restructure simulation data into large blocks to facilitate I/O aggregation. Finally, in a more recent work, Kumar et al. (2013) apply machine learning to automatically tune the library parameters.

Chen et al. (2010a) use data layout information to improve collective I/O performance. Their proposal consists of rearranging the partition of the file domain and of requests from aggregators, aiming at matching the physical layout on the servers. In a more recent publication by Chen et al. (2011), they use the physical data layout information in their collective I/O approach so each aggregator will access as few servers as possible. McLay et al. (2014) demonstrate that choosing the appropriate stripe size is critical to collective write performance. They propose some heuristics to facilitate this choice. Wang et al. (2014) also propose a new approach for collective I/O. They break collective I/O calls into multiple iterations to fit the buffer size. These partitions are optimized so each server is accessed by only one aggregator at each iteration.

The approach of decreasing the number of clients concurrently accessing each server, applied by some papers, improves performance for multiple reasons. First, concurrency on the servers is decreased and network contention avoided. Additionally, in some systems there is a cost associated with maintaining a connection between client and server, so these techniques save on this cost. Lastly, this approach potentially avoids contention caused by the server lock mechanism.

Liao (2011) proposes domain partitioning methods for collective I/O aiming at mitigating conflicts under stripe-based locking. The focus is to reduce lock contention on write operations to shared files, which typically causes serialization of concurrent I/O operations. Another technique to reduce lock contention is presented by Nisar et al. (2012). Their approach consists of statically mapping file regions based on the stripe size and count to delegate processes, reducing the concurrency on each server.

Despite being a popular technique, it is not always trivial for a developer to know if collective I/O is advantageous. Due to its added difficulty, many developers do not use this optimization when developing a scientific application and thus observe poor I/O performance. Natvig et al. (2010) propose a mechanism to monitor communication and I/O from applications and translate them into MPI-IO collective calls. Moreover, their approach works to aggregate writes and reads to eliminate overlaps and improve performance. With a similar motivation, Yu et al. (2013) present a user-level library to provide transparent collective I/O for applications through a POSIX-like interface. Their interface aims at being easier to use than MPI-IO. Zhang et al. (2012) present a data management middleware for parallel scripting. Their approach also works to automatically generate collective operations. Miller et al. (2011) present a collective output middleware for Charm++.

Performance improvements by collective operations come from both aggregation and reordering of small requests. If made independently by clients, these requests would hardly arrive to the servers in offset order. Avoiding random accesses, as previously discussed, can improve performance to access storage devices and promote better cache usage, also helping the efficacy of techniques like prefetching and read-ahead. Request reordering is the focus of some optimization techniques.

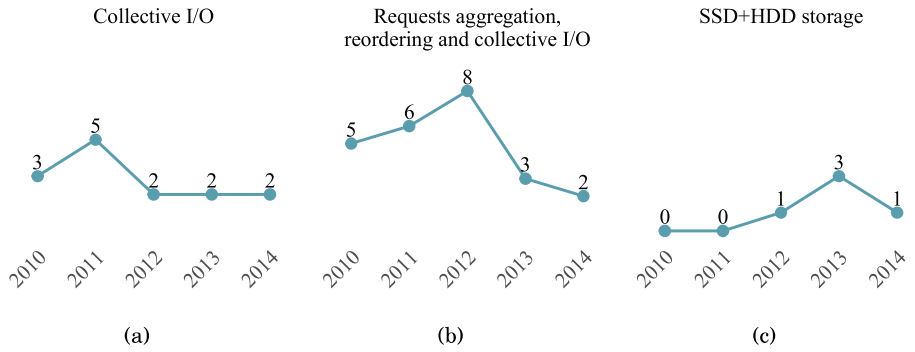

Fig. 7. Number of publications that present discussed optimization techniques, separated by year.

PLFS is a library that transparently maps application shared files into multiple actual files in the file system. This approach is adequate, for instance, for applications where each process accesses multiple sparse portions of a shared file (common when checkpointing). PLFS will then map each process request into an independent file, where they are contiguous. Manzanares et al. (2012) discuss some performance improvements to PLFS, focusing on concurrent reading (of a file previously written with PLFS) and metadata access. File creation is optimized by distributing the files managed by the library into multiple metadata servers. The latter optimization is conceived for file systems that do not distribute metadata storage. When these systems are deployed in a large-scale architecture, multiple file system trees are often kept at the same time to alleviate the bottleneck of a centralized server. Each of these concurrent metadata servers is responsible for a part of the directory tree. The optimization proposed by Manzanares et al. (2012) consists of distributing the library files across different points of this tree to avoid a situation where only one of the servers is keeping all of them.

Figure 7(a) presents the number of publications on collective I/O by year. These publications are more common in the beginning of the time window. A similar behavior can be observed in the graph from Figure 7(b), which shows publications classified as any of the three techniques: requests reordering, aggregation, or collective I/O.

4.3 Caching and Prefetching

A way of providing performance-transparent remote storage is placing caches at the different levels of the I/O stack to hide the latency of remote access. The success of caching can be improved by the prefetching technique, which, as previously stated, tries to fetch data from the next levels before it is actually requested by applications, so it is already present in the cache when needed.

Eshel et al. (2010) present a cache file system called "Panache," which uses pNFS to maintain a distributed cache for data stored in GPFS. The technique proposed by Frings et al. (2013) uses prefetching to increase the performance of loading parallel applications with dynamically linked libraries. Rajachandrasekar et al. (2013) propose a user-level file system to keep checkpointing requests in the main memory and transparently flush them to persistent storage. Their approach includes support to *remote direct memory access* (RDMA).

Another caching middleware is proposed by Zhao et al. (2014). They introduce a two-stage mechanism to decrease the amount of data to be transferred between processing and intermediate I/O nodes. Isaila et al. (2011) improve the IBM Blue Gene's I/O forwarding layer by proposing a two-level prefetching scheme (between clients and I/O nodes, and between I/O nodes and file

system servers). Prabhakar et al. (2010) model the optimal cache allocation on two-level cache systems through linear programming.

Kandemir et al. (2012) define the concept of requests' *urgency*, given by how long a request can be delayed without affecting the application performance. They improve a caching mechanism by prioritizing urgent requests. The approach by Seelam et al. (2010) applies a library that traces and detects the application access pattern. This information is used to guide prefetching to a local buffer. Similar approaches—using access pattern detection to guide prefetching—are proposed by Patrick et al. (2010), He et al. (2012), Lu et al. (2014), and Tang et al. (2014).

Suei et al. (2014) propose a cache design for storage clusters using an SSD as cache for an HDD. Their design focuses on wear-awareness, response time, and hit ratio. This idea—a fast SSD as a cache for an HDD—is also explored by Zhang et al. (2012).

The hybrid SSD+HDD approach is also used by Zhang et al. (2013). They apply SSDs to store "fragments," which are the initial and final portions of files that are not stripe-size aligned. Since the performance to obtain small portions is lower, the authors argue that the performance of accessing the whole file is limited by these fragments, so accelerating the access to them improves overall performance. Welch and Noer (2013) store small files in the SSD to optimize access to them, as they have observed that small files are the majority in PFSs. The approach presented by He et al. (2013) applies a cost model to make data placement decisions. They evaluate the access costs of different regions of a file and place high-cost regions in SSDs.

As discussed in Section 3, these hybrid storage solutions have been gaining popularity because simply replacing all hard disks by solid state drives would have a high cost. Therefore, HDDs are kept for storage capacity and SSDs for performance. Other NVRAM technologies are also being studied. New supercomputers are expected to include NVRAM devices in computing nodes. These devices can be used as "burst buffers," and work to hide the remote file system latency. A current research topic that has been receiving some attention seeks to determine how to use these burst buffers, where to place them, how to make them transparent, and so on (DOE 2014). Liu et al. (2012) evaluate the approach of having burst buffers in the intermediate I/O nodes. They use simulation for this analysis.

The graph from Figure 7(c) presents the number of publications studying hybrid storage solutions per year. We can see this is a rather recent trend, as we have not found surveyed papers on this subject before 2012.

4.4 I/O Scheduling

It is usual for large HPC architectures to dedicate a set of nodes for storage, with a PFS deployment. This file system will be concurrently accessed by all applications running in the machine. In this situation, an application's performance may be impaired in a phenomenon called "interference (Kuo et al. 2014)." I/O scheduling techniques are applied to alleviate interference effects by coordinating request processing. This coordination can work at different levels of the I/O stack.

Dai et al. (2014) propose a client-side I/O scheduling approach. They focus on avoiding *stragglers*—data servers that are slower than the others due to software bugs or interference effects. Write requests are redirected to other data servers to improve performance, and data can be later moved to the right server according to the striping schema in place. Zhang and Jiang (2010) identify portions of data that are causing interference during concurrent accesses. These portions are then replicated to other servers to decrease concurrency.

Dorier et al. (2014) propose a client-side cross-application coordination strategy. They use information about application access patterns to dynamically decide between three scheduling strategies, seeking to optimize a given metric. Lofstead et al. (2010) propose an adaptive I/O method,

implemented in the ADIOS middleware, which monitors the file system performance and balances the workload accordingly.

Liu et al. (2013) propose a server-side hierarchical scheduling algorithm for two-phase collective I/O. They focus on the "shuffle" phase, when data is moved between processing nodes. This phase is not synchronous, i.e., each aggregator passes data to other nodes as soon as it is available. They propose that servers prioritize aggregators with higher shuffle cost to provide better overall performance.

Zhang et al. (2010) propose an approach named IOrchestrator to the PVFS PFS. Their idea is to synchronize all data servers to serve only one application during a given period. This decision is made through a model considering the cost of this synchronization and the benefits of this dedicated service. The same authors adapt their approach to provide QoS support for end users (Zhang et al. 2011). Through a QoS performance interface, requirements can be defined in terms of execution time (deadline). The application access pattern is obtained from a profiling execution, and a machine learning technique is used to translate the provided deadline to requirements in bandwidth from the file system.

Song et al. (2011b) propose a server coordination scheme that also aims at serving one application at a time. They implemented a coordination strategy with a queue where applications are separated in time windows and ordered by application IDs. The time windows are used to avoid starvation.

