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

Francieli Zanon Boito, Eduardo Camilo Inacio, Jean Luca Bez, Philippe Navaux, Mario Dantas, Yves Denneulin

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

Francieli Zanon Boito[1], Eduardo Camilo Inacio[2], Jean Luca Bez[1],
Philippe O. A. Navaux[1], Mario A. R. Dantas[2], Yves Denneulin[3]

[1]Institute of Informatics, Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{*francieli.zanon, jlbez, navaux*}*@inf.ufrgs.br*
[2]Department of Informatics and Statistics, Federal University of Santa Catarina
Florianópolis, Brazil
eduardo.camilo@posgrad.ufsc.br, mario.dantas@ufsc.br
[3]Laboratory of Informatics of Grenoble, INRIA, University of Grenoble Alpes
Grenoble, France
yves.denneulin@grenoble-inp.fr

First submitted in December 2015

Abstract

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 latencies, which causes applications 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, 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 five-year time window, we discuss the state of the art of I/O optimization approaches, access pattern extraction techniques, and performance modeling, in addition to general aspects of parallel I/O research. Through this approach, we aim at identifying the general characteristics of the field and the main current and future research topics.

Keywords: Parallel file systems, high performance computing, storage systems.

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 on cluster or massively parallel processing (MPP) architectures. In these supercomputers, the application's workload is separated into multiple processes, executing on different computing nodes. These processes often need access to shared files. *Parallel file systems* (PFS) allow applications processes to access shared files transparently – i.e., without knowledge of where these files are actually stored. Another key characteristic of parallel file systems 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 [82]. Because of that, applications that need to 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 using filters to decrease the amount of output, by the 2020s the LHC is expected to be dealing with exabytes of data [19]. Despite decades of research effort into providing high performance parallel I/O, as applications' needs and data amounts grow, I/O continues to be a central issue on the path to exascale [21].

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 parallel file system. Therefore, we do not discuss storage tools for grid and cloud environments, or for big data processing, such as HDFS [98]. Nonetheless, many of the presented techniques and discussions could be expanded for other storage systems.

Our contributions are twofold: first, we provide the basic concepts so readers which are unfamiliar with the subject can understand what is 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 five years of publications from the field's most important conferences and journals. By exploring the research activity of the last five 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, etc.

The remainder of this article is organized as follows: Section 2 explains the survey methodology, lists the considered conferences and journals, and presents some data on 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 identifying the most used tools for research. The state of the art on techniques to improve performance of parallel file system access is discussed in Section 4, followed by the state of the art in applications' 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

In order 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 five-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 5629 publications – to identify relevant work by looking at title and abstract. 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, 103 **articles remained** (1.8%).

Papers were considered relevant when discussing traditional (POSIX) parallel I/O in an HPC context, even if the presented experimental results do not contemplate this scenario. Moreover, techniques to characterize parallel applications' 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 or journal. It is important to notice that we have not found relevant articles in the ACM Computing Surveys journal, which motivates the present work. In fact, **in our analysis no survey was found**.

[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 for the survey

	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	10 (5.6%)	**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	10 (7.4%)			
SC	396	25 (6.3%)			
TOTAL	2480	91 (3.7%)	**TOTAL**	3149	12 (0.4%)

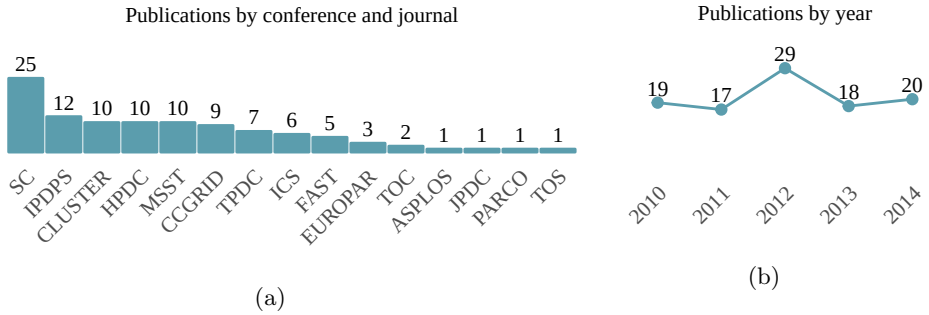
From 103 surveyed papers, 91 (88%) are from conferences and 12 (12%) from journals. These numbers are further explored in Figure 1, that shows the number of selected publications by conference or journal (Figure 1(a)) and the number of publications by year (Figure 1(b)). These numbers indicate that **research 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, MSST, and SC. However, considering the absolute number of relevant publications, the most relevant conferences are SC and IPDPS.

We can also notice from Figure 1(b) that 2012 was particularly productive regarding parallel I/O research, with approximately 60% 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 and MSST also presented "parallel I/O peaks" 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-



(a) (b)

Figure 1: Publications on parallel I/O separated by year and by form of publication

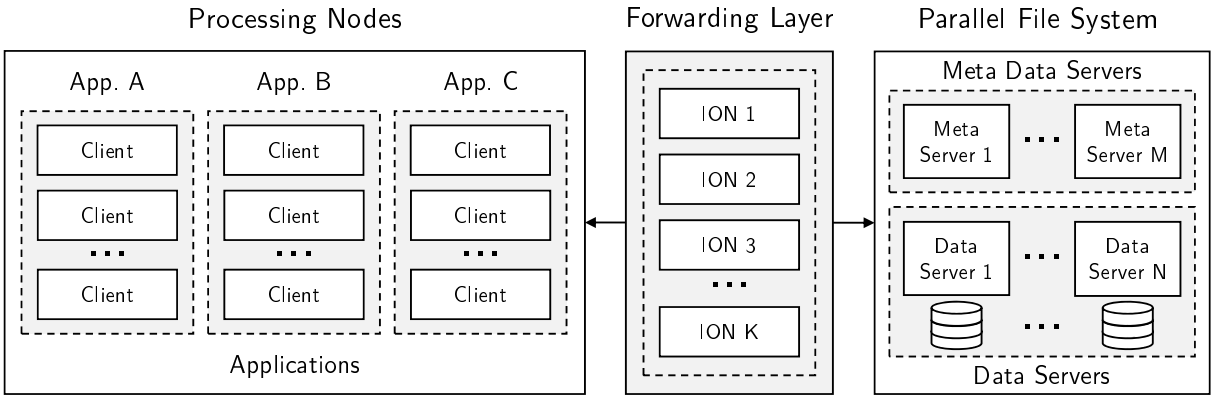


Figure 2: Logical components involved in performing I/O to parallel file systems

scale architectures such as clusters, parallel file systems 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 parallel file systems. 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 by 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 while accessing a parallel file system.

Hard disk drives (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 having better performance when accesses are done sequentially instead of randomly because the former minimizes seek times. [82]

A popular solution for storage in HPC systems is the use of *redundant array of independent disks* (RAID), 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, which improves performance. RAID performance is affected by the combination of stripe size and access size.

Solid state drives (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 [45]. There is a growing adoption of SSDs, although their larger cost per byte still cause many parallel file system deployments in clusters to store data in hard disks.

In addition to storage devices' physical characteristics, performance behavior observed when accessing them 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 devices' access costs. Both these caches typically perform *prefetching* and *read-ahead*, techniques that try to predict data that will be accessed by applications in the future and make these requests earlier. 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 parallel file systems clients' caches, both for data and metadata. The next section discusses parallel file systems.

3.2 Parallel File Systems

Parallel file systems are composed of two specialized servers: the *data server* and the *metadata server*. The latter is responsible for metadata, which is information about data like 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] [53]. Other systems, like Lustre [10], decide not to distribute metadata in order to keep its management simple.

Files are distributed among data servers in an operation called *data striping*. Each file is divided into portions of a fixed size, called *stripes*, and the stripes are given to the servers following a round-robin approach. PFSs 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 [31] employed a 64MB stripe size because its target applications performed very large sequential accesses only. PVFS, Lustre, and GPFS [89] have defaults between 64KB and 1MB.

Some systems apply *locking* on the servers in order 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.

In order to include 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 give an overview of some popular parallel file systems.

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 system and MPI-IO by leveraging the ROMIO implementation. The latter links directly to a low-level PVFS API for access. PVFS also supports distributed metadata, and metadata servers may be collocated with data servers.

The Lustre file system is another open-source PFS, entirely implemented in the Linux kernel. It has three major functional units: metadata servers (MDS), object storage servers (OSS) and clients. MDS nodes have one or more metadata targets (MDT) devices that store metadata in a local file system. OSS nodes store files data on one or more object storage target (OST) devices. The capacity of the PFS is the sum of the capacity of its OSTs. Lustre presents a global POSIX namespace to all clients.

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 for all nodes.

The Panasas file system [71] is divided into storage nodes and manager nodes. The former provide access to object storage devices (OSDs) and are accessed directly from clients during

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

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 (NFS) and Common Internet File System (CIFS).

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 only a very simplified local I/O stack in order 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. [103]

The I/O forwarding technique is applied in several 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 16000 computing nodes do not have a local I/O stack, but instead, all I/O operations are transferred 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 parallel file system [109].

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, etc.

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 their 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. On 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.

