Chapter 8
High-Performance Storage Support for Scientific Big Data Applications on the Cloud

Dongfang Zhao, Akash Mahakode, Sandip Lakshminarasaiah and Ioan Raicu

8.1 Introduction

While cloud computing has become one of the most prevailing paradigms for big data applications, many legacy scientific applications are still struggling to leverage this new paradigm. One challenge for scientific big data applications to be deployed on the cloud lies in the storage subsystem. Popular file systems such as HDFS [1] are designed for many workloads in data centers that are built with commodity hardware. Nevertheless, many scientific applications deal with a large number of small files [2]—a workload that is not well supported by the data parallelism provided by HDFS. The root cause to the storage discrepancy between scientific applications and many commercial applications on cloud computing stems from their original design goals. Scientific applications assume their data to be stored in remote parallel file systems, and cloud platforms provide node-local storage available on each virtual machine.

This chapter shares our views on how to design storage systems for scientific big data applications on the cloud. Based on the literature and our own experience on big data, cloud computing, and high-performance computing (HPC) in the last decade, we believe that cloud storage would need to provide the following three essential services for scientific big data applications:

D. Zhao (✉)
University of Washington, Seattle, WA, USA
e-mail: dzhao@cs.washington.edu

A. Mahakode · S. Lakshminarasaiah · I. Raicu
Illinois Institute of Technology, Chicago, IL, USA
e-mail: amahakod@hawk.iit.edu

S. Lakshminarasaiah
e-mail: slakshm6@hawk.iit.edu

I. Raicu
e-mail: iraicu@cs.iit.edu

F. Pop et al. (eds.), *Resource Management for Big Data Platforms*,
Computer Communications and Networks, DOI 10.1007/978-3-319-44881-7_8

1. Scalable metadata accesses. Conventional centralized mechanisms for managing metadata on cloud computing, such as GFS [3] and HDFS [1], would not suffice for the extensive metadata accesses of scientific big data applications.

2. Optimized data write. Due to the nature of scientific big data applications, checkpointing is the de facto approach to achieve fault tolerance. This implies that the underlying storage system is expected to be highly efficient on data write as checkpointing itself involves frequent data write.

3. Localized file read. When a failure occurs, some virtual machines (VM) need to restart. Instead of transferring VM images from remote file systems, it would be better to keep a local copy of the image and load it from the local disk if at all possible.

In order to justify the above arguments, we analyze four representative file systems. Two of them are originated from cloud computing (S3FS [4], HDFS [1]). S3FS is built on top of the S3 storage offered by Amazon EC2 cloud as a remote shared storage with the added POSIX support with FUSE [5]. HDFS is an open-source clone of Google File System (GFS [3]) without POSIX support. The other two file systems were initially designed for high-performance computing (Ceph [6], FusionFS [7, 8]). Ceph employs distributed metadata management and the CRUSH [9] algorithm to balance the load. FusionFS is first introduced in [10] and supports several unique features such as erasure coding [11], provenance [12], caching [13, 14], compression [15, 16], and serialization [17]. This study involves two test beds: a conventional cluster Kodiak [18] and a public cloud Amazon EC2 [19].

The remainder of this chapter is organized as follows. Section 8.2 discusses the scalability of metadata accesses. We present the design and performance of achieving optimized file write and localized file read in Sects. 8.3 and 8.4, respectively. Section 8.5 details a real system that employs the proposed design principles as well as unique features in caching, compression, GPUs, provenance, and serialization. We review important literature in big data systems and HPC systems in Sect. 8.6 and finally conclude this chapter in Sect. 8.7.

8.2 Scalable Metadata Accesses

State-of-the-art distributed file systems on cloud computing, such as HDFS [1], still embrace the decade-old design of a centralized metadata server. The reason of such a design is due to the workload characteristic in data centers. More specifically, a large portion of workloads in data centers involve mostly large files. For instance, HDFS has a default 64 MB chunk size (typically 128 MB though), which implicitly implies that the target workload has many files larger than 64 MB; HDFS is not designed or optimized for files smaller than 64 MB. Because many large files are expected, the metadata accesses are not intensive and one single metadata server in many cases is sufficient. In other words, a centralized metadata server in the conventional workloads of cloud computing is not a performance bottleneck.

The centralized design of metadata service, unfortunately, would not meet the requirement of many HPC applications that deal with a larger number of

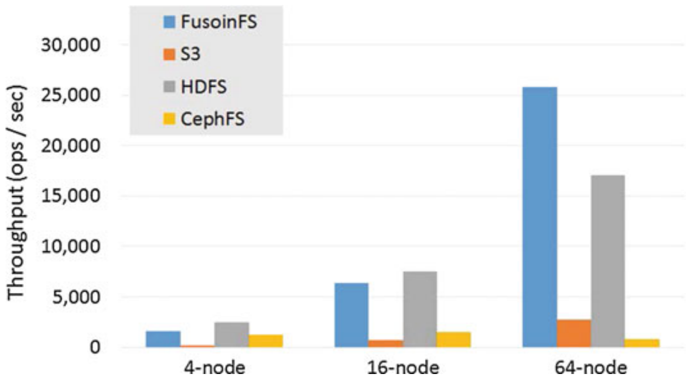

Fig. 8.1 Metadata performance comparison

concurrent metadata accesses. HPC applications are, in nature, highly different than those conventionally deployed on cloud platforms. One of the key differences is file sizes. For instance, Welch and Noer [20] report that 25–90 % of all the 600 million files from 65 Panasas [21] installations are 64 KB or smaller. Such a huge number of small files pose a significantly higher pressure to the metadata server than the cloud applications. A single metadata server would easily become the bottleneck in these metadata-intensive workloads.