Vishwanath et al. (2010) evaluate performance of the I/O forwarding layer of an IBM Blue Gene/P, and apply a simple *first come first served* scheduling algorithm. Ohta et al. (2010) take it further by including a handle-based round-robin scheduling algorithm. A data layout aware scheduler for the I/O forwarding layer is proposed by Xu et al. (2012) to provide proportional sharing between applications.

4.5 Other Techniques

When multiple processes generate requests to the remote file system from the same processing node, there may be contention in the access to memory and network resources. Dorier et al. (2012) propose an approach named *Damaris* that dedicates cores from SMP nodes for I/O. Processes assign their I/O operations to Damaris through a simple API, and data is kept in main memory until the I/O thread uses routines provided by the application itself to perform the operations to the PFS.

Dong et al. (2012) propose a load-balancing scheme for PFS data servers. Inadequate striping size (that does not reflect the application's characteristics), small files, and heterogeneous servers are the main causes for load imbalance at the servers, which may result in poor performance. They employ an agent on each server to monitor load and make decisions about data migrations. Another load-balancing approach is proposed by Ou et al. (2014). Their technique focuses on SSDs, taking their characteristics into consideration to perform wear leveling while avoiding write amplification.

Some applications and libraries perform data compression to decrease the amount of data to be sent to the remote PFS. Jenkins et al. (2012) propose an approach that adapts to different precision needs. A parallel data compression approach for I/O libraries is presented by Bicer et al. (2014). The approach presented by Filgueira et al. (2014) determines, through heuristics, when it is advantageous to use compression. It also allows for only parts of the file to be decompressed when necessary. Schendel et al. (2012) present a framework for overlapping I/O operations and data compression.

Active storage is a technique where servers are equipped to perform some simple operations over data they store. For instance, if clients are interested in reading an array from the file system

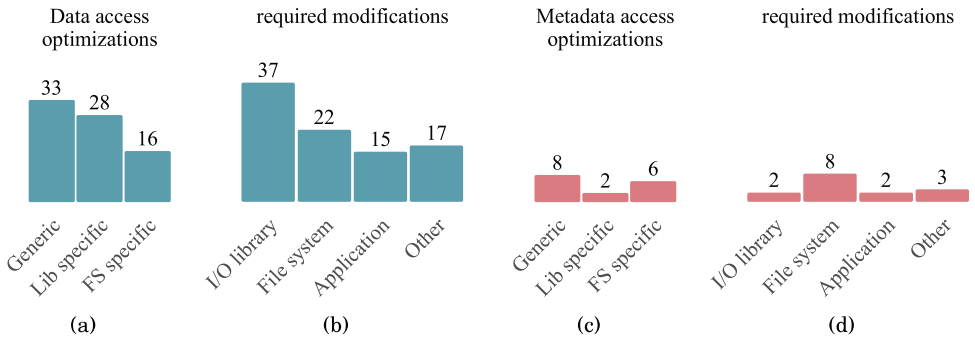

Fig. 8. Surveyed data on optimizations—required modifications and how generic proposed techniques are.

to calculate the average of its values, through active storage they could obtain the average directly from the system, decreasing the amount of transferred data and leveraging the servers' processing power. This concept is related to the near-data processing trend, which tries to avoid data movement impacts on performance, power efficiency, and reliability (Balasubramonian et al. 2014). The active storage approach is discussed by Piernas-Canovas and Nieplocha (2010). Son et al. (2010) discuss how active storage can be implemented in PFSs without information about data layout and striping.

A technique to leverage GPU processing power for the PFS client functions is presented by Al-Kiswany et al. (2013). They use the GPU to compute hash functions and detect block boundaries.

To guarantee consistency under concurrent access, MPI-IO implementations usually apply locking. This approach may present poor performance under a high level of concurrency. Tran et al. (2011) propose a versioning approach to provide atomicity with better performance.

I/O performance depends on the different I/O stack levels and their complicated interaction. For this reason, it is often difficult for application developers and users to achieve the best I/O performance. Behzad et al. (2013) present an auto-tuning system for HDF5 applications, which selects parameters and hints at runtime.

Jung et al. (2014) propose a flash array that is not based on SSDs. Their proposal achieves performance while not suffering from contention problems typical of SSD arrays. This is done through autonomic link and storage contention management, which includes changing the data layout to avoid contention on the flash modules.

An approach for automatic layout configuration is proposed by Song et al. (2011c). The approach consists of dividing a large file into multiple segments and adopting different layout configurations for each segment according to the observed access pattern. The same authors present a model to estimate the data access cost of different data layout policies (Song et al. 2011a). Another automatic data-placement approach is proposed by Yin et al. (2013), through a data-replication scheme that reorganizes data according to access patterns.

Meister et al. (2012) investigate the use of data deduplication to save storage capacity in HPC environments. In the work by Zou et al. (2012), task scheduling is enhanced with I/O information to group interrupts associated with the same I/O requests in the same core and to improve data locality.

4.6 Surveyed Data

We have presented techniques used to improve parallel I/O performance in the HPC context. As previously discussed, data access optimizations are typically done on the client side, while metadata access optimizations happen mostly on the server side. The graphs from Figure 8 further

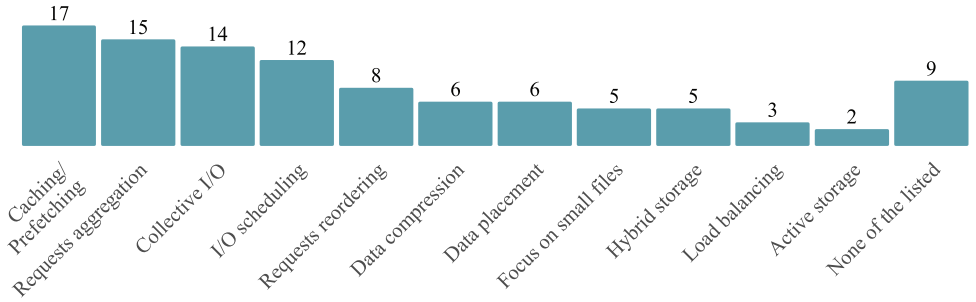

Fig. 9. Surveyed publications on data access optimization techniques.

characterize these optimizations. Figure 8(b) and (d) show the levels where proposed techniques (for data and metadata access optimization, respectively) require modifications. Similar to other presented graphs, they are not mutually exclusive, in the sense the same technique could require modifications to the I/O library and to the file system, for example. We can see metadata access optimizations are mostly done by modifying the file system. From the 73 publications on data access optimization, 37 require modifications to the I/O library, 22 to the file system, and 9 to both. The I/O library is the most popular place to implement data access optimizations. One reason for this is accessibility to researchers, as it is often easier to modify a user-space library than the deployed file system or I/O forwarder. Users of a supercomputer are not usually allowed to make such changes in the system for experimentation. Additionally, techniques including modifications to compilers are not common—only three were found, all for caching/prefetching data access optimizations (Kandemir et al. 2010; Patrick et al. 2010; Ding et al. 2012).

The graphs from Figure 8(a) and (c) show how generic the proposed optimization techniques are. In the context of this survey article, saying a solution is library or file-system specific means it only makes sense in the context of the library or file system where it was implemented. For instance, an improvement to how Lustre handles metadata operations is file-system specific, an improvement to ROMIO collective operations is library specific. It is important to notice a technique can be both library specific and file-system specific, but it can only be generic if not specific to any level of the I/O stack. We can see roughly half of the proposed techniques are generic. For specific solutions, data access optimizations are mostly specific to the I/O library, while metadata optimizations are more often specific to the file system. These results make sense if we remember where these optimizations usually take place. Moreover, one could argue being specific to a popular I/O library such as ROMIO is close to being generic.

Figure 9 classifies the articles on data access optimization according to the used technique. Some publications were classified as more than one technique. For instance, a paper could propose a data placement technique considering hybrid storage solutions, and thus it would count to both. Nonetheless, the last bar—"None of the listed"—gives only the number of publications where none of the listed techniques were used.

The most common technique among the surveyed papers is caching/prefetching. Figure 10(a), (b), and (c) present information on publications that discuss this technique. The first graph says if these techniques are generic or specific to a file system or I/O library, the second shows where they take place, and the last shows where these techniques require modifications. The "other" column in Figure 10(c) includes the three papers with modifications in the compiler. We can see caching/prefetching happens usually on the client side and most of these techniques are generic. The I/O library is the most usual place for this optimization, but other implementations are possible.

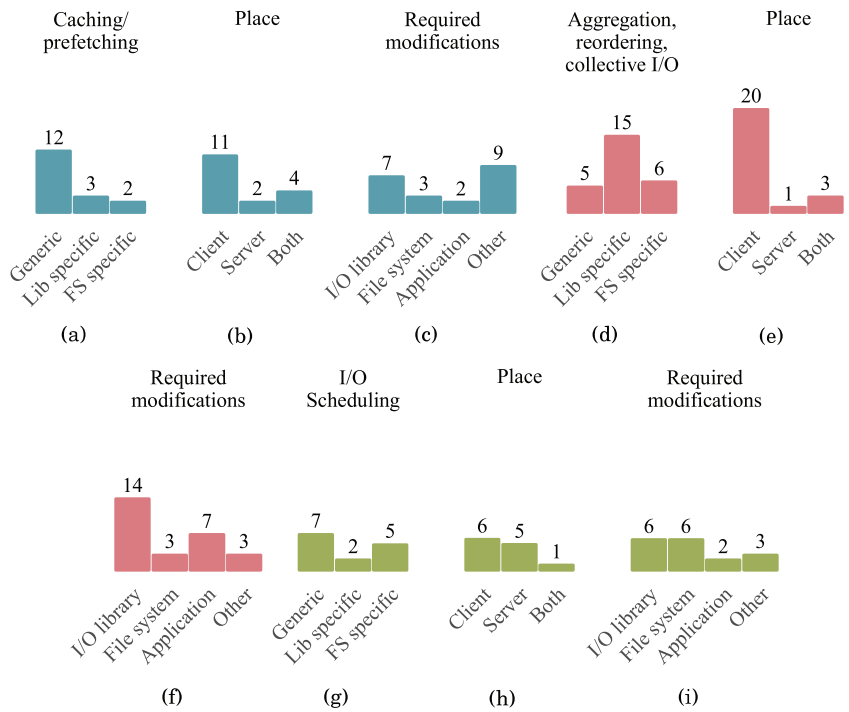

Fig. 10. Surveyed data on caching/prefetching, requests aggregation, reordering, collective I/O, and I/O scheduling—place, required modifications, and how generic proposed techniques are.