The most usual access pattern aspect is *spatiality*. It tells the location of requests into files: contiguous, distant by a fixed value, randomly positioned, etc. 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. Other important aspects usually considered are request size, number of generated/accessed files, intra-node concurrency, time between consecutive requests, arrival rate, and operation (write or read).

3.4.1 I/O Libraries

Parallel file systems usually deploy a client module in processing 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 applications' accesses suit the used system. Nevertheless, this tuning between applications and systems is not easily achieved.

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

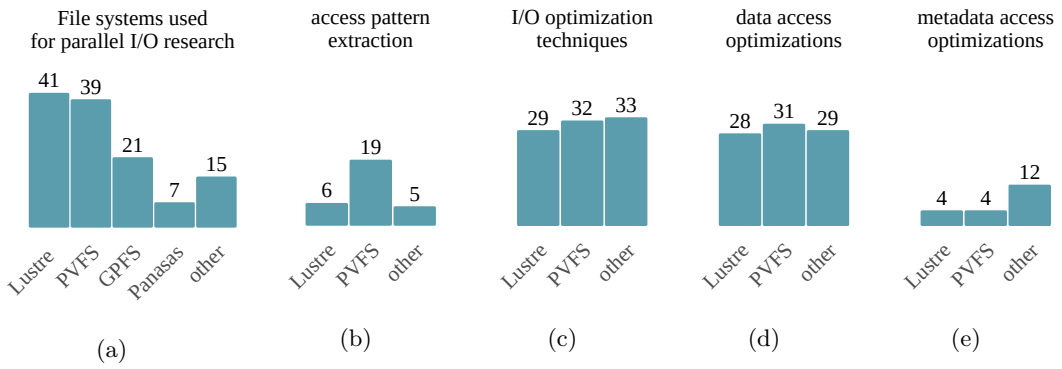


Figure 3: Parallel file system usage among the surveyed publications - in general and separated by technique

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 file system, which would compromise their portability. Moreover, developers would need to know details about the target file 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 [15]. These libraries take charge of applications' I/O operations and have the power to perform optimizations to adapt their access pattern. High-level I/O libraries as HDF5 [100] and netCDF[4] [54] 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 several techniques for I/O performance improvement.

3.5 Surveyed Data

There are already a few well accepted parallel file systems, 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. [111] 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 parallel file system (96 out of 103). 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 3(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 96 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 ten 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 is the most used PFS to prototype and evaluate access pattern extraction techniques and data access optimizations**.

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

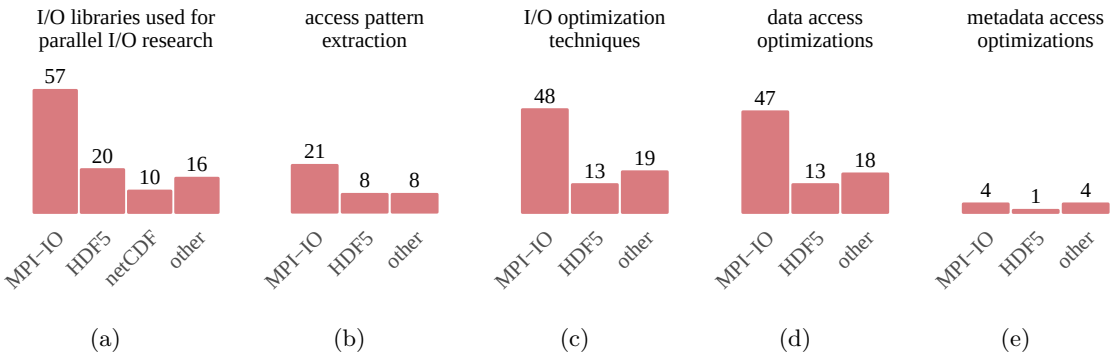


Figure 4: I/O library usage among the surveyed publications - in general and separated by technique

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 (68%) use I/O libraries for their experiments. Figure 4 presents data on I/O library usage among the surveyed papers. Figure 4(a) shows the number of publications using each library, and Figure 4(b) to 4(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 which 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 techniques (48 out of 81) implementations or evaluations have used MPI-IO.

Among articles that present techniques for access pattern extraction (30), 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.

Since applications present different access patterns, parallel I/O researchers often use benchmarks to represent the many possible situations when evaluating their techniques. From the research efforts considered in this survey, 79 (77%) use benchmarks for their experiments. Numbers on benchmark usage among the surveyed publications 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 5(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 (50 out of 79). 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 that parallel I/O researchers 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

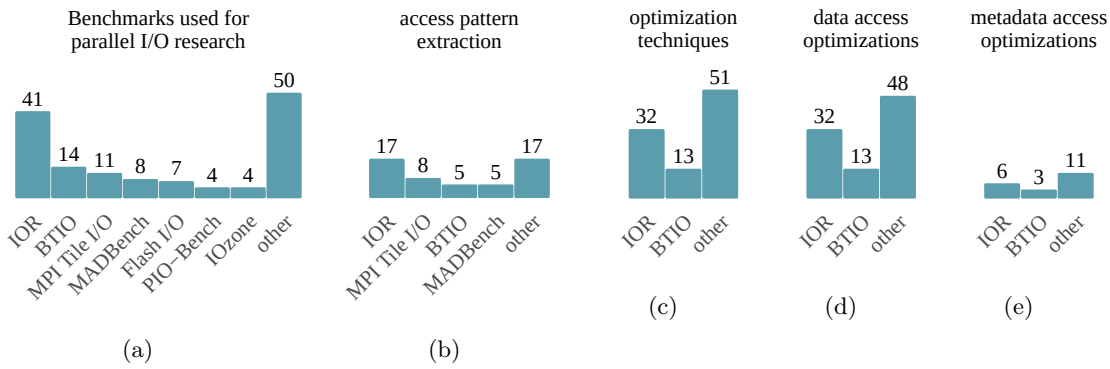


Figure 5: Benchmark usage among the surveyed publications - in general and separated by technique

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 high performance computing, 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 surveyed data do not indicate progress in this aspect.

Finally, 26 (25%) of the papers used checkpointing as a motivation for work on parallel I/O.

4 Techniques to Improve Parallel I/O Performance

Since performance depends on applications access patterns, and some patterns are known to perform better than others in some systems, ways of improving the performance for an application often involve changing its access pattern to make it more suitable for the used system. This can be done at the server side, mainly by modifying the file system; or at the client side, by changing applications, I/O libraries, APIs, etc.

From the studied publications, 81 (79%) 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** (65, 80%). Moreover, most techniques work on the client side only (48, 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 some typical I/O optimizations techniques. Sections 4.1 to 4.5 discuss these optimizations, Section 4.6 presents more surveyed data on them, and Section 4.7 concludes this part of the survey.

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

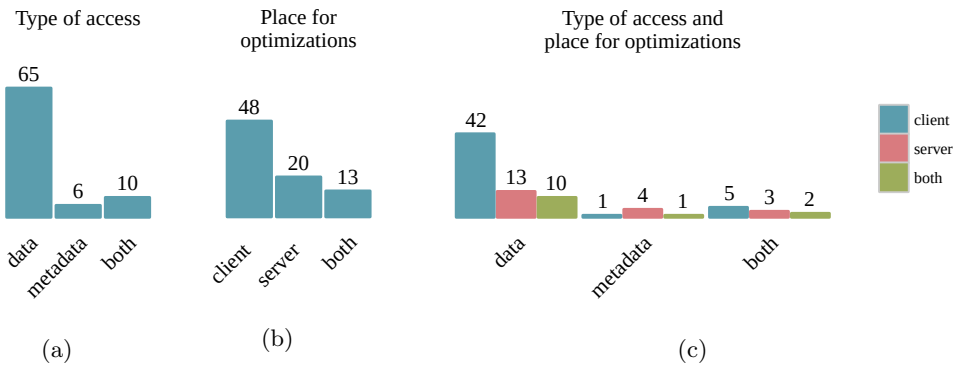


Figure 6: Surveyed data on optimizations - types of access and place for optimizations

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, Zhu, and Jiang [36]. Their approach applies Bloom filters to route metadata accesses among the servers. Patil and Ganger [80] propose a file system directory service called GIGA+, consisting of distributing directories entries among multiple servers using a decentralized hash-based indexing. Consistency on directories indexes in clients' caches is eventually maintained. Another metadata distribution design called IndexFS was proposed by Ren et al. [87], 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. [108] 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 a permanent storage, visualization, post-processing, etc. In this context, LaFon, Misra, and Bringhurst [51] 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. Hence, 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 alterations in a journal, where these modifications are only valid after committed through transactions (with guaranteed atomicity).

Chen, Xiong, and Meng [14] 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 service. This mechanism decreases the impact of server failures and avoids the interruption of service.

Oral et al. [77] 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 block-based symmetric architecture parallel file systems, metadata is often stored along

with data in underlying storage devices, which may compromise reliability. Yi et al. [111] propose an asymmetric file system where metadata are 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 which 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. [66] work to improve performance in such situations. They tackle the ordered writes mechanism, which keeps consistency under distributed writes by ordering involved sub-operations 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 I/O merge.

4.2 Requests Aggregation and Reordering

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

Islam et al. [39] present a checkpoint-restart library that works to coalesce requests from the multiple processes to the PFS. They use information – variables name and type – to place similar data close together. They do so because compression schemes work better with similar data. Vishwanath et al. [104] 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.