A distributed approach to manage metadata seems to be the natural choice for scientific applications on the cloud. Fortunately, several systems (for example, [6, 7]) have employed this design principle. In the remainder of this section, we pick FusionFS and HDFS as two representative file systems to illustrate the importance of a distributed metadata service under intensive metadata accesses. Before discussing the experiment details, we provide a brief introduction of the metadata management of both systems.

HDFS, as a clone of the Google File System [3], has a logically[1] single metadata server (i.e., namenode).The replication of the namenode is for fault tolerance rather than balancing the I/O pressure. That is, all the metadata requests are directed to the single namenode—a simple, yet effective design decision for the cloud workloads. FusionFS is designed to support extremely high concurrency of metadata accesses. It achieves this goal by dispersing metadata to as many nodes as possible. This might be overkill for small- to medium-scale applications, but is essential for those metadata-intensive workloads that are common in scientific applications.

On Amazon EC2, we compare the metadata performance of all four file systems, i.e., FusionFS, S3, HDFS, and CephFS. The workload we use is asking each client to write 10,000 empty files to the according file system. Results are reported in Fig. 8.1.

There are a few observations worth further discussing. First, HDFS outperforms other peers on four nodes and scales well toward 16 nodes. And yet, its scalability is not as good as FusionFS, whose metadata throughput is significantly higher than

[1] because it gets replicated on multiple nodes, physically.

HDFS although the former delivers a lower throughput on four nodes. Second, S3 scales well but is hardly competitive compared to other systems because only with 64 nodes its performance becomes comparable to others on four nodes. Third, CephFS's scalability is poor even from 4 to 16 nodes. Even worse, its performance is degraded when scaling from 16 to 64 nodes.

8.3 Optimized Data Write

Data write is one of the most common I/O workloads in scientific applications due to their de facto mechanism to achieve fault tolerance—checkpointing. Essentially, checkpointing asks the system to periodically persist its memory states to the disks, which involves a larger number of data writes. The persisted data only need to be loaded (i.e., read) after a failure occurs in a completely nondeterministic manner. As the system is becoming increasingly larger, the time interval between consecutive checkpoints is predicted to be dramatically smaller in future systems. [22] From storage's perspective, cloud platform will have to provide highly efficient data write throughput for scientific applications.

Unfortunately, HDFS could hardly provide optimized data write due to the metadata limitation discussed in Sect. 8.2. Figure 8.2 shows the write throughput of FusionFS and HDFS on Kodiak. Similarly to the metadata trend, the write throughput of HDFS also suffers poor scalability beyond 128 nodes.

Another option in cloud platforms is the remote shared storage. It usually provides a unified interface and scalable I/O performance for applications. One example is the S3 storage on Amazon EC2 cloud. S3 does not only provide a set of API but also leverages FUSE [5] to serve as a fully POSIX-compliant file system named S3FS. Therefore S3FS is becoming a popular replacement of the conventional remote shared file systems [23, 24] in HPC.

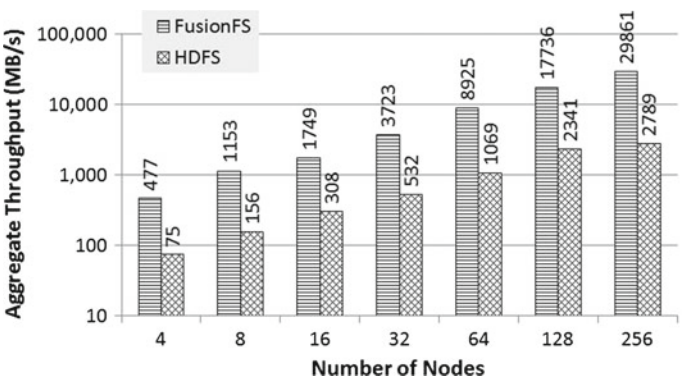

Fig. 8.2 Write throughput of FusionFS and HDFS are compared

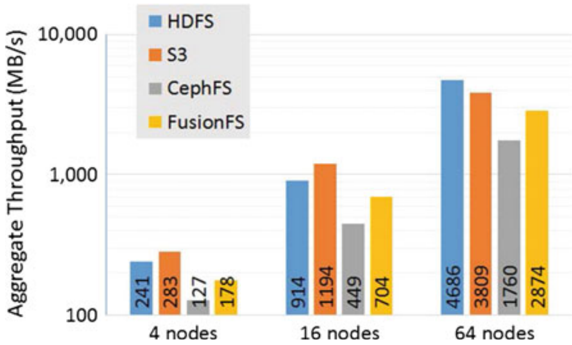

Fig. 8.3 Scalable write throughput on Amazon EC2

We compare all the file systems in discussion so far on the same testbed, i.e., m3.large instance on Amazon EC2. The experiment is in modest scale, from four nodes to 16 nodes, and to 64 nodes in a weak-scaling manner. That is, every node works on the same amount of data—in this case, writing a hundred of 100 MB files to the respective file system.

From Fig. 8.3 we observe that all these systems scale well up to 64 nodes. Note that HDFS and S3 were designed for data centers and cloud computing, while CephFS and FusionFS targeted at scientific applications and high-performance computing. While CephFS is relatively slower than others, FusionFS performs faster comparatively to HDFS and S3 on Amazon EC2 even though FusionFS was not originally designed for data centers. With FusionFS as an example, we believe in the near future a gradual convergence, from the perspective of storage and file system, is emerging