Similarly, Figure 10(d), (e), and (f) present information on publications using the request aggregation, reordering, and collective I/O techniques (24 papers fall in at least one of the 3 categories). These numbers demonstrate that request aggregation, reordering, and collective I/O are client-side techniques typically implemented in the I/O library. The fact that most of these research efforts are library specific is due to the technique implementation depending on data representation and on the way processes generate requests.

A different situation can be observed in the graphs from Figure 10(g), (h), and (i), which present information on I/O scheduling publications. I/O scheduling can take place on the client or server side, being implemented in the I/O library, in the file system, or even in the application.

4.7 Discussion

This section has discussed techniques to improve parallel I/O performance. Most of the surveyed publications focus on data access rather than metadata, and thus these data access optimization publications were classified according to the applied techniques.

Papers on metadata access optimization usually propose new distribution or consistency strategies. Reliability is also a concern, since losing metadata may incur in losing the associated data. Another motivation comes from situations where applications handle a large number of small files since the concurrency on metadata servers increases and thus the metadata operation cost becomes more important.

Many techniques work to aggregate and reorder requests, since access sequentiality can be important for performance. Moreover, reordering may be performed to avoid having a large number of clients accessing the same data servers concurrently or competing by the same stripes. The most usual optimization technique that performs aggregation and reordering is the use of collective

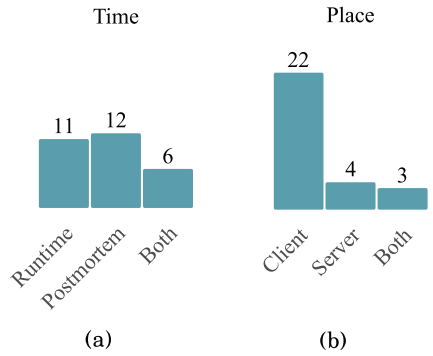

Fig. 11. Surveyed data on access pattern extraction—where and when these techniques take place.

operations. Despite being used for many years (two-phase I/O was introduced to ROMIO in the'90s), many papers propose new collective approaches or improvements to existing ones. Nonetheless, these publications are mostly from the beginning of our time window, suggesting this research topic is not as popular now as it used to be.

Caching is a typical solution to hide latency at all levels of the I/O stack. The prefetching technique usually requires an access pattern extraction strategy to get information from applications.

A recent trend is the use of non-volatile technologies together with hard disks, forming hybrid storage solutions. Researchers are working to integrate these new devices into the I/O stack, as they decrease the conceptual distance between memory and storage. SSDs may be used on the file system servers as a cache for the HDDs, or to store data that is most frequently used or most expensive to access. Moreover, burst buffers can be used on processing nodes or on intermediate I/O nodes to allow data movement pipelining. All surveyed publications on this subject are from 2012 or newer.

Other techniques to improve data access performance include I/O scheduling, dedicating cores from an SMP node to perform I/O on behalf of the others, load balancing and data placement guided by access patterns, data compression to decrease the amount of data being accessed, and leveraging processing power at the data servers to perform operations on data they store (active storage). Both surveyed publications on active storage are from 2010, suggesting this research subject is no longer as active as before. On the other hand, publications on data compression are mostly recent.

5 APPLICATIONS CHARACTERIZATION AND PERFORMANCE MODELING

Many techniques to improve parallel I/O performance need information about application access patterns. Prefetching techniques and cache substitution policies are common examples. Another source of information on optimization techniques is the use of models to represent and predict performance behavior. Models abstract systems, thus techniques can explore their parameter space to optimize given objectives—e.g., performance, resource utilization, load balancing, and so on.

Section 5.1 discusses techniques to obtain information about parallel applications. Section 5.2 presents publications on performance modeling. Both sections illustrate their discussions by presenting surveyed data. In Section 5.3, we summarize the discussion of this subject.

5.1 Access Pattern Extraction

From the surveyed articles, 29 (29%) discuss techniques to obtain information about application access patterns. The graph from Figure 11(a) shows when these techniques take place. The three bars are mutually exclusive, and the "both" option represents hybrid approaches where some

information is gathered after the application execution (postmortem), but a part of the detection is still done at runtime. Both postmortem and runtime approaches are popular. Nonetheless, hybrid-approach publications are less common.

At runtime, techniques can typically only use information from operations already performed by the application in the present execution. On the other hand, a more complete analysis could be conducted after the execution (postmortem). Postmortem techniques do not have the same time constraints runtime techniques have, since the latter are required to provide decisions on optimization techniques as fast as possible. The efficiency of runtime techniques is important to minimize the imposed overhead. Nonetheless, postmortem techniques are only adequate for workloads that will be observed multiple times, otherwise the obtained information will not be useful. If this information is used for a data access optimization technique, performance improvements will only be possible in future executions of the applications, as "profiling" executions will be required.

The graph from Figure 11(b) shows where the proposed techniques are applied: client side, server side, or with parts on both sides (these options are also mutually exclusive). Most of the access pattern extraction approaches work on the client side. The client side is where most information is available, since PFS servers are typically stateless and most high-level information is lost through the I/O stack. Server-side techniques are typically proposed to feed server-side optimizations.

Access pattern extraction techniques that work at runtime are discussed in Section 5.1.1, hybrid solutions in Section 5.1.2, and postmortem in Section 5.1.3.

5.1.1 Runtime Detection. Dorier et al. (2014) propose a grammar-based approach called Omnisc'IO. Their mechanism, integrated into POSIX and ROMIO to observe I/O calls, is adequate for applications that work on timesteps or perform regular checkpoints. In a few I/O phases, Omnisc'IO is able to build a grammar that predicts future accesses with good accuracy. It does so by tracking request size, offset, and inter-arrival time.

The approach proposed by Tang et al. (2014) periodically analyzes past accesses and applies a rules library to predict future accesses (for prefetching). They collect spatiality of read requests from the MPI-IO library.

It is usual for these techniques to benefit from information available in I/O libraries. Ge et al. (2012) collect information from the MPI-IO library: operation (write, read, seek, open, or close), data size, spatiality (contiguous or strided), if operations are collective, and if operations are synchronous. Liu et al. (2013) collect from MPI-IO the number of processes, the number of aggregators (of collective operations), and binding between nodes and processes. Lu et al. (2014) use the offsets accessed by each process during collective operations. The processes' access spatiality is also obtained from MPI-IO in the approach proposed by Song et al. (2011a). Similarly, He et al. (2012) present an approach to collect I/O information from the PnetCDF library. Semantic data access information is obtained by collecting high-level variables.

All the discussed techniques so far work on the client side. As previously discussed, that is where information is more easily obtained from I/O libraries, applications, and so forth.

Dong et al. (2012) use a time series model to estimate file system server load. The approach by Zhang et al. (2010) applies a "reuse distance", defined as the time difference between consecutive requests from the same application to the same server.

The graphs from Figure 12(a) and (b) present information on access pattern extraction at runtime. The first graph shows where these techniques take place and the second shows to what levels modifications are required. In the latter, options are not mutually exclusive, as a technique may require modifications at multiple levels. Most runtime techniques work on the client side and require modifications to the I/O library.

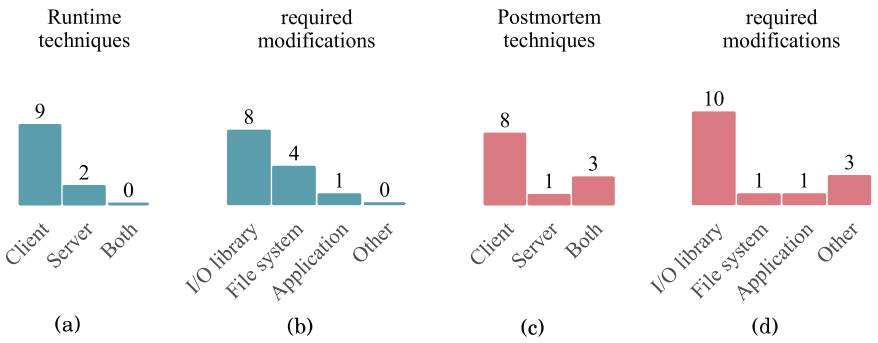

Fig. 12. Surveyed data on access pattern extraction techniques that work at runtime or postmortem—where they take place and to which levels they require modifications.

5.1.2 Hybrid Runtime + Postmortem Approaches. Yin et al. (2012) propose IOSIG, a tool that generates application traces and performs postmortem analysis to describe access patterns according to spatiality, request size, time between requests, and operation (read or write). An application is described by a sequence of different access patterns, as its behavior may change during its execution. They propose prefetching and data layout optimizations guided by access patterns, which are identified at runtime using the profiles obtained from the postmortem analysis. A similar technique is used by Zhang et al. (2011), where a profiling execution is required to detect application access patterns regarding the portion of time used for I/O, average request size, and average offset difference between requests. At runtime, a machine learning technique is used to translate a given deadline requirement in bandwidth, according to the profiled access pattern and information from the file system.

The approach proposed by Patrick et al. (2010) relies on hints placed in the source code by the application developer and captured by the compiler. He et al. (2013) propose an approach to improve read operations with the PLFS library. Information from trace files—generated while writing data—is used to reconstruct the high-level data structures used by the application (through MPI-IO or HDF5). The knowledge of the data structures allows for better metadata representation and data prefetching.

5.1.3 Postmortem Analysis. Liu et al. (2014) gather information from server-side traces generated by their target architecture. The traces, which contain the system throughput measured every 2 seconds, are noisy from interference caused by concurrent applications. By gathering multiple traces from different executions of the same application, they are able to filter the interference and determine the I/O requirements of the application during its execution.

In the approach proposed by Yin et al. (2013), the MPI-IO library was modified to generate traces detailing, to each file operation, MPI rank, process identifier, file identifier, offset inside the file, request size, operation (read or write), starting time, and end time. This information is later used to guide data replication. He et al. (2013) use the IOSIG tool to capture a trace from an application. Data access costs are then calculated for each file region and results are stored in a region table, which is used to optimize future executions of the application. Similarly, Song et al. (2011c) collect traces to identify the number of requests and their size to each file region. This information is later used to compute an optimal stripe size for each part of the file. Zhang and Jiang (2010) use client-side traces and a simulator to detect file portions which are related to interference, and then replicate these portions to avoid it.