One popular technique is the use of *collective operations*, whose idea consists on 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 [99].

The typical two-phase I/O strategy works when the multiple processes access a single file. Kumar et al. [49] implement a two-phase I/O for multiple files in the context of the PIDX library. In a following work, Kumar et al. [48] 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. [47] apply machine learning to automatically tune the library parameters.

Chen et al. [12] 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. [13], 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. [69] demonstrate choosing the appropriate stripe size is critical to collective write performance. They propose some heuristics to facilitate this choice. Wang et al. [105] 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 some reasons. First, the 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 avoids contention caused by the servers' lock mechanism.

Liao [55] 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 in shared file accesses, which typically causes serialization of concurrent I/O operations. Another technique to reduce lock contention is presented by Nisar, Liao, and Choudhary [73]. 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, for a developer to know if collective I/O is advantageous is not always trivial. Due to its added difficulty, many developers do not use this optimization in the development of scientific applications and thus observe poor I/O performance. Natvig, Elster, and Meyer [72] 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 in order to eliminate overlaps and improve performance. With a similar motivation, Yu et al. [114] 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. [121] present a data management middleware for parallel scripting. Their approach also works to automatically generate collective operations.

Performance improvements by collective operations come from both aggregation and reordering of small requests. If made independently by clients, these requests would hardly arrive at 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. Requests reordering is the focus of some optimization techniques.

PLFS is a library that transparently maps applications' 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 for checkpointing). PLFS will then map each process' requests into an independent file, where they are contiguous. Manzanares et al. [68] 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 in order 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. [68] consists of distributing the library files in different points of this tree to avoid a situation where only one of the servers is keeping all of them. Bent et al. [5] point PLFS as one of the key solutions to support storage demands at the exascale era.

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 the use of caches, in the different levels of the I/O stack, to hide the latency of the remote access. The success of caching can be

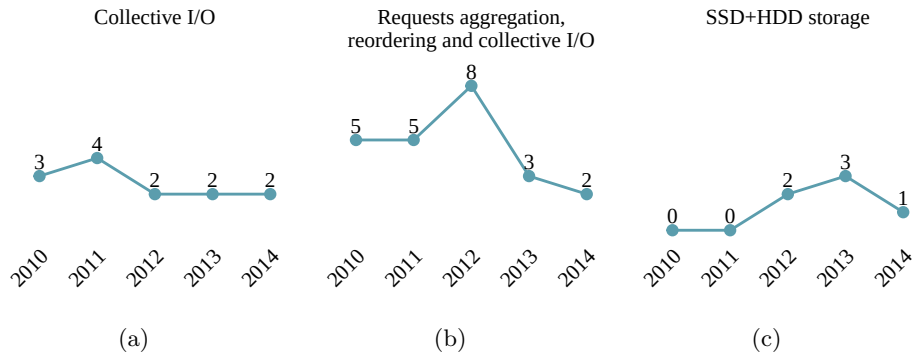


Figure 7: Number of publications that present discussed optimization techniques, separated by year

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. [26] 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. [28] uses prefetching to increase the performance of loading parallel applications with dynamically linked libraries. Rajachandrasekar et al. [86] propose a user-level file system to keep checkpointing requests in the main memory and transparently flush then to persistent storage. Their approach includes support to *remote direct memory access* (RDMA).

Another caching middleware is proposed by Zhao, Qiao, and Raicu [123]. 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. [38] 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. [84] model the optimal cache allocation on two-level cache systems through linear programming.

Kandemir et al. [43] 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. [90] applies a library that traces and detects the application access pattern. This information is used to guide prefetching to a local buffer. Similar approaches - access pattern detection to guide prefetching - are proposed by Patrick et al. [81], He, Sun, and Thakur [32], Lu et al. [65], and Tang et al. [97].

Suei, Yeh, and Kuo [96] 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, Davis, and Jiang [117].

The hybrid SSD+HDD approach is also used by Zhang et al. [120]. 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 [106] store small files in the SSD in order to optimize access to them, as they have observed that small files are the majority in parallel file systems. The approach presented by He et al. [34] 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 are often called "burst buffers", and would work to hide the remote file system latency. A current research topic which has been receiving some attention seeks to determine how to use these burst buffers, where to place them, how to make them transparent, etc. [20]

Bent et al. [4] argue that the intermediate I/O nodes are the best place for burst buffers, since this choice allows to pipeline computation with data movement to servers. The same approach is evaluated by Liu et al. [58] through simulation.

The graph from Figure 7(c) presents the number of publications studying hybrid storage solutions per year. We can see this is a 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, applications performance may be impaired, in a phenomenon called "interference". 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. [17] propose a client-side I/O scheduling approach. They focus on avoiding *stragglers* – data servers which 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 data striping schema in place. Zhang and Jiang [119] identify portions of data which are causing interference during concurrent accesses. These portions are then replicated to other servers to decrease the concurrency.

Dorier, Antoniu, and Ross [23] propose a client-side cross-application coordination strategy. They use information about applications access patterns to dynamically decide between three scheduling strategies, seeking to optimize a given metric. Lofstead et al. [62] propose an adaptive I/O method, implemented in their ADIOS middleware, which monitors the file system performance and balances the workload accordingly.

Liu, Chen, and Zhuang [57] 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 the servers prioritize aggregators with higher shuffle cost in order to provide better overall performance.

Zhang, Davis, and Jiang [116] propose an approach named IOrchestrator to the PVFS parallel file system. 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 [118]. Through a QoS performance interface, requirements can be defined in terms of execution time (deadline). Applications need a profiling execution, where the proposed mechanism obtains the application access pattern. A machine learning technique is used to translate the provided deadline to requirements in bandwidth from the file system.

Song et al. [95] propose a server coordination scheme that also aims at serving one application at a time. They implemented a window-wide coordination strategy where requests are separated in time windows ordered by application ID, and the different windows must be processed in order to avoid starvation.

Vishwanath et al. [103] evaluate performance of the I/O forwarding layer of an IBM Blue Gene/P, and apply a simple *First Come First Served* (FCFS) scheduling algorithm. Ohta et al. [74] 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. [110] to provide proportional sharing between applications.

4.5 Other Techniques

When multiple processes generate requests to the remote file system from the same node, there may be contention in the access to memory and network resources. Dorier et al. [24] 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 actually perform I/O to the parallel file system.

Dong et al. [22] propose a load balancing scheme for PFS data servers. Inadequate striping size (that does not reflect the applications 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. [79]. Their approach focuses on SSDs, taking their characteristics into consideration to perform wear leveling while avoiding write amplification.

Qian et al. [85] present a performance model to estimate Lustre I/O latency and a distributed dynamic congestion I/O mechanism. The mechanism defines the number of I/O requests in flight for each client, based on the servers load. Through this approach, clients performance is improved during light load periods, while the number of re-transmits is decreased during heavy load periods.

Some applications and libraries perform data compression to decrease the amount of data to be accessed in the remote PFS. Jenkins et al. [40] propose an approach that adapts to different precision needs. A parallel data compression approach for I/O libraries is presented by Bicer, Yin, and Agrawal [6]. The approach presented by Filgueira et al. [27] 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. [88] 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 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 (NDP) trend, which tries to avoid data movement impacts on performance, power efficiency, and reliability [2]. The active storage approach is discussed by Piernas-Canovas and Nieplocha [83]. Son et al. [92] discuss how active storage can be implemented in parallel file systems without information about data layout and striping.

An approach to leverage GPU processing power for the PFS client functions is presented by Al-Kiswany, Gharaibeh, and Ripeanu [1]. They use the GPU to compute hash functions, detect block boundaries and calculate blocks hash.

To guarantee consistency under concurrent access, MPI-IO implementations usually employ locking. This approach may present poor performance under a high level of concurrency. Tran et al. [101] 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 applications developers and users to configure systems for better performance. Behzad et al. [3] present an auto-tuning system for HDF5 applications, which selects parameters and hints at runtime.

Jung et al. [41] 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 flash modules.

An approach for automatic layout configuration is proposed by Song et al. [94]. The approach consists of dividing a large file into multiple segments and adopting different layout configurations

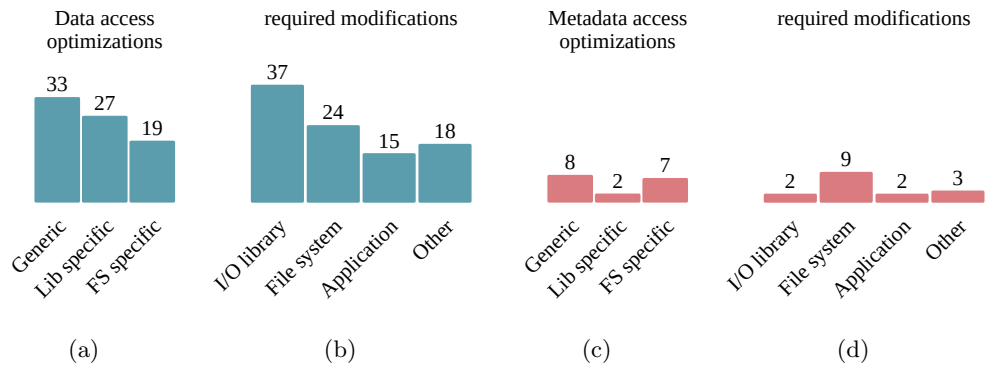


Figure 8: Surveyed data on optimizations - required modifications and how generic proposed techniques are

for each segment according to the observed access pattern. The same authors present a model to estimate data access cost of different data layout policies [93]. Another automatic data placement approach is proposed by Yin et al. [113], through a data replication scheme that reorganizes data according to access patterns.