Kandemir et al. (2012) automatically instrument applications to delay I/O operations to measure the effect of this delay in final performance. Obtained information is then used to prioritize more "urgent" operations.

The technique presented by Logan et al. (2012) differs from the previously discussed publications because it does not aim at providing information to guide optimizations. They obtain information from the XML file provided by applications to the ADIOS library and use it to build "I/O skeletal applications," which mimic the original application's I/O behavior. Therefore, their tool automatically creates I/O kernels from applications. Moreover, they also provide tools for I/O performance evaluation.

Similarly, Sigovan et al. (2013) present a method to extract and visualize network performance metrics from I/O trace data collected on HPC platforms. Their visualization approach consists of representing each layer as a concentric ring, and communications between layers as connections between the rings. In the work by Uselton et al. (2010), statistical analysis of request processing time, obtained from traces, is used to identify modes and moments of the processing time distribution, revealing I/O behavior of applications and potential bottlenecks. Ilsche et al. (2012) aggregate trace data from applications before sending it to the parallel file system, aiming at decreasing the perturbation caused by this type of workload.

Carns et al. (2011a) showcase the Darshan profiling tool. They evaluate the I/O performance of a large architecture during two months. Darshan instruments POSIX, MPI-IO, HDF5, and netCDF libraries, providing data for a wide range of applications Carns et al. (2011b). Liu et al. (2012) use Darshan to characterize target applications' I/O behaviors.

The graphs from Figure 12(c) and (d) show information on postmortem access pattern characterization. Among them, six proposals aim at providing information to optimization techniques, while the others provide methods for application evaluation. The first graph shows where these techniques take place and the second graph shows what modifications they require. Most techniques work on the client side. Almost all of them require modifications to the I/O library (ten out of twelve).

5.2 Performance Modeling

Among the surveyed papers, 15 (15%) use performance models. Most of them (13) use a model of the system to drive optimization decisions. The only exceptions are the papers by Xie et al. (2012) and Zhang et al. (2013). Xie et al. (2012) focus on characterizing the storage performance on a large-scale machine. A model is used in their work to estimate the capability of a storage system to absorb the output of multiple parallel processes. The work by Zhang et al. (2013) evaluates the capability of large-scale computers in the context of parallel scripting applications. They present a model to estimate read and write times for data and metadata-bound workloads.

Natvig et al. (2010) use analytical models to estimate I/O performance and to evaluate the benefits of collective operations. Their models account for both network and file system costs, considering also the problem size and the number of nodes in the system. A similar approach is adopted by Piernas-Canovas and Nieplocha (2010). They use an analytical model to identify when applications can take advantage of active storage. Parameters considered in their model include processing speed on compute and storage nodes, maximum network and single-link bandwidth, and disk read/write rate. Schendel et al. (2012) propose a theoretical performance model to be used by the ISOBAR framework to make decisions about data compression.

A model-driven data layout scheme is proposed by Song et al. (2011c). They propose an analytical model to estimate the data access cost to each segment of the file and tune striping. I/O time is computed considering the server's startup time (i.e., disk seek and software overhead), stripe size, request size, the number of storage nodes, and storage device transfer time. This model is

extended by He et al. (2013) to evaluate I/O performance improvements in hybrid environments through the placement of specific file segments on faster servers.

Data placement decisions are also guided by analytical models in the work by Yin et al. (2013). They propose cost models for three data layout policies, considering the number of processing and storage nodes, request size, network connection time, network bandwidth, the startup time of one disk I/O operation, time to read/write one unit of data, and the number of storage groups in a 2D layout.

In the work by Dong et al. (2012), an autoregressive time series model is used to predict the load of storage nodes. The prediction model is built online to each storage node with load information from local and remote nodes. Model estimates are used to guide the data migration process. Prabhakar et al. (2010) use a regression model to estimate per application I/O latency.

Liu et al. (2012) integrate an analytical model of burst buffers in the CODES storage system simulator to investigate the performance impact of adopting them in the storage infrastructure. Burst buffer access cost is modeled using device throughput and latency. Rajachandrasekar et al. (2013) use a model to estimate the throughput of their user-space file system CRUISE, which stores data in main memory and transparently flushes to persistent storage. Their model considers spill-out to SSDs and includes parameters such as the amount of data, and main memory and SSD throughput.

Machine learning techniques are used by Kumar et al. (2013) to tune parameters of the PIDX I/O library. A tree-based modeling approach was chosen because it presented better accuracy results, and also because such models provide understandable information about the prediction process. Among the evaluated tunable parameters are the number of aggregators, the aggregation factor, the number of files, and whether or not restructuring is used. Lakshminarasimhan et al. (2013) apply a network-based performance model to estimate indexing, aggregation, and I/O times for DIRAQ, a parallel in situ output indexing and compression technique.

5.3 Discussion

This section has discussed two common classes of techniques found among parallel I/O publications: access pattern extraction and performance modeling, both typically used to guide techniques to improve parallel I/O performance. Additionally, 9 (out of 29) surveyed publications on access pattern extraction did not use information to guide optimizations but focused on methodologies to evaluate and profile application I/O performance.

Access pattern extraction techniques can work at runtime, after the application execution (postmortem), or through a hybrid solution that is a combination of the two. Surveyed publications are divided evenly among runtime and postmortem, and hybrid solutions are less common. Moreover, most proposed techniques work on the client side by modifying the I/O library. Client-side runtime techniques typically obtain information from the I/O library, where it is possible to gather request details such as operation type, size, spatiality, and sometimes information like the number of processes, the number of aggregators, and binding between processes and machines. Server-side runtime techniques, on the other hand, have little information and are able to gather, for instance, request arrival rate and current file system load. Most postmortem techniques work with traces: 14 publications use them, 9 for postmortem analysis, and 5 for hybrid solutions. In addition to other information available from I/O libraries, traces provide file system access time, which allow for deriving elaborate information like file region access cost.

Thirteen of the surveyed papers (13%) propose optimizations based on prediction models. It is worth noticing the trade-off between model accuracy and usefulness, as although many relevant factors could be considered, a wide range of factors may turn models unfeasible for online techniques. The next section is the last part of this survey article and presents practical aspects of parallel I/O research for HPC, identifying general characteristics of researchers and publications.

6 PRACTICAL ASPECTS OF PARALLEL I/O RESEARCH

Most published contributions to the parallel I/O field come from universities or research institutions, as from the surveyed publications only 12 (12%) of them had authors from a company. Among these, nine were collaborations between a company and a university or laboratory. The companies with more publications are: IBM, with three papers, and Intel, with two.

Table 2 shows the number of publications per country. The numbers do not add up to 99 because a paper can be authored by institutions from multiple countries. This is the case in 21 (21%) of them. From these, most of them (17) are co-authored by an institution from the United States of America. Additionally, no surveyed publication was the subject of a collaboration between more than two countries.

It is clear the USA is involved in most of the surveyed publications (87, 88%). The most active USA state on parallel I/O research during the considered time window was Illinois, with 47 papers, followed by California, with 17. One of the main reasons for Illinois' prominent position is the Argonne National Laboratory,[7] which was involved in 35 publications, being the institution with the largest number of papers. It is important to notice, however, that this result could, to some extent, be biased by the fact some of the considered conferences—like SC and FAST—are from the USA.

Another thing to be noticed from the presented numbers is the fact that most surveyed research comes from developed countries that make large investments on research. One reason for this is that parallel I/O in the HPC context makes more sense for countries with large scientific research centers. Moreover, as researchers argue about techniques at exascale, it is important they have access to large-scale architectures to perform experiments and collect data.

The graphs from Figure 13 present information on the scale of surveyed experiments. Their x axes represent the number of *machines involved in the experiments*[8]: tens, hundreds, thousands, tens of thousands, or hundreds of thousands of nodes. The y axes give the number of articles considered in this article. Since only the size of the largest experiment from each publication is taken, there is no overlap between the different bars from each graph.

The first graph [Figure 13(a)] presents the general distribution of experiment scale. We can see most publications included experiments conducted on hundreds of nodes or more. A similar behavior is observed in the second graph [Figure 13(b)], which shows the scale of experiments among papers proposing data or metadata access optimization techniques. Nonetheless, most experiments for access pattern extraction techniques—presented in Figure 13(c)—were conducted in small scale (tens of nodes).

In parallel I/O publications, large-scale machines are not only useful to conduct experiments to validate new ideas, but evaluations of them are also sometimes contributions to the field. This happens because there is interest and curiosity on how things work in extreme scale, and not every researcher has access to supercomputers. Among the surveyed publications, over a third mentioned the use of famous Top500 machines. Three of them focused on the machine evaluation and lessons learned with it, like the work by Lofstead and Ross (2013). Others focused on large-scale demonstrations of scientific applications (Byna et al. 2012; Oldfield et al. 2014; Yu et al. 2012) and file systems (Henschel et al. 2012). Lofstead et al. (2011) classifies and evaluates read access patterns that are common across a range of scientific applications. These papers provide insights on how "real life" parallel I/O looks like.

Simulation is an alternative for when researchers do not have access to a large-scale machine. Even when access is possible, they are often not allowed to make deep modifications to the system, or the modifications would be too time consuming and thus it is important to be sure they

[7] http://www.anl.gov.

[8] We took the number of machines used for the experiment, not the size of the cluster.

Table 2. Number of Surveyed Publications per Country

	USA	China	France	Germany	Japan	Spain	Canada	Taiwan	UK	Norway	Qatar	Singapore	South Korea
Publications	87	8	4	5	3	3	2	2	2	1	1	1	1

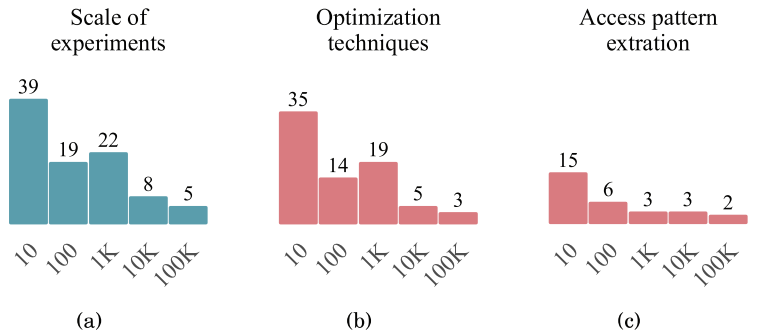

Fig. 13. Scale of the surveyed experiments.

are advantageous before proceeding with them. Despite being widely used in fields such as microprocessor design and memory hierarchy, simulators are not usual among parallel I/O publications. From the surveyed papers, only five of them use simulation: two for validating data access optimization techniques, two for validating metadata access optimizations, and one proposes a new simulator. Costa et al. (2014) propose a queue-based simulation model composed of three elements: a centralized metadata server, multiple data servers, and clients. Their simulator parameters are collected through tracing and monitoring of real workloads and environments.

In addition to the machine size, another important concern when designing experiments is making sure results are statistically sound. Noise can happen at the many different levels of the I/O stack, and cause parallel I/O experiments to present a high variability. Therefore, experiments are usually repeated multiple times to account for this noise. Among the 93 articles presenting experimental results, only 30 (32%) of them[9] presented some statistics-related methodology information such as the number of repetitions, standard deviation, confidence interval, or variability. Many research efforts do not appear to handle experimental variability, and this is an issue because it could compromise how valid their results are and their reproducibility.

A big reason for this somewhat common practice may be related to the cost of the experiments. Replicating every evaluated scenario multiple times has a cost in time that can be prohibitive in some situations. The problem seems to be, in fact, inherent to the I/O field, as data access tends to be time-consuming. We believe adequate experimental designs, considering constraints inherent to experiments in computing environments, should be researched.

7 CONCLUSION

Parallel I/O has been an important topic in the HPC community for decades, motivated by the everlasting gap between processing and data access speeds and by increases in HPC architecture scale and thus in application I/O requirements. As the HPC community moves toward the exascale

[9]For another four papers this information was not necessary, due to their experiments' nature.

era, I/O remains a central issue. This article has presented a comprehensive survey on parallel I/O in the HPC context, aiming at both providing basic concepts and identifying the field's main current and future research topics. To do that, we have discussed 5 years of papers from some of the most important conferences and journals.

One of the first questions that arise is, "Has the amount of research effort put in the field grown in the past few years?" We *have not* observed a consistent increase in the volume of publications over the considered 5 years. This is the case because parallel I/O has been an issue for many years, as HPC advanced towards petascale.

As a mature research field, the parallel I/O community has well-known and established tools regarding experimental and production environments. We have shown Lustre and PVFS are the most used PFSs (PVFS mostly for research), and MPI-IO is the most used I/O library. Nonetheless, the field still lacks standardization when it comes to benchmarking. A standard set of benchmarks and applications—as the role played by the NAS parallel benchmarks in other fields of HPC—would facilitate the comparison between techniques and strengthen the presented validations. Although an effort toward standardization was done approximately 10 years ago, our surveyed data makes it clear that this effort is still necessary, as customized benchmarks are the rule instead of the exception.

Another research aspect that requires work is parallel I/O simulation. From 93 papers presenting experiments, only 5 of them use some kind of simulation, and one of those propose a new simulator. Although we can find a few simulators in the literature, it is unusual to find papers that use them. One possible reason for this is these simulators are too specific to a given system architecture or software, limiting its usage by other researchers. As our results have pointed out the importance of large-scale experiments, we believe some research effort should be employed to reach high-quality parallel I/O simulation to facilitate the development and validation of new techniques.

An advantage of accurate and widely available simulators would be to allow I/O experiments that are less subject to variability. Although experiments suffer from noise at the multiple levels of the I/O stack and hence tend to present high variability, most of the surveyed articles did not seem to account for this variability. A statistically sound methodology is important so findings can be reproducible and trusted.

Most of the surveyed publications proposed techniques to improve data access performance. We have classified these techniques according to the main strategy applied by them, and shown their representation among the whole set of publications, as well as trends through the years. Among the most usual strategies for optimization are collective I/O, requests aggregation and reordering, I/O scheduling, caching and prefetching, I/O forwarding, data compression, load balancing, active storage, and data placement.

Some of these techniques were mostly discussed in publications from the first half of the considered time window, which suggests there is now less research activity on these topics than before. That is the case of request aggregation, reordering, collective I/O, and active storage. It is possible most opportunities with these techniques for the current technologies and systems were already explored.

We have discussed publications on hybrid storage solutions, where hard disks are still used for permanent storage, but faster devices with lower capacity (such as SSDs) work as caches or are used to store performance-critical data. When placed at processing nodes or intermediate I/O nodes, these faster storage devices can work as burst buffers. All publications on this subject are from the second half of the considered time window. As new non-volatile storage technologies become more popular, future large-scale architectures are expected to include them. In the near future, intense research activity is expected to focus on the integration of these devices in the I/O stack.

The HPC community has been putting some effort into decreasing systems' energy consumption. This is a central issue because, through current technologies, an exascale machine would consume more power than what is reasonable (Kogge et al. 2008; Mair et al. 2015). It should also be a concern for parallel I/O researchers as, although the processing units are responsible for most of the power demand in typical computational systems, many applications spend most of their time performing I/O operations (Orgerie et al. 2013; Chandrasekar et al. 2015). For this reason, increasing the energy efficiency of the I/O subsystem is also an important step to tackle the energy and power challenge. In the next few years, some research activity is expected to focus on this issue.

New interconnection technologies for HPC allow for RDMA, which is a technique where one process can directly access other processes' memory without involvement of their operating systems (Gerstenberger et al. 2013). The use of RDMA has the potential to decrease the negative impact of communication on storage performance and is a likely topic for future research.

We have also discussed access pattern extraction techniques, most of them focused on providing information to guide data access optimization techniques. It is clear integrating access pattern information into optimization techniques allows for more intelligent solutions that achieve better performance. We have shown most of the proposed techniques work on the client side by interacting with the I/O library. This is a popular approach for optimizations because on the client side it is possible to obtain more information from applications, as this information is typically lost through the I/O stack. The studied server-side techniques are able to gather only very simple information, such as request arrival rate. Better server-side optimizations could be achieved by having access to more information on what happens on the application side. We believe future parallel I/O research should focus on a better integration across the I/O stack, so all levels have access to information which can be used to customize optimizations and achieve the best performance.

Reformulations of the I/O stack must also keep the application developers in mind. Working with as little input from the user as possible and making I/O libraries easier to use is an important goal so parallel I/O researchers' findings can improve the quality perceived by all HPC users. We have discussed some techniques that work to automatically tune and adjust parameters and calls to improve performance, and this topic is expected to continue to be of great importance for the field.

REFERENCES

Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. 2013. GPUs as storage system accelerators. *IEEE Trans. Parallel Distrib. Syst.* 24, 8 (2013), 1556–1566. DOI: http://dx.doi.org/10.1109/TPDS.2012.239arXiv:1202.3669

Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H. Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. 2014. Near-data processing: Insights from a MICRO-46 workshop. *IEEE Micro* 34, 4 (2014), 36–42. DOI: http://dx.doi.org/10.1109/MM.2014.55

Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Prabhat, Ruth Aydt, Quincey Koziol, and Marc Snir. 2013. Taming parallel I/O complexity with auto-tuning. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (*SC'13*). ACM Press, 1–12. DOI: http://dx.doi.org/10.1145/2503210.2503278

Tekin Bicer, Jian Yin, and Gagan Agrawal. 2014. Improving I/O throughput of scientific applications using transparent parallel compression. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'14)*. IEEE, 1–10. DOI: http://dx.doi.org/10.1109/CCGrid.2014.112

Surendra Byna, Jerry Chou, Oliver Rübel, Prabhat, Homa Karimabadi, William S. Daughter, Vadim Roytershteyn, E. Wes Bethel, Mark Howison, Ke Jou Hsu, Kuan Wu Lin, Arie Shoshani, Andrew Uselton, and Kesheng Wu. 2012. Parallel I/O, analysis, and visualization of a trillion particle simulation. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–12. DOI: http://dx.doi.org/10.1109/SC.2012.92

Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011a. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. Storage* 7, 3 (Oct. 2011), 1–26. DOI: http://dx.doi.org/10.1145/2027066.2027068

Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011b. Understanding and improving computational science storage access through continuous characterization. In *Proc. 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*. IEEE, 1–14. DOI: http://dx.doi.org/10.1109/MSST.2011.5937212

CFS. 2002. *Lustre: A Scalable, High-Performance File System*. White paper. Cluster File Systems, Inc.

Raghunath Raja Chandrasekar, Akshay Venkatesh, Khaled Hamidouche, and Dhabaleswar K. Panda. 2015. Power-check: An energy-efficient checkpointing framework for HPC clusters. In *Proceedings of the IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing (CCGRID'15)*. IEEE, 261–270. DOI: http://dx.doi.org/10.1109/CCGrid.2015.169

Yong Chen, Xian He Sun, Rajeev Thakur, Philip C. Roth, and William D. Gropp. 2011. LACIO: A new collective I/O strategy for parallel I/O systems. In *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS'11)*. IEEE, 794–804. DOI: http://dx.doi.org/10.1109/IPDPS.2011.79

Yong Chen, Xian-He Sun, Rajeev Thakur, Huaiming Song, and Hui Jin. 2010a. Improving parallel I/O performance with data layout awareness. In *CLUSTER'10 Proceedings of the 2010 IEEE International Conference on Cluster Computing*. IEEE, 302–311. DOI: http://dx.doi.org/10.1109/CLUSTER.2010.35

Zhuan Chen, Jin Xiong, and Dan Meng. 2010b. Replication-based highly available metadata management for cluster file systems. In *Proceedings of the 2010 IEEE International Conference on Cluster Computing (CLUSTER'10)*. IEEE, 292–301. DOI: http://dx.doi.org/10.1109/CLUSTER.2010.34

Peter Corbett, Dror Feitelson, Sam Fineberg, Yarsun Hsu, Bill Nitzberg, Jean-Pierre Prost, Marc Snirt, Bernard Traversat, and Parkson Wong. 1996. *Overview of the MPI-IO Parallel I/O Interface*. Springer, 127–146. DOI: http://dx.doi.org/10.1007/978-1-4613-1401-1_5

Lauro Beltrão Costa, Samer Al-Kiswany, Hao Yang, and Matei Ripeanu. 2014. Supporting storage configuration for I/O intensive workflows. In *Proceedings of the 28th ACM international conference on Supercomputing (ICS'14)*. ACM Press, 191–200. DOI: http://dx.doi.org/10.1145/2597652.2597679

Dong Dai, Yong Chen, Dries Kimpe, and Robert Ross. 2014. Two-choice randomized dynamic I/O scheduler for object storage systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC'14*). IEEE, 635–646. DOI: http://dx.doi.org/10.1109/SC.2014.57

Wei Ding, Yuanrui Zhang, Mahmut Kandemir, and Seung Woo Son. 2012. Compiler-directed file layout optimization for hierarchical storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.35

DOE. 2013. *Data Crosscutting Requirements Review*. Technical Report. U.S. Department of Energy.