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 characterize these optimizations. Figure 8(b) and 8(d) show the levels where proposed techniques (for data and metadata access optimization, respectively) require modifications. Similarly 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 75 publications on data access optimization, 37 require modifications to the I/O library, 24 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 [18, 42, 81].

The graphs from Figure 8(a) and 8(c) show how generic the proposed optimization techniques are. In the context of this survey, 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 most 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 facts 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 optimizations 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

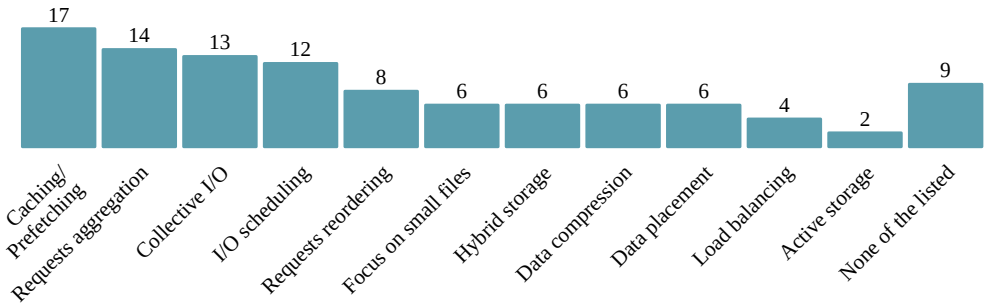


Figure 9: Surveyed publications on data access optimization techniques

publications where none of the listed techniques were used.

The most common technique among the surveyed papers is caching/prefetching. Figure 10(a), 10(b), and 10(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 surveyed papers for caching/prefetching 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 many other implementations are possible.

Similarly, Figure 10(d), 10(e), and 10(f) present surveyed information on publications using the requests aggregation, reordering, and collective I/O techniques (23 papers fall in at least one of the three categories). These numbers demonstrate **requests 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), 10(h), and 10(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 metadata operations 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 operations. Despite being used for many years (two-phase I/O was introduced to ROMIO in the 90s), several 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.

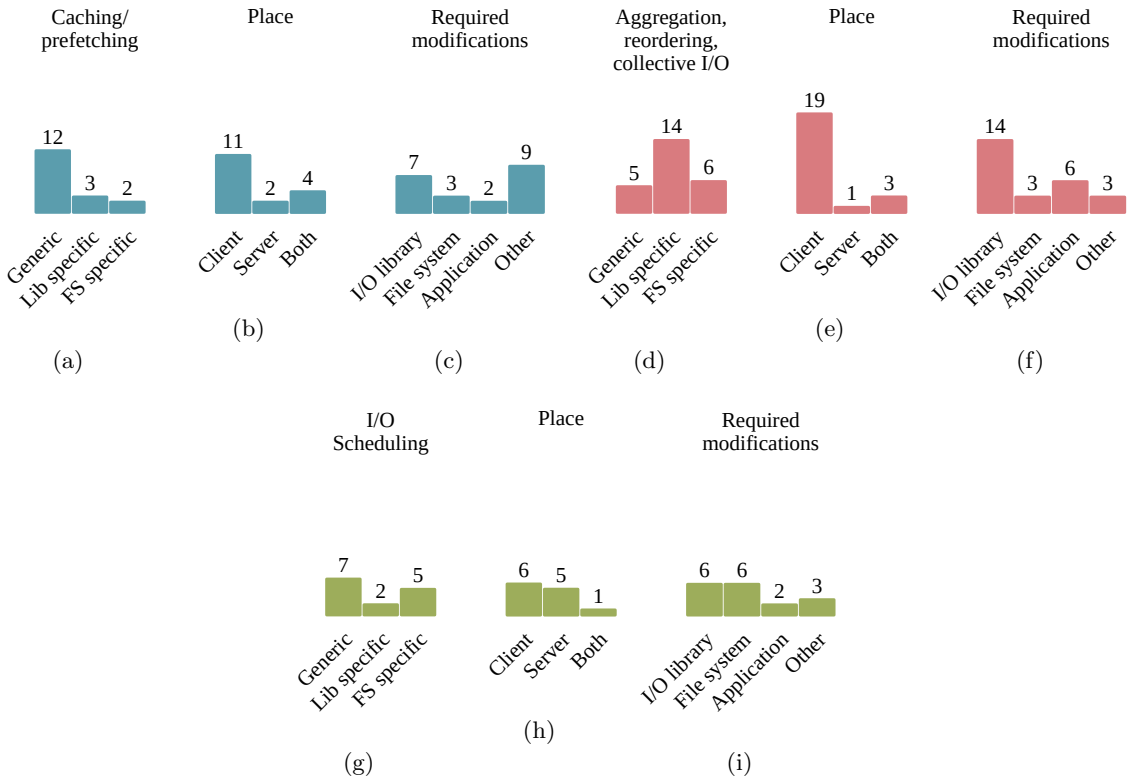


Figure 10: Surveyed data on caching/prefetching, requests aggregation, reordering, collective I/O, and I/O scheduling - place, required modifications and how generic proposed techniques are

Caching is a typical solution to hide latency in 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

Several techniques to improve parallel I/O performance need information about the applications access patterns. Prefetching techniques and cache substitution policies are common examples. Other source of information to optimization techniques is the use of models to represent and predict performance behavior. Models abstract systems, and techniques can explore their parameter

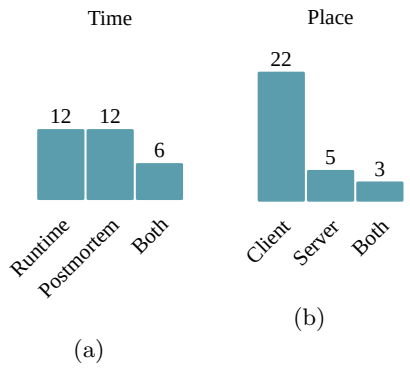


Figure 11: Surveyed data on access pattern extraction - where and when these techniques take place

space to optimize given objectives – e.g., performance, resource utilization, load balancing, etc.

Section 5.1 discusses techniques to obtain information about parallel applications accesses. 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, 30 (29%) discuss techniques to obtain information about the applications 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 approaches publications are less common.

At runtime, techniques can typically only use information from past accesses. A complete analysis could be conducted after the application execution. Moreover, postmortem techniques do not have time constraints as tight as runtime techniques, since the latter are usually required to provide decisions to optimization techniques as fast as possible. Runtime techniques efficiency is important in order to avoid imposing overhead on the system. On the other hand, 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 parallel file system 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 which 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. [25] 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,

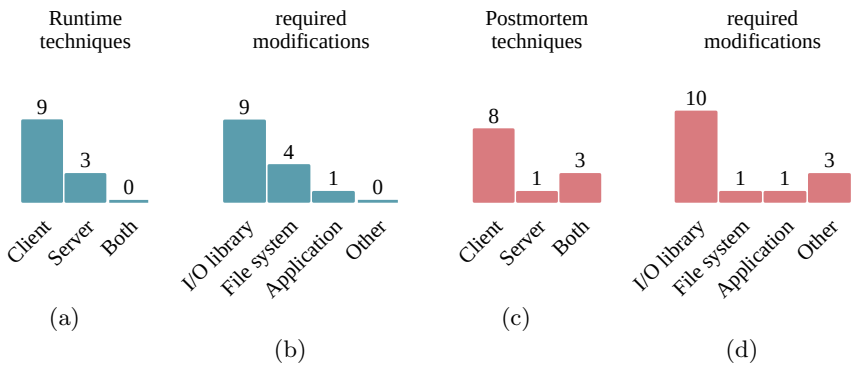


Figure 12: Surveyed data on access pattern extraction techniques that work at runtime or postmortem - where they take place and to which levels they require modifications

offsets and inter-arrival times.

The approach proposed by Tang et al. [97] periodically analyzes past accesses and applies a rules library to predict future accesses (for prefetching). They collect spatiality information about read requests from the MPI-IO library.

It is usual for these techniques to benefit from the information available in I/O libraries. Ge, Feng, and Sun [29] 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, Chen, and Zhuang [57] also collect from MPI-IO the number of processes, the number of aggregators (of collective operations), and binding between nodes and processes. Lu et al. [65] 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. [93]. Similarly, He, Sun, and Thakur [32] 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, etc.

Zou, Zhu, and Feng [125] have analyzed traces from a large Lustre deployment and propose a new stochastic model to adequately predict I/O arrival rate. Dong et al. [22] use a time series model to estimate file system servers' load. The approach by Zhang, Davis, and Jiang [116] applies a "reuse distance", defined as the time difference between consecutive requests from the same application at the servers.

The graphs from Figure 12(a) and 12(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 in the I/O library**.