DOE. 2014. *Storage Systems and Input/Output to Support Extreme Scale Science: Report of the DOE Workshops on Storage Systems and Input/Output*. Technical Report. U.S. Department of Energy and National Nuclear Security Administration.

DOE/NNSA. 2014. *Preliminary Conceptual Design for an Exascale Computing Initiative*. Technical Report. U.S. Department of Energy and National Nuclear Security Administration.

Bin Dong, Xiuqiao Li, Qimeng Wu, Limin Xiao, and Li Ruan. 2012. A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers. *J. Parallel Distrib. Comput.* 72, 10 (2012), 1254–1268. DOI: http://dx.doi.org/10.1016/j.jpdc.2012.05.006

Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, and Leigh Orf. 2012. Damaris: How to efficiently leverage multicore parallelism to achieve scalable, jitter-free I/O. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER'12)*. IEEE, 155–163. DOI: http://dx.doi.org/10.1109/CLUSTER.2012.26

Matthieu Dorier, Gabriel Antoniu, and Robert Ross. 2014. CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*. IEEE, 155–164. DOI: http://dx.doi.org/10.1109/IPDPS.2014.27

Matthieu Dorier, Shadi Ibrahim, Gabriel Antoniu, and Robert B. Ross. 2014. Omnisc'IO: A grammar-based approach to spatial and temporal I/O patterns prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. IEEE, 623–634. DOI: http://dx.doi.org/10.1109/SC.2014.56

Marc Eshel, Roger Haskin, Dean Hildebrand, Manoj Naik, Frank Schmuck, and Renu Tewari. 2010. Panache: A parallel file system cache for global file access. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*. USENIX Association, 1–14.

Rosa Filgueira, Malcolm Atkinson, Yusuke Tanimura, and Isao Kojima. 2014. Applying selectively parallel I/O compression to parallel storage systems. In *Euro-Par 2014—Parallel Processing*, Fernando Silva, Inês Dutra, and Vitor Santos Costa (Eds.). Lecture Notes in Computer Science, Vol. 8632. Springer International Publishing, 282–293.

Wolfgang Frings, Dong H. Ahn, Matthew LeGendre, Todd Gamblin, Bronis R. de Supinski, and Felix Wolf. 2013. Massively parallel loading. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing (ICS'13)*. ACM Press, 389–398. DOI: http://dx.doi.org/10.1145/2464996.2465020

Rong Ge, Xizhou Feng, and Xian He Sun. 2012. SERA-IO: Integrating energy consciousness into parallel I/O middleware. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'12)*. IEEE, 204–211. DOI: http://dx.doi.org/10.1109/CCGrid.2012.39

Robert Gerstenberger, Maciej Besta, and Torsten Hoefler. 2013. Enabling highly-scalable remote memory access programming with MPI-3 one sided. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing*. ACM Press, 1–12. DOI: http://dx.doi.org/10.3233/SPR-140383

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. *ACM SIGOPS Op. Syst. Rev.* 37, 5 (2003), 43. DOI: http://dx.doi.org/10.1145/945445.945450

HDF Group, The. 1997-2016. HDF5—Hierarchical Data Format, Version 5.

Jun He, John Bent, Aaron Torres, Gary Grider, Garth Gibson, Carlos Maltzahn, Xian-He Sun, D Operating Systems, and Storage Management. 2013. I/O acceleration with pattern detection. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC'13)*. ACM Press, 25–36. DOI: http://dx.doi.org/10.1145/2462902.2462909

Jun He, Xian-He Sun, and Rajeev Thakur. 2012. KNOWAC: I/O prefetch via accumulated knowledge. In *2012 Proc. IEEE International Conference on Cluster Computing*. IEEE, 429–437. DOI: http://dx.doi.org/10.1109/CLUSTER.2012.83

Shuibing He, Xian-He Sun, Bo Feng, Xin Huang, and Kun Feng. 2013. A cost-aware region-level data placement scheme for hybrid parallel I/O systems. In *Proceedings of the 2013 IEEE International Conference on Cluster Computing (CLUSTER'13)*. IEEE, 1–8. DOI: http://dx.doi.org/10.1109/CLUSTER.2013.6702615

Robert Henschel, Stephen Simms, David Hancock, Scott Michael, Tom Johnson, Nathan Heald, Thomas William, Donald Berry, Matt Allen, Richard Knepper, Matthew Davy, Matthew Link, and Craig A. Stewart. 2012. Demonstrating lustre over a 100Gbps wide area network of 3,500km. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–8. DOI: http://dx.doi.org/10.1109/SC.2012.43

Yu Hua, Yifeng Zhu, and Hong Jiang. 2011. Supporting scalable and adaptive metadata management in ultralarge-scale file systems. *IEEE Trans. Parallel Distrib. Syst.* 22, 4 (2011), 580–593. DOI: http://dx.doi.org/10.1109/TPDS.2010.116

Thomas Ilsche, Joseph Schuchart, Jason Cope, Dries Kimpe, Terry Jones, Andreas Knüpfer, Kamil Iskra, Robert Ross, Wolfgang E. Nagel, and Stephen Poole. 2012. Enabling event tracing at leadership-class scale through I/O forwarding middleware. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*. ACM, 49–60. DOI: http://dx.doi.org/10.1145/2287076.2287085

Florin Isaila, Javier Garcia Blas, Jesus Carretero, Robert Latham, and Robert Ross. 2011. Design and evaluation of multiple-level data staging for blue gene systems. *IEEE Trans. Parallel Distrib. Syst.* 22, 6 (2011), 946–959. DOI: http://dx.doi.org/10.1109/TPDS.2010.127

Tanzima Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R. De Supinski, and Rudolf Eigenmann. 2012. McrEngine: A scalable checkpointing system using data-aware aggregation and compression. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.3233/SPR-130371

John Jenkins, Eric R. Schendel, Sriram Lakshminarasimhan, David A. Boyuka, Terry Rogers, Stephane Ethier, Robert Ross, Scott Klasky, and Nagiza F. Samatova. 2012. Byte-precision level of detail processing for variable precision analytics. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.26

Myoungsoo Jung, Wonil Choi, John Shalf, and Mahmut Kandemir. 2014. Triple-A: A non-SSD based autonomic all-flash array for high performance storage systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. ACM Press, 441–454. DOI: http://dx.doi.org/10.1145/2541940.2541953

Mahmut Kandemir, Sai Prashanth Muralidhara, Mustafa Karakoy, and Seung Woo Son. 2010. Computation mapping for multi-level storage cache hierarchies. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. ACM Press, 179–190. DOI: http://dx.doi.org/10.1145/1851476.1851498

Mahmut Kandemir, Taylan Yemliha, Ramya Prabhakar, and Myoungsoo Jung. 2012. On urgency of I/O operations. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'12)*. IEEE, 188–195. DOI: http://dx.doi.org/10.1109/CCGrid.2012.40

Michael P. Kasick, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. 2010. Black-box problem diagnosis in parallel file systems. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*. USENIX Association, 1–14. http://dl.acm.org/citation.cfm?id=1855511.1855515

Jaehong Kim, Sangwon Seo, Dawoon Jung, Jin-Soo Kim, and Jaehyuk Huh. 2012. Parameter-aware i/o management for solid state disks (SSDs). *IEEE Trans. Comput.* 61, 5 (2012), 636–649. DOI: http://dx.doi.org/10.1109/TC.2011.76

Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, et al. 2008. *Exascale Computing Study: Technology Challenges in Achieving Exascale Systems*. Technical report. Defense Advanced Research Projects Agency (DARPA IPTO).

Sidharth Kumar, Robert Ross, Michael E. Papkafa, Jacqueline Chen, Valerio Pascucci, Avishek Saha, Venkatram Vishwanath, Philip Carns, John A. Schmidt, Giorgio Scorzelli, Hemanth Kolla, Ray Grout, and Robert Latham. 2013. Characterization and modeling of PIDX parallel I/O for performance optimization. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*. ACM Press, 1–12. DOI: http://dx.doi.org/10.1145/2503210.2503252

Sidharth Kumar, Venkatram Vishwanath, Philip Carns, Joshua A. Levine, Robert Latham, Giorgio Scorzelli, Hemanth Kolla, Ray Grout, Robert Ross, Michael E. Papka, Jacqueline Chen, and Valerio Pascucci. 2012. Efficient data restructuring and aggregation for I/O acceleration in PIDX. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (SC'12). IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.54

Sidharth Kumar, Venkatram Vishwanath, Philip Carns, Brian Summa, Giorgio Scorzelli, Valerio Pascucci, Robert Ross, Jacqueline Chen, Hemanth Kolla, and Ray Grout. 2011. PIDX: Efficient parallel I/O for multi-resolution multi-dimensional scientific datasets. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing (CLUSTER'11)*. IEEE, 103–111. DOI: http://dx.doi.org/10.1109/CLUSTER.2011.19

Chih-Song Kuo, Aamer Shah, Akihiro Nomura, Satoshi Matsuoka, and Felix Wolf. 2014. How file access patterns influence interference among cluster applications. In *Proceedings of the 2014 IEEE International Conference on Cluster Computing (CLUSTER'14)*. IEEE, 185–193. DOI: http://dx.doi.org/10.1109/CLUSTER.2014.6968743

Jharrod LaFon, Satyajayant Misra, and Jon Bringhurst. 2012. On distributed file tree walk of parallel file systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.82

Sriram Lakshminarasimhan, David A. Boyuka, Saurabh V. Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E. Papka, and Nagiza F. Samatova. 2013. Scalable in situ scientific data encoding for analytical query processing. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC'13)*. ACM, 1–12. DOI: http://dx.doi.org/10.1145/2462902.2465527

Rob Latham, Neil Miller, Robert Ross, and Phil Carns. 2004. A next-generation parallel file system for Linux clusters. *LinuxWorld Magazine* 2, 1 (2004), 1–11.