5.1.2 Hybrid runtime + postmortem approaches

Yin et al. [112] propose IOSIG, a tool that generates applications traces and performs post-mortem analysis to describe their 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 throughout its execution. They propose prefetching and data layout optimizations guided by access patterns, which are identified at runtime with previously obtained information. A similar technique is used by Zhang, Davis, and Jiang [118], where a profiling execution is required to detect applications 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. [81] relies on hints placed in the source code by the application developer and captured by the compiler. He et al. [33] 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. [59] gather information from server-side traces generated by their target architecture. The traces, which contain the system throughput measured every two 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 its I/O requirements throughout its execution.

In the approach proposed by Yin et al. [113], the MPI-IO library was modified to generate traces detailing, to each file operation, MPI rank and 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. [34] use the IOSIG tool to capture trace information 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. [94] 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 [119] 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. [43] automatically instrument applications to delay I/O operations in order 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. [64] differs from the previously discussed publications because they do 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. [91] present a general method for extracting and visualizing 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. [102], statistical analysis of requests time duration, obtained from traces, allows to identify modes and moments of the distribution of I/O times, revealing I/O behavior of applications and potential bottlenecks.

Carns et al. [9] showcase the Darshan profiling tool. They evaluate the I/O performance of a large architecture through two months. Darshan instruments POSIX, MPI-IO, HDF5, and PnetCDF libraries, providing data for a wide range of applications. Liu et al. [58] use Darshan to characterize target applications I/O behaviors.

The graphs from Figure 12(c) and 12(d) show information on postmortem access pattern characterization. Among them, six works aim at providing information to optimization techniques, while the others provide methods for applications 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, 16 (15%) use performance models. **Most of them (14) use a model of the system to drive optimization decisions**. The only exceptions are the papers by Xie et al. [107] and Zhang et al. [122]. Xie et al. [107] 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. [122] 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, Elster, and Meyer [72] use analytical models to estimate the performance of operations and evaluate the benefits of collective I/O. 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 [83]. 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. [88] 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. [94]. The proposed analytical model is used to estimate data access costs to each segment of the file and tune striping. I/O time is computed considering servers 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. [34] 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. [113]. Cost models for three data layout policies are proposed, 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 2-D layout.

In the work by Dong et al. [22], an autoregressive (AR) time series model is used to predict the load of storage nodes. The prediction model is built online in each storage node with load information from local and remote nodes. Model estimates are used to guide the data migration process. Prabhakar et al. [84] use a regression model to estimate per application I/O latency.

Liu et al. [58] integrate an analytical model of burst buffers in the CODES storage system simulator in order to investigate the performance impact of adopting them in the storage infrastructure. Burst buffers data access costs are modeled with device throughput and latency. Rajachandrasekar et al. [86] use a model to estimate throughput of their user-space file system CRUISE, which stores data in main memory and transparently flushes to other persistent storage. Their model considers spill-out to SSDs and considers parameters such as the amount of data and throughput of main memory and SSD.

Machine learning techniques are used by Kumar et al. [47] 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. [52] apply a network-based performance model to estimate indexing, aggregation and I/O times for DIRAQ, a parallel in situ output data indexing and compression technique.

Qian et al. [85] present a performance model to the Lustre file system. The model considers the depth of the server I/O queue, the number of I/O requests in flight, the number of I/O active clients, and operations rate at the server. The proposed model is used to guide a dynamic I/O congestion control mechanism.

5.3 Discussion

This section has discussed two aspects of parallel I/O publications: access pattern extraction and performance modeling, both typically used to guide techniques to improve parallel I/O performance. Additionally, six (out of 30) surveyed publications on access pattern extraction did not use information to guide optimizations but focused on methodologies to evaluate and profile applications I/O performance.

Access pattern extraction techniques can work at runtime, after the applications execution (postmortem), or through a hybrid solution with 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 at 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 requests details such as operation type, size, and spatiality; and sometimes processes information such as 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, operations arrival rate and current file system load. Postmortem techniques work mostly with traces: 14 surveyed publications use them, 9 for postmortem analysis and 5 for hybrid solutions. In addition to other information available at I/O libraries, traces provide file system access times, which allow for deriving elaborate information like file regions access cost.

14 of the surveyed papers (14%) of the surveyed papers propose optimizations based on prediction models. It is worth noticing the trade-off between models 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 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

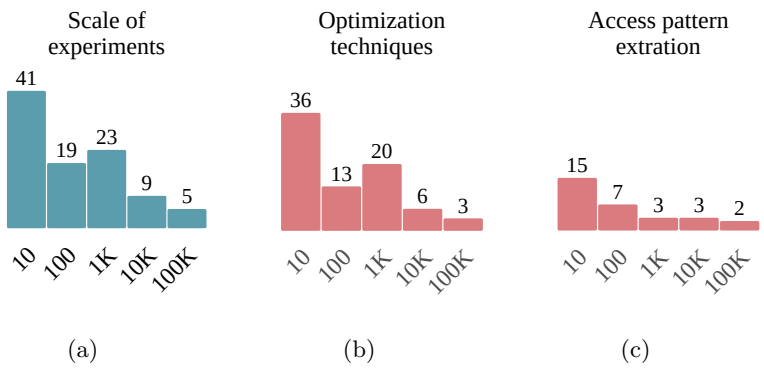
Most published contributions to the parallel I/O field come from universities or research institutions, as from the surveyed publications only 14 (13%) of them had authors from a company. Among these, nine were collaborations among a company and a university or laboratory. The companies with more publications are EMC Corporation and IBM, with three papers each, followed by Intel, with two.

Table 2: Number of surveyed publications per country

	USA	China	France	Germany	Japan	Spain	Canada	Taiwan	UK	Norway	Qatar	Singapore	South Korea
Publications	90	10	4	5	3	3	2	2	2	1	1	1	1

Table 2 shows the number of publications per country. The numbers do not add up to 103 because a paper can be authored by institutions from multiple countries. This is the case of 22 (21%) of them. From these, most of them (18) 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 are involved in most of the surveyed publications** (90, 87%). **The most active USA state on parallel I/O research during the considered time window was Illinois**, with 46 papers, followed by California and New Mexico, with 19 publications each.



(a) (b) (c)

Figure 13: Scale of the surveyed experiments - in general and among publications from the USA

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 most productions.

In addition to the USA leadership position in the field, another thing to be noticed from the presented numbers is the fact that most surveyed research come from developed countries which make large investments on research. One reason for this is that parallel I/O in the high performance computing 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 study. Since only the size of the largest presented experiment of 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 experiments 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)), that shows 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 on 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 on extreme scale, and not every researcher has access to supercomputers. Among the surveyed publications, over a third mentioned the use of famous Top500 machines. Four of them focused on the machine's evaluation and lessons learned with it. These works 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, researchers 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 are advantageous before proceeding with them. Despite being widely used in fields such as microprocessors design and memory hierarchy, **simulators are not usual among parallel I/O publications**. From the surveyed papers, only seven of them use simulation: three for validating data access optimization techniques, two for validating metadata access optimizations, and two propose new simulators. Liu et al. [60] present PFSSim, a parallel file system simulator inspired on PVFS. They provide an interface for implementing I/O scheduling

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

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

algorithms, and "proxy" nodes which can be modeled to reflect an I/O forwarding scheme. Costa et al. [16] 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 actual workloads and environments.

In addition to the machine's 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 in order to account for this noise. Among the 97 articles presenting experimental results, only 29 (30%) of them[9] presented some statistics-related methodology information such as the number of repetitions, standard deviation, confidence interval, or variability. Most of them do not report a formal experimental design. These research efforts do not appear to handle experiments variability, and this is an issue because it could compromise how valid these publications' results are and their reproducibility.

One main reason for this somewhat common practice may be related to the cost of the experiments. More replications for each evaluated scenario means more time consumed, making this approach prohibitive sometimes. The problem is in fact inherent to the I/O field, as data access experiments tend 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 high performance computing community for decades, motivated by the everlasting gap between processing and data access speeds and by increases in HPC architectures' scale and thus in applications' I/O requirements. As the HPC community works towards the exascale 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 five 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 five 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 parallel file systems (PVFS for research mostly), 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 high performance computing – would facilitate the comparison between techniques and strengthen the presented validations. Although an effort towards standardization was done approximately ten 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 which requires work is parallel I/O simulation. From 97 papers presenting experiments, only seven of them use some kind of simulation, two of those propose new simulators. Although we can find a few simulators in the literature, it is unusual to find papers which 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. In our opinion, one of the central issues to achieve accurate I/O simulation is accurate storage devices simulation. As our results on practical aspects of parallel I/O research have pointed the importance of large-scale experiments, we believe some research effort should be employed to reach high quality parallel I/O simulation in order to facilitate new techniques' development and validation.

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

An advantage of good and widely available simulators would be to allow I/O experiments which are less subject to variability. Although these experiments suffer from noise in 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 during 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, what suggests there is now less research activity on these topics as before. That is the case of requests 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 are often called 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 believed 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 [46, 67]. 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 [11, 78]. 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 remove direct memory access (RDMA). RDMA is a technique where one process can directly access other processes' memory without involvement of their operating systems [30]. The use of RDMA has the potential to decrease the negative impact of communication on storage performance and is a possible 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 information on access patterns into optimization techniques allows for more intelligent customized solutions which achieve better performance. We have shown most of the proposed techniques work at client side by interacting with the I/O library. This is a popular approach for optimizations because at 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 I/O 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 in 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 applications 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.