Jianwei Li, Wei keng Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. 2003. Parallel netCDF: A high-performance scientific I/O interface. In *Proc. 2003 ACM/IEEE Conference on Supercomputing*. IEEE, 39–49. DOI: http://dx.doi.org/10.1109/SC.2003.10053

Wei-keng Liao. 2011. Design and evaluation of MPI file domain partitioning methods under extent-based file locking protocol. *IEEE Trans. Parallel Distrib. Syst.* 22, 2 (2011), 260–272. DOI: http://dx.doi.org/10.1109/TPDS.2010.74

Heshan Lin, Xiaosong Ma, Wuchun Feng, and Nagiza F. Samatova. 2011. Coordinating computation and I/O in massively parallel sequence search. *IEEE Trans. Parallel Distrib. Syst.* 22, 4 (2011), 529–543. DOI: http://dx.doi.org/10.1109/TPDS.2010.101

Jialin Liu, Yong Chen, and Yu Zhuang. 2013. Hierarchical I/O scheduling for collective I/O. In *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 211–218. DOI: http://dx.doi.org/10.1109/CCGrid.2013.30

Ning Liu, Jason Cope, Philip Carns, Christopher Carothers, Robert Ross, Gary Grider, Adam Crume, and Carlos Maltzahn. 2012. On the role of burst buffers in leadership-class storage systems. In *Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/MSST.2012.6232369

Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S. Vazhkudai. 2014. Automatic identification of application I/O signatures from noisy server-side traces. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14)*. USENIX Association, 213–228. http://dl.acm.org/citation.cfm?id=2591305.2591326

Jay Lofstead, Milo Polte, Garth Gibson, Scott A. Klasky, Karsten Schwan, Ron Oldfield, Matthew Wolf, and Qing Liu. 2011. Six degrees of scientific data: Reading patterns for extreme scale science IO. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. ACM, 49–60. DOI: http://dx.doi.org/10.1145/1996130.1996139

Jay Lofstead and Robert Ross. 2013. Insights for exascale IO APIs from building a petascale IO API. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13)*. ACM Press, 1–12. DOI: http://dx.doi.org/10.1145/2503210.2503238

Jay Lofstead, Fang Zheng, Qing Liu, Scott Klasky, Ron Oldfield, Todd Kordenbrock, Karsten Schwan, and Matthew Wolf. 2010. Managing variability in the IO performance of petascale storage systems. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. IEEE, 1–12. DOI: http://dx.doi.org/10.1109/SC.2010.32

Jeremy Logan, Scott Klasky, Hasan Abbasi, Qing Liu, George Ostrouchov, Manish Parashar, Norbert Podhorszki, Yuan Tian, and Matthew Wolf. 2012. Understanding I/O performance using I/O skeletal applications. In *Euro-Par 2012—Parallel Processing*, Christos Kaklamanis, Theodore Papatheodorou, and Paul G. Spirakis (Eds.). Lecture Notes in Computer Science, Vol. 7484. Springer, 77–88. DOI: http://dx.doi.org/10.1007/978-3-642-32820-6_10

Yin Lu, Yong Chen, Rob Latham, and Yu Zhuang. 2014. Revealing applications' access pattern in collective I/O for cache management. In *Proceedings of the 28th ACM International Conference on Supercomputing (ICS'14)*. ACM Press, 181–190. DOI: http://dx.doi.org/10.1145/2597652.2597686

Youyou Lu, Jiwu Shu, Shuai Li, and Letian Yi. 2012. Accelerating distributed updates with asynchronous ordered writes in a parallel file system. In *Proc. 2012 IEEE International Conference on Cluster Computing (CLUSTER'12)*. IEEE, 302–310. DOI: http://dx.doi.org/10.1109/CLUSTER.2012.38

Jason Mair, Zhiyi Huang, David Eyers, and Yawen Chen. 2015. Quantifying the energy efficiency challenges of achieving exascale computing. In *Proc. IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*. IEEE, 943–950. DOI: http://dx.doi.org/10.1109/CCGrid.2015.130

Adam Manzanares, John Bent, Meghan Wingate, and Garth Gibson. 2012. The power and challenges of transformative I/O. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER'12)*. IEEE, 144–154. DOI: http://dx.doi.org/10.1109/CLUSTER.2012.86

Robert McLay, Doug James, Si Liu, John Cazes, and William Barth. 2014. A user-friendly approach for tuning parallel file operations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. IEEE, 229–236. DOI: http://dx.doi.org/10.1109/SC.2014.24

Dirk Meister, Jürgen Kaiser, Andre Brinkmann, Toni Cortes, Michael Kuhn, and Julian Kunkel. 2012. A study on data deduplication in HPC storage systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.14

Phil Miller, Shen Li, and Chao Mei. 2011. Asynchronous collective output with non-dedicated cores. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing (CLUSTER'11)*. IEEE, 494–502. DOI: http://dx.doi.org/10.1109/CLUSTER.2011.82

David Nagle, Denis Serenyi, and Abbie Matthews. 2004. The panasas activescale storage cluster: Delivering scalable high bandwidth storage. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'04)*. IEEE, 53–62. DOI: http://dx.doi.org/10.1109/SC.2004.57

Thorvald Natvig, Anne C. Elster, and Jan Christian Meyer. 2010. Automatic run-time parallelization and transformation of I/O. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. IEEE, 1–10. DOI: http://dx.doi.org/10.1109/SC.2010.11

Arifa Nisar, Wei-keng Liao, and Alok Choudhary. 2012. Delegation-based I/O mechanism for high performance computing systems. *IEEE Trans. ParallelDistrib. Syst.* 23, 2 (2012), 271–279. DOI: http://dx.doi.org/10.1109/TPDS.2011.166

Kazuki Ohta, Dries Kimpe, Jason Cope, Kamil Iskra, Robert Ross, and Yutaka Ishikawa. 2010. Optimization techniques at the I/O forwarding layer. In *Proceedings of the 2010 IEEE International Conference on Cluster Computing (CLUSTER'10)*. IEEE, 312–321. DOI: http://dx.doi.org/10.1109/CLUSTER.2010.36

Ron Oldfield, Kenneth Moreland, Nathan Fabian, and David Rogers. 2014. Evaluation of methods to integrate analysis into a large-scale shock shock physics code. In *Proceedings of the 28th ACM International Conference on Supercomputing (ICS'14)*. ACM, 83–92. DOI: http://dx.doi.org/10.1145/2597652.2597668

Sarp Oral, Feiyi Wang, David Dillow, Galen Shipman, Ross Miller, and Oleg Drokin. 2010. Efficient object storage journaling in a distributed parallel file system. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*. USENIX Association, 1–12.

Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. 2013. A survey on techniques for improving the energy efficiency of large scale distributed systems. *Comput. Surv.* (2013), 1–35. DOI: http://dx.doi.org/10.1145/2532637

Jiaxin Ou, Jiwu Shu, Youyou Lu, Letian Yi, and Wei Wang. 2014. EDM: An endurance-aware data migration scheme for load balancing in SSD storage clusters. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS'14)*. IEEE, 787–796. DOI: http://dx.doi.org/10.1109/IPDPS.2014.86

Swapnil Patil and Gregory R. Ganger. 2011. Scale and concurrency of GIGA+: File system directories with millions of files. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'11)*. USENIX Association, 1–14.

Christina M. Patrick, Mahmut Kandemir, Mustafa Karaköy, Seung Woo Son, and Alok Choudhary. 2010. Cashing in on hints for better prefetching and caching in PVFS and MPI-IO. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*. ACM, 191–202. DOI: http://dx.doi.org/10.1145/1851476.1851499

David A. Patterson and John L. Hennessy. 2013. *Computer Organization and Design: The Hardware/Software Interface* (Newnes).

Juan Piernas-Canovas and Jarek Nieplocha. 2010. Implementation and evaluation of active storage in modern parallel file systems. *Parallel Comput.* 36, 1 (2010), 26–47. DOI: http://dx.doi.org/10.1016/j.parco.2009.11.002

Ramya Prabhakar, Shekhar Srikantaiah, Mahmut Kandemir, and Christina Patrick. 2010. Adaptive multi-level cache allocation in distributed storage architectures. In *Proceedings of the 24th ACM International Conference on Supercomputing (ICS'10)*. ACM Press, 211–221. DOI: http://dx.doi.org/10.1145/1810085.1810115

Raghunath Rajachandrasekar, Adam Moody, Kathryn Mohror, and Dhabaleswar K. (Dk) Panda. 2013. A 1 PB/s file system to checkpoint three million MPI tasks. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC'13)*. ACM, 143–154. DOI: http://dx.doi.org/10.1145/2462902.2462908

Kai Ren, Qing Zheng, Swapnil Patil, and Garth Gibson. 2014. IndexFS: Scaling file system metadata performance with stateless caching and bulk insertion. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14)*. IEEE, 237–248. DOI: http://dx.doi.org/10.1109/SC.2014.25

Eric Schendel, Saurabh Pendse, John Jenkins, David A. Boyuka II, Zhenhuan Gong, Sriram Lakshminarasimhan, Qing Liu, Hemanth Kolla, Jackie Chen, Scott Klasky, Robert Ross, and Nagiza Samatova. 2012. ISOBAR hybrid compression-I/O interleaving for large-scale parallel I/O optimization. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing (HPDC'12)*. ACM Press, 61–72. DOI: http://dx.doi.org/10.1145/2287076.2287086

Frank Schmuck and Roger Haskin. 2002. GPFS: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*. USENIX Association, Article 19. http://dl.acm.org/citation.cfm?id=1083323.1083349

Seetharami Seelam, I-Hsin Chung, John Bauer, and Hui-fang Wen. 2010. Masking I/O latency using application level I/O caching and prefetching on blue gene systems. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10)*. IEEE, 1–12. DOI: http://dx.doi.org/10.1109/IPDPS.2010.5470438

Carmen Sigovan, Chris Muelder, Kwan-Liu Ma, Jason Cope, Kamil Iskra, and Robert Ross. 2013. A visual network analysis method for large-scale parallel I/O systems. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS'13)*. IEEE, 308–319. DOI: http://dx.doi.org/10.1109/IPDPS.2013.96

Seung Woo Son, Samuel Lang, Philip Carns, Robert B. Ross, Rajeev Thakur, Berkin Ozisikyilmaz, Prabhat Kumar, Wei-Keng Liao, and Alok Choudhary. 2010. Enabling active storage on parallel I/O software stacks. In *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST'10)*. IEEE, 1–12. DOI: http://dx.doi.org/10.1109/MSST.2010.5496981

Huaiming Song, Yanlong Yin, Yong Chen, and Xian-He Sun. 2011a. A cost-intelligent application-specific data layout scheme for parallel file systems. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing (HPDC'11)*. ACM, 37–48. DOI: http://dx.doi.org/10.1145/1996130.1996138

Huaiming Song, Yanlong Yin, Xian-He Sun, Rajeev Thakur, and Samuel Lang. 2011c. A segment-level adaptive data layout scheme for improved load balance in parallel file systems. In *Proceedings of the 11th International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 414–423. DOI: http://dx.doi.org/10.1109/CCGrid.2011.26

Huaiming Song, Yanlong Yin, Xian-He Sun, Rajeev Thakur, and Samuel Lang. 2011b. Server-side I/O coordination for parallel file systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM Press, 1–11. DOI: http://dx.doi.org/10.1145/2063384.2063407

Pei-lun Suei, Mi-yen Yeh, and Tei-wei Kuo. 2014. Endurance-aware flash-cache management for storage servers. *IEEE Trans. Comput.* 63, 10 (2014), 2416–2430. DOI: http://dx.doi.org/10.1109/TC.2013.119

Houjun Tang, Xiaocheng Zou, John Jenkins, David A. Boyuka II, Stephen Ranshous, Dries Kimpe, Scott Klasky, and Nagiza F. Samatova. 2014. Improving read performance with online access pattern analysis and prefetching. In *Euro-Par 2014—Parallel Processing*, F. Silva, I. Dutra, and V. S. Costa (Eds.). Lecture Notes in Computer Science, Vol. 8632. Springer International Publishing, 246–257. DOI: http://dx.doi.org/10.1007/978-3-319-09873-9_21

Wittawat Tantisiriroj, Seung Woo Son, Swapnil Patil, Samuel J. Lang, Garth Gibson, and Robert B. Ross. 2011. On the duality of data-intensive file system design: Reconciling HDFS and PVFS. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. 1–12. DOI: http://dx.doi.org/10.1145/2063384.2063474

Rajeev Thakur, William Gropp, and Ewing Lusk. 1999. Data sieving and collective I/O in ROMIO. In *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation (Frontiers'99)*. IEEE, 182–189. DOI: http://dx.doi.org/10.1109/FMPC.1999.750599

Viet Trung Tran, Bogdan Nicolae, Gabriel Antoniu, and Luc Boug. 2011. Efficient support for MPI-I/O atomicity based on versioning. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'11)*. IEEE, 514–523. DOI: http://dx.doi.org/10.1109/CCGrid.2011.60

Andrew Uselton, Mark Howison, Nicholas J. Wright, David Skinner, Noel Keen, John Shalf, Karen L. Karavanic, and Leonid Oliker. 2010. Parallel I/O performance: From events to ensembles. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'10)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/IPDPS.2010.5470424

Venkatram Vishwanath, Mark Hereld, Kamil Iskra, Dries Kimpe, Vitali Morozov, Michael E. Papka, Robert Ross, and Kazutomo Yoshii. 2010. Accelerating I/O forwarding in IBM blue gene/P systems. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. IEEE, 1–10. DOI: http://dx.doi.org/10.1109/SC.2010.8

Venkatram Vishwanath, Mark Hereld, Vitali Morozov, and Michael E. Papka. 2011. Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM Press, 1–11. DOI: http://dx.doi.org/10.1145/2063384.2063409

Zhixiang Wang, Xuanhua Shi, Hai Jin, Song Wu, and Yong Chen. 2014. Iteration based collective I/O strategy for Parallel I/O systems. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'14)*. IEEE, 287–294. DOI: http://dx.doi.org/10.1109/CCGrid.2014.61

Brent Welch and Geoffrey Noer. 2013. Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions. In *Proceedings of the 29th Symp. on Mass Storage Systems and Technologies (MSST'10)*. IEEE, 1–12. DOI: http://dx.doi.org/10.1109/MSST.2013.6558449

Bing Xie, Jeffrey Chase, David Dillow, Oleg Drokin, Scott Klasky, Sarp Oral, and Norbert Podhorszki. 2012. Characterizing output bottlenecks in a supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.28

Jin Xiong, Yiming Hu, Guojie Li, Rongfeng Tang, and Zhihua Fan. 2011. Metadata distribution and consistency techniques for large-scale cluster file systems. *IEEE Trans. Parallel Distrib. Syst.* 22, 5 (2011), 803–816. DOI: http://dx.doi.org/10.1109/TPDS.2010.154

Weixia Xu, Yutong Lu, Qiong Li, Enqiang Zhou, Zhenlong Song, Yong Dong, Wei Zhang, Dengping Wei, Xiaoming Zhang, Haitao Chen, and others. 2014. Hybrid hierarchy storage system in MilkyWay-2 supercomputer. *Frontiers Comput. Sci.* 8, 3 (2014), 367–377. DOI: http://dx.doi.org/10.1007/s11704-014-3499-6

Yiqi Xu, Dulcardo Arteaga, Ming Zhao, Yonggang Liu, Renato Figueiredo, and Seetharami Seelam. 2012. vPFS: Bandwidth virtualization of parallel storage systems. In *Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies*. IEEE, 1–12. DOI: http://dx.doi.org/10.1109/MSST.2012.6232370

Letian Yi, Jiwu Shu, Ying Zhao, Yingjin Qian, Youyou Lu, and Weimin Zheng. 2014. Design and implementation of an asymmetric block-based parallel file system. *IEEE Trans. Comput.* 63, 7 (2014), 1723–1735. DOI: http://dx.doi.org/10.1109/TC.2013.6

Yanlong Yin, Surendra Byna, Huaiming Song, Xian He Sun, and Rajeev Thakur. 2012. Boosting application-specific parallel I/O optimization using IOSIG. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'12)*. IEEE, 196–203. DOI: http://dx.doi.org/10.1109/CCGrid.2012.136

Yanlong Yin, Jibing Li, Jun He, Xian-He Sun, and Rajeev Thakur. 2013. Pattern-direct and layout-aware replication scheme for parallel I/O systems. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS'13)*. IEEE, 345–356. DOI: http://dx.doi.org/10.1109/IPDPS.2013.114

Yongen Yu, Douglas H. Rudd, Zhiling Lan, Nickolay Y. Gnedin, Andrey Kravtsov, and Jingjin Wu. 2012. Improving parallel IO performance of cell-based AMR cosmology applications. In *Proceedings of the 2012 IEEE International Symposium on Parallel and Distributed Processing (IPDPS'12)*. IEEE, 933–944. DOI: http://dx.doi.org/10.1109/IPDPS.2012.88

Yongen Yu, Jingjin Wu, Zhiling Lan, Douglas H. Rudd, Nickolay Y. Gnedin, and Andrey Kravtsov. 2013. A transparent collective I/O implementation. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS'13)*. IEEE, 297–307. DOI: http://dx.doi.org/10.1109/IPDPS.2013.36

Xuechen Zhang, Kei Davis, and Song Jiang. 2010. IOrchestrator: Improving the performance of multi-node I/O systems via inter-server coordination. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2010.30

Xuechen Zhang, Kei Davis, and Song Jiang. 2011. QoS support for end users of I/O-intensive applications using shared storage systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*. ACM Press, 1–12. DOI: http://dx.doi.org/10.1145/2063384.2063408

Xuechen Zhang, Kei Davis, and Song Jiang. 2012. ITransformer: Using SSD to improve disk scheduling for high-performance I/O. In *Proc. 2012 IEEE International Symposium on Parallel and Distributed Processing (IPDPS'12)*. IEEE, 715–726. DOI: http://dx.doi.org/10.1109/IPDPS.2012.70

Xuechen Zhang and Song Jiang. 2010. InterferenceRemoval: Removing interference of disk access for MPI programs through data replication. In *Proceedings of the 24th ACM International Conference on Supercomputing (ICS'10)*. ACM Press, 223–232. DOI: http://dx.doi.org/10.1145/1810085.1810116

Xuechen Zhang, Ke Liu, Kei Davis, and Song Jiang. 2013. iBridge: Improving unaligned parallel file access with solid-state drives. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS'13)*. IEEE, 381–392. DOI: http://dx.doi.org/10.1109/IPDPS.2013.21

Zhao Zhang, Daniel S. Katz, Michael Wilde, Justin M. Wozniak, and Ian Foster. 2013. MTC envelope: Defining the capability of large scale computers in the context of parallel scripting applications. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC'13)*. ACM Press, 37–48. DOI: http://dx.doi.org/10.1145/2462902.2462913

Zhao Zhang, Daniel S. Katz, Justin M. Wozniak, Allan Espinosa, and Ian Foster. 2012. Design and analysis of data management in scalable parallel scripting. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE, 1–11. DOI: http://dx.doi.org/10.1109/SC.2012.44

Dongfang Zhao, Kan Qiao, and Ioan Raicu. 2014. HyCache+: Towards scalable high-performance caching middleware for parallel file systems. In *Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'14)*. IEEE, 267–276. DOI: http://dx.doi.org/10.1109/CCGrid.2014.11

Hongbo Zou, Xian He Sun, Siyuan Ma, and Xi Duan. 2012. A source-aware interrupt scheduling for modern parallel I/O systems. In *Proc. 2012 IEEE International Symposium on Parallel and Distributed Processing (IPDPS'12)*. IEEE, 156–166. DOI: http://dx.doi.org/10.1109/IPDPS.2012.24

Received January 2016; revised December 2016; accepted October 2017