Acknowledgments

This research has been partially supported by CAPES-BRAZIL through the project PVEA117-2013.

References

[1] Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. "GPUs as storage system accelerators". In: *IEEE Transactions on Parallel and Distributed Systems* 24.8 (2013), pp. 1556–1566. ISSN: 10459219. eprint: 1202.3669.

[2] Rajeev Balasubramonian et al. "Near-data processing: Insights from a MICRO-46 workshop". In: *IEEE Micro* 34.4 (2014), pp. 36–42.

[3] Babak Behzad et al. "Taming parallel I/O complexity with auto-tuning". In: *SC '13 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2013, pp. 1–12. ISBN: 9781450323789. DOI: 10.1145/2503210.2503278.

[4] John Bent et al. "Jitter-free co-processing on a prototype exascale storage stack". In: *MSST '12 Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies*. IEEE, 2012, pp. 1–5. ISBN: 978-1-4673-1747-4. DOI: 10.1109/MSST.2012.6232382.

[5] John Bent et al. "Storage challenges at Los Alamos National Lab". In: *MSST '12 Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies*. IEEE, 2012, pp. 1–5. ISBN: 978-1-4673-1747-4. DOI: 10.1109/MSST.2012.6232376.

[6] Tekin Bicer, Jian Yin, and Gagan Agrawal. "Improving I/O Throughput of Scientific Applications Using Transparent Parallel Compression". In: *CCGRID '14 Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 1–10. ISBN: 978-1-4799-2784-5.

[7] Surendra Byna et al. "Parallel I/O, analysis, and visualization of a trillion particle simulation". In: *SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–12. ISBN: 9781467308069. DOI: 10.1109/SC.2012.92.

[8] Philip Carns et al. "Understanding and improving computational science storage access through continuous characterization". In: *MSST '11 Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies*. IEEE, 2011, pp. 1–14. ISBN: 978-1-4577-0427-7.

[9] Philip Carns et al. "Understanding and Improving Computational Science Storage Access through Continuous Characterization". In: *ACM Transactions on Storage* 7.3 (2011), pp. 1–26. ISSN: 15533077.

[10] CFS. *Lustre: A Scalable, High-Performance File System*. Whitepaper. Cluster File Systems, Inc., 2002. URL: http://www.lustre.org/docs/whitepaper.pdf.

[11] Raghunath Raja Chandrasekar et al. “Power-check: An energy-efficient checkpointing framework for HPC clusters”. In: *Proceedings - IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2015*. IEEE, 2015, pp. 261–270. ISBN: 9781479980062. DOI: 10.1109/CCGrid.2015.169.

[12] Yong Chen et al. “Improving Parallel I/O Performance with Data Layout Awareness”. In: *CLUSTER ’10 Proceedings of the 2010 IEEE International Conference on Cluster Computing*. IEEE, 2010, pp. 302–311. ISBN: 978-1-4244-8373-0.

[13] Yong Chen et al. “LACIO: A new collective I/O strategy for parallel I/O systems”. In: *IPDPS ’11 Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 794–804. ISBN: 9780769543857. DOI: 10.1109/IPDPS.2011.79.

[14] Zhuan Chen, Jin Xiong, and Dan Meng. “Replication-based highly available metadata management for cluster file systems”. In: *CLUSTER ’10 Proceedings of the 2010 IEEE International Conference on Cluster Computing*. IEEE, 2010, pp. 292–301. ISBN: 9780769542201. DOI: 10.1109/CLUSTER.2010.34.

[15] P. Corbett et al. “Overview of the MPI-IO parallel I/O interface”. In: *Input/Output in Parallel and Distributed Computer Systems*. Ed. by Ravi Jain, John Werth, and James C. Browne. Vol. 362. The Kluwer International Series in Engineering and Computer Science. Springer US, 1996, pp. 127–146.

[16] Lauro Beltrão Costa et al. “Supporting storage configuration for I/O intensive workflows”. In: *ICS ’14 Proceedings of the 28th ACM international conference on Supercomputing*. ACM Press, 2014, pp. 191–200. ISBN: 9781450326421.

[17] Dong Dai et al. “Two-Choice Randomized Dynamic I/O Scheduler for Object Storage Systems”. In: *SC ’14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 635–646. ISBN: 978-1-4799-5500-8. DOI: 10.1109/SC.2014.57.

[18] Wei Ding et al. “Compiler-directed file layout optimization for hierarchical storage systems”. In: *SC ’12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, 41:1–41:11. ISBN: 978-1-4673-0806-9.

[19] DOE. *Data Crosscutting Requirements Review*. Tech. rep. U.S. Department of Energy, 2013.

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

[21] DOE/NNSA. *Preliminary Conceptual Design for an Exascale Computing Initiative*. Tech. rep. U.S. Department of Energy and National Nuclear Security Administration, 2014.

[22] Bin Dong et al. “A dynamic and adaptive load balancing strategy for parallel file system with large-scale I/O servers”. In: *Journal of Parallel and Distributed Computing* 72.10 (2012), pp. 1254–1268. DOI: 10.1016/j.jpdc.2012.05.006.

[23] Matthieu Dorier, Gabriel Antoniu, and Robert Ross. “CALCioM: Mitigating I/O interference in HPC systems through cross-application coordination”. In: *IPDPS ’14 Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 155–164. ISBN: 9780769552071. DOI: 10.1109/IPDPS.2014.27.

[24] Matthieu Dorier et al. “Damaris: How to efficiently leverage multicore parallelism to achieve scalable, jitter-free I/O”. In: *CLUSTER ’12 Proceedings of the 2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 155–163. ISBN: 978-0-7695-4807-4. DOI: 10.1109/CLUSTER.2012.26.

[25] Matthieu Dorier et al. “Omnisc’IO: A Grammar-Based Approach to Spatial and Temporal I/O Patterns Prediction”. In: *SC ’14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014. DOI: 10.1109/SC.2014.56.

[26] Marc Eshel et al. “Panache: A Parallel File System Cache for Global File Access”. In: *FAST ’10 Proceedings of the 8th USENIX conference on File and storage technologies*. USENIX Association, 2010, pp. 1–14.

[27] Rosa Filgueira et al. “Applying Selectively Parallel I/O Compression to Parallel Storage Systems”. In: *Euro-Par 2014 – Parallel Processing*. Ed. by Fernando Silva, Inês Dutra, and Vitor Santos Costa. Vol. 8632. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 282–293.

[28] W. Frings et al. “Massively parallel loading”. In: *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*. ICS ’13. ACM Press, 2013, pp. 389–398. ISBN: 9781450321303.

[29] Rong Ge, Xizhou Feng, and Xian He Sun. “SERA-IO: Integrating energy consciousness into parallel I/O middleware”. In: *CCGRID ’12 Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2012, pp. 204–211. ISBN: 9780769546919. DOI: 10.1109/CCGrid.2012.39.

[30] Robert Gerstenberger, Maciej Besta, and Torsten Hoefler. “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, 2013, pp. 1–12. DOI: 10.3233/SPR-140383.

[31] S. Ghemawat, H. Gobioff, and S. T. Leung. “The Google file system”. In: *ACM SIGOPS Operating Systems Review* 37.5 (2003), p. 43.

[32] Jun He, Xian-He Sun, and Rajeev Thakur. “KNOWAC: I/O Prefetch via Accumulated Knowledge”. In: *2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 429–437. ISBN: 978-0-7695-4807-4.

[33] Jun He et al. “I/O Acceleration with Pattern Detection”. In: *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing*. HPDC ’13. ACM Press, 2013, pp. 25–36. ISBN: 978-1-4503-1910-2.

[34] Shuibing He et al. “A cost-aware region-level data placement scheme for hybrid parallel I/O systems”. In: *CLUSTER ’13 Proceedings of the 2013 IEEE International Conference on Cluster Computing*. IEEE, 2013, pp. 1–8. ISBN: 978-1-4799-0898-1.

[35] Robert Henschel et al. “Demonstrating Lustre over a 100Gbps wide area network of 3,500km”. In: *SC ’12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, 6:1–6:8. ISBN: 978-1-4673-0806-9.

[36] Yu Hua, Yifeng Zhu, and Hong Jiang. “Supporting scalable and adaptive metadata management in ultralarge-scale file systems”. In: *IEEE Trans. on Parallel and Distributed Systems* 22.4 (2011), pp. 580–593. DOI: 10.1109/TPDS.2010.116.

[37] Thomas Ilsche et al. “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, 2012, pp. 49–60. ISBN: 9781450308052.

[38] Florin Isaila et al. “Design and evaluation of multiple-level data staging for blue gene systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 22.6 (2011), pp. 946–959. DOI: 10.1109/TPDS.2010.127.

[39] Tanzima Islam et al. "McrEngine: A scalable checkpointing system using data-aware aggregation and compression". In: *SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11. ISBN: 9781467308069. DOI: 10.3233/SPR-130371.

[40] John Jenkins et al. "Byte-precision level of detail processing for variable precision analytics". In: *SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11. ISBN: 9781467308069. DOI: 10.1109/SC.2012.26.

[41] Myoungsoo Jung et al. "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, 2014, pp. 441–454. ISBN: 9781450323055.

[42] Mahmut Kandemir et al. "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, 2010, pp. 179–190. ISBN: 9781605589428.

[43] Mahmut Kandemir et al. "On urgency of I/O operations". In: *CCGRID '12 Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2012, pp. 188–195. ISBN: 9780769546919. DOI: 10.1109/CCGrid.2012.40.

[44] Michael P. Kasick et al. "Black-box problem diagnosis in parallel file systems". In: *FAST '10 Proceedings of the 8th USENIX conference on File and storage technologies*. USENIX Association, 2010, pp. 1–14.

[45] Jaehong Kim et al. "Parameter-aware I/O management for solid state disks (SSDs)". In: *IEEE Transactions on Computers* 61.5 (2012), pp. 636–649.

[46] P. Kogge et al. *Exascale Computing Study: Technology Challenges in achieving Exascale Systems*. Tech. rep. Defense Advanced Research Projects Agency (DARPA IPTO), 2008.

[47] Sidharth Kumar et al. "Characterization and modeling of PIDX parallel I/O for performance optimization". In: *SC '13 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2013, pp. 1–12. ISBN: 9781450323789. DOI: 10.1145/2503210.2503252.

[48] Sidharth Kumar et al. "Efficient data restructuring and aggregation for I/O acceleration in PIDX". In: *SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11. ISBN: 978-1-4673-0806-9.

[49] Sidharth Kumar et al. "PIDX: Efficient parallel I/O for multi-resolution multi-dimensional scientific datasets". In: *CLUSTER '11 Proceedings of the 2011 IEEE International Conference on Cluster Computing*. IEEE, 2011, pp. 103–111. ISBN: 9780769545165. DOI: 10.1109/CLUSTER.2011.19.

[50] Chih-Song Kuo et al. "How file access patterns influence interference among cluster applications". In: *CLUSTER '14 Proceedings of the 2014 IEEE International Conference on Cluster Computing*. IEEE, 2014, pp. 185–193. ISBN: 978-1-4799-5548-0.

[51] Jharrod LaFon, Satyajayant Misra, and Jon Bringhurst. "On distributed file tree walk of parallel file systems". In: *SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, 87:1–87:11. ISBN: 9781467308069. DOI: 10.1109/SC.2012.82.

[52] Sriram Lakshminarasimhan et al. "Scalable in situ scientific data encoding for analytical query processing". In: *HPDC '13 Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM, 2013, pp. 1–12. ISBN: 978-1-4503-1910-2.

[53] Rob Latham et al. “A Next-Generation Parallel File System for Linux Clusters”. In: *LinuxWorld Magazine* 2.1 (2004), pp. 1–11.

[54] Jianwei Li et al. “Parallel netCDF: A High-Performance Scientific I/O Interface”. In: *Supercomputing, 2003 ACM/IEEE Conference*. SC ’03. 2003, pp. 39–50. ISBN: 1-58113-695-1.

[55] Wei-keng Liao. “Design and Evaluation of MPI File Domain Partitioning Methods under Extent-Based File Locking Protocol”. In: *IEEE Transactions on Parallel and Distributed Systems* 22.2 (2011), pp. 260–272. ISSN: 1045-9219.

[56] Heshan Lin et al. “Coordinating Computation and I/O in Massively Parallel Sequence Search”. In: *IEEE Transactions on Parallel and Distributed Systems* 22.4 (2011), pp. 529–543. ISSN: 1045-9219.

[57] Jialin Liu, Yong Chen, and Yu Zhuang. “Hierarchical I/O scheduling for collective I/O”. In: *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2013, pp. 211–218. ISBN: 978-0-7695-4996-5. DOI: 10.1109/CCGrid.2013.30.

[58] Ning Liu et al. “On the role of burst buffers in leadership-class storage systems”. In: *MSST ’12 Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies*. IEEE, 2012, pp. 1–11. ISBN: 978-1-4673-1747-4.

[59] Yang Liu et al. “Automatic Identification of Application I/O Signatures from Noisy Server-Side Traces”. In: *FAST ’14 Proceedings of the 12th USENIX conference on File and Storage Technologies*. USENIX Association, 2014, pp. 213–228. ISBN: ISBN 978-1-931971-08-9.

[60] Yonggang Liu et al. “On the design and implementation of a simulator for parallel file system research”. In: *MSST ’13 Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies*. IEEE, 2013, pp. 1–5. ISBN: 978-1-4799-0218-7. DOI: 10.1109/MSST.2013.6558438.

[61] Jay Lofstead and Robert Ross. “Insights for exascale IO APIs from building a petascale IO API”. In: *SC ’13 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2013, pp. 1–12. ISBN: 9781450323789. DOI: 10.1145/2503210.2503238.

[62] Jay Lofstead et al. “Managing Variability in the IO Performance of Petascale Storage Systems”. In: *SC ’10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–12. ISBN: 978-1-4244-7557-5.

[63] Jay Lofstead et al. “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, 2011, pp. 49–60.

[64] Jeremy Logan et al. “Understanding I/O Performance Using I/O Skeletal Applications”. In: *Euro-Par 2012 – Parallel Processing*. Ed. by Christos Kaklamanis, Theodore Papatheodorou, and Paul G. Spirakis. Vol. 7484. Lecture Notes in Computer Science. Springer, 2012, pp. 77–88. DOI: 10.1007/978-3-642-32820-6_10.

[65] Yin Lu et al. “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, 2014, pp. 181–190. ISBN: 9781450326421.

[66] Youyou Lu et al. “Accelerating Distributed Updates with Asynchronous Ordered Writes in a Parallel File System”. In: *CLUSTER ’12 Proceedings of the 2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 302–310. ISBN: 978-0-7695-4807-4.

[67] Jason Mair et al. “Quantifying the energy efficiency challenges of achieving exascale computing”. In: *Proceedings - IEEE/ACM 15th Int. Symp. on Cluster, Cloud, and Grid Computing, CCGrid 2015*. IEEE, 2015, pp. 943–950. ISBN: 9781479980062. DOI: 10.1109/CCGrid.2015.130.

[68] Adam Manzanares et al. “The power and challenges of transformative I/O”. In: *CLUSTER ’12 Proceedings of the 2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 144–154. ISBN: 978-0-7695-4807-4. DOI: 10.1109/CLUSTER.2012.86.

[69] Robert McLay et al. “A User-Friendly Approach for Tuning Parallel File Operations”. In: *SC ’14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 229–236. ISBN: 978-1-4799-5500-8.

[70] Dirk Meister et al. “A study on data deduplication in HPC storage systems”. In: *SC ’12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, 7:1–7:11. ISBN: 9781467308069. DOI: 10.1109/SC.2012.14.

[71] David Nagle, Denis Serenyi, and Abbie Matthews. “The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage”. In: *Proceedings of the ACM/IEEE Conference on Supercomputing*. SC ’04. IEEE, 2004, pp. 53–62. ISBN: 0-7695-2153-3.

[72] Thorvald Natvig, Anne C. Elster, and Jan Christian Meyer. “Automatic run-time parallelization and transformation of I/O”. In: *SC ’10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–10. ISBN: 9781424475575. DOI: 10.1109/SC.2010.11.

[73] Arifa Nisar, Wei-keng Liao, and Alok Choudhary. “Delegation-Based I/O Mechanism for High Performance Computing Systems”. In: *IEEE Trans. on Parallel and Distributed Systems* 23.2 (2012), pp. 271–279. ISSN: 1045-9219.

[74] Kazuki Ohta et al. “Optimization techniques at the I/O forwarding layer”. In: *CLUSTER ’10 Proceedings of the 2010 IEEE International Conference on Cluster Computing*. IEEE, 2010, pp. 312–321. ISBN: 9780769542201. DOI: 10.1109/CLUSTER.2010.36.

[75] Ron Oldfield et al. “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, 2014, pp. 83–92. ISBN: 9781450326421.

[76] Sarp Oral et al. “Best Practices and Lessons Learned from Deploying and Operating Large-Scale Data-Centric Parallel File Systems”. In: *SC ’14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 217–228. ISBN: 9781479955008. DOI: 10.1109/SC.2014.23.

[77] Sarp Oral et al. “Efficient object storage journaling in a distributed parallel file system”. In: *FAST ’10 Proceedings of the 8th USENIX conference on File and storage technologies*. USENIX Association, 2010, pp. 1–12.

[78] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. “A Survey on Techniques for Improving the Energy Efficiency of Large Scale Distributed Systems”. In: *ACM Computing Surveys* (2013), pp. 1–35. DOI: 10.1145/2532637.

[79] Jiaxin Ou et al. “EDM: An Endurance-Aware Data Migration Scheme for Load Balancing in SSD Storage Clusters”. In: *IPDPS ’14 Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 787–796. ISBN: 978-1-4799-3800-1. DOI: 10.1109/IPDPS.2014.86.

[80] Swapnil Patil and Gregory R. Ganger. “Scale and Concurrency of GIGA+: File system directories with Millions of files”. In: *FAST ’11 Proceedings of the 9th USENIX conference on File and Storage Technologies*. USENIX Association, 2011, pp. 1–14.

[81] Christina M. Patrick et al. “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, 2010, pp. 191–202. ISBN: 9781605589428.

[82] David A. Patterson and John L. Hennessy. *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[83] Juan Piernas-Canovas and Jarek Nieplocha. “Implementation and evaluation of active storage in modern parallel file systems”. In: *Parallel Computing* 36.1 (2010), pp. 26–47. DOI: 10.1016/j.parco.2009.11.002.

[84] Ramya Prabhakar et al. “Adaptive Multi-level Cache Allocation in Distributed Storage Architectures”. In: *ICS ’10 Proceedings of the 24th ACM International Conference on Supercomputing*. ACM Press, 2010, pp. 211–221. ISBN: 9781450300186.

[85] Yingjin Qian et al. “Dynamic I/O congestion control in scalable lustre file system”. In: *MSST ’13 Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies*. IEEE, 2013, pp. 1–5. ISBN: 978-1-4799-0218-7.

[86] Raghunath Rajachandrasekar et al. “A 1 PB/s file system to checkpoint three million MPI tasks”. In: *HPDC ’13 Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM, 2013, pp. 143–154. ISBN: 978-1-4503-1910-2.

[87] Kai Ren et al. “IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion”. In: *SC ’14 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 237–248. ISBN: 978-1-4799-5500-8.

[88] Eric Schendel et al. “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, 2012, pp. 61–72. ISBN: 9781450308052.

[89] Frank Schmuck and Roger Haskin. “GPFS: A Shared-Disk File System for Large Computing Clusters”. In: *FAST ’02 Proceedings of the First USENIX Conference on File and Storage Technologies*. USENIX Association, 2002, pp. 231–244. ISBN: 1-880446-03-0.

[90] Seetharami Seelam et al. “Masking I/O Latency using Application Level I/O Caching and Prefetching on Blue Gene Systems”. In: *IPDPS ’10 Proceedings of the IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2010, pp. 1–12. ISBN: 9781424464432. DOI: 10.1109/IPDPS.2010.5470438.

[91] Carmen Sigovan et al. “A Visual Network Analysis Method for Large-Scale Parallel I/O Systems”. In: *IPDPS ’13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 308–319. ISBN: 978-1-4673-6066-1.

[92] Seung Woo Son et al. “Enabling Active Storage on Parallel I/O Software Stacks”. In: *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies*. MSST ’10. IEEE, 2010, pp. 1–12. ISBN: 978-1-4244-7152-2. DOI: 10.1109/MSST.2010.5496981.

[93] Huaiming Song et al. “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, 2011, pp. 37–48.

[94] Huaiming Song et al. “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, 2011, pp. 414–423. ISBN: 978-1-4577-0129-0.

[95] Huaiming Song et al. “Server-side I/O coordination for parallel file systems”. In: *SC ’11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2011. ISBN: 9781450307710. DOI: 10.1145/2063384.2063407.

[96] Pei-lun Suei, Mi-yen Yeh, and Tei-wei Kuo. “Endurance-Aware Flash-Cache Management for Storage Servers”. In: *IEEE Transactions on Computers* 63.10 (2014), pp. 2416–2430. DOI: 10.1109/TC.2013.119.

[97] Houjun Tang et al. “Improving Read Performance with Online Access Pattern Analysis and Prefetching”. In: *Euro-Par 2014 – Parallel Processing*. Ed. by F. Silva, I. Dutra, and V. S. Costa. Vol. 8632. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 246–257. DOI: 10.1007/978-3-319-09873-9_21.

[98] Wittawat Tantisiriroj et al. “On the duality of data-intensive file system design: reconciling HDFS and PVFS”. In: *SC ’11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2011.

[99] Rajeev Thakur, William Gropp, and Ewing Lusk. “Data sieving and collective I/O in ROMIO”. In: *FRONTIERS ’99 Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*. IEEE, 1999, pp. 182–189. ISBN: 0-7695-0087-0.

[100] The HDF Group. *HDF5 - Hierarchical Data Format, version 5*. 1997-2016.

[101] Viet Trung Tran et al. “Efficient support for MPI-I/O atomicity based on versioning”. In: *CCGRID ’11 Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2011, pp. 514–523. ISBN: 9780769543956. DOI: 10.1109/CCGrid.2011.60.

[102] Andrew Uselton et al. “Parallel I/O performance: From events to ensembles”. In: *IPDPS ’10 Proceedings of the IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2010, pp. 1–11. ISBN: 978-1-4244-6442-5.

[103] Venkatram Vishwanath et al. “Accelerating I/O Forwarding in IBM Blue Gene/P Systems”. In: *SC ’10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010. DOI: 10.1109/SC.2010.8.

[104] Venkatram Vishwanath et al. “Topology-aware data movement and staging for I/O acceleration on Blue Gene/P supercomputing systems”. In: *SC ’11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2011, pp. 1–11. ISBN: 978-1-4503-0771-0. DOI: 10.1145/2063384.2063409.

[105] Zhixiang Wang et al. “Iteration based collective I/O strategy for Parallel I/O systems”. In: *CCGRID ’14 Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 287–294. ISBN: 9781479927838. DOI: 10.1109/CCGrid.2014.61.

[106] Brent Welch and Geoffrey Noer. “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, 2013, pp. 1–12. ISBN: 978-1-4799-0218-7. DOI: 10.1109/MSST.2013.6558449.

[107] Bing Xie et al. “Characterizing output bottlenecks in a supercomputer”. In: *SC ’12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, 8:1–8:11. ISBN: 978-1-4673-0806-9.

[108] Jin Xiong et al. “Metadata distribution and consistency techniques for large-scale cluster file systems”. In: *IEEE Transactions on Parallel and Distributed Systems* 22.5 (2011), pp. 803–816. DOI: 10.1109/TPDS.2010.154.

[109] Weixia Xu et al. “Hybrid hierarchy storage system in MilkyWay-2 supercomputer”. In: *Frontiers of Computer Science* 8.3 (2014), pp. 367–377.

[110] Yiqi Xu et al. “vPFS: Bandwidth virtualization of parallel storage systems”. In: *MSST ’12 Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies*. IEEE, 2012, pp. 1–12. ISBN: 978-1-4673-1747-4.

[111] Letian Yi et al. “Design and Implementation of an Asymmetric Block-Based Parallel File System”. In: *IEEE Transactions on Computers* 63.7 (2014), pp. 1723–1735. ISSN: 0018-9340. DOI: 10.1109/TC.2013.6.

[112] Yanlong Yin et al. “Boosting application-specific parallel I/O optimization using IOSIG”. In: *CCGRID ’12 Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2012, pp. 196–203. ISBN: 9780769546919. DOI: 10.1109/CCGrid.2012.136.

[113] Yanlong Yin et al. “Pattern-Direct and Layout-Aware Replication Scheme for Parallel I/O Systems”. In: *IPDPS ’13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 345–356. ISBN: 978-1-4673-6066-1.

[114] Yongen Yu et al. “A Transparent Collective I/O Implementation”. In: *IPDPS ’13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 297–307. ISBN: 978-1-4673-6066-1.

[115] Yongen Yu et al. “Improving parallel IO performance of cell-based AMR cosmology applications”. In: *IPDPS ’12 Proceedings of the 2012 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2012, pp. 933–944. ISBN: 9780769546759.

[116] Xuechen Zhang, Kei Davis, and Song Jiang. “IOrchestrator: Improving the performance of multi-node I/O systems via inter-server coordination”. In: *SC ’10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010. ISBN: 9781424475575. DOI: 10.1109/SC.2010.30.

[117] Xuechen Zhang, Kei Davis, and Song Jiang. “ITransformer: Using SSD to improve disk scheduling for high-performance I/O”. In: *IPDPS ’12 Proceedings of the 2012 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2012, pp. 715–726. ISBN: 9780769546759.

[118] Xuechen Zhang, Kei Davis, and Song Jiang. “QoS Support for End Users of I / O-intensive Applications using Shared Storage Systems”. In: *SC ’11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM Press, 2011. ISBN: 9781450307710. DOI: 10.1145/2063384.2063408.

[119] Xuechen Zhang and Song Jiang. “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, 2010, pp. 223–232. ISBN: 9781450300186.

[120] Xuechen Zhang et al. “iBridge: Improving Unaligned Parallel File Access with Solid-State Drives”. In: *IPDPS ’13 Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 381–392. ISBN: 978-1-4673-6066-1. DOI: 10.1109/IPDPS.2013.21.

[121] Zhao Zhang et al. “Design and analysis of data management in scalable parallel scripting”. In: *SC ’12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, 85:1–85:11. ISBN: 978-1-4673-0804-5.

[122] Zhao Zhang et al. “MTC envelope: defining the capability of large scale computers in the context of parallel scripting applications”. In: *HPDC ’13 Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*. ACM Press, 2013, pp. 37–48. ISBN: 978-1-4503-1910-2.

[123] Dongfang Zhao, Kan Qiao, and Ioan Raicu. "HyCache+: Towards scalable high-performance caching middleware for Parallel file systems". In: *CCGRID '14 Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2014, pp. 267–276. ISBN: 9781479927838.

[124] Hongbo Zou et al. "A source-aware interrupt scheduling for modern parallel I/O systems". In: *IPDPS '12 Proceedings of the 2012 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2012, pp. 156–166. ISBN: 9780769546759.

[125] Qiang Zou, Yifeng Zhu, and Dan Feng. "A Study of Self-similarity in Parallel I/O Workloads". In: *MSST '10 Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies*. IEEE, 2010, pp. 1–6. ISBN: 978-1-4244-7152-2. DOI: 10.1109/MSST.2010.5496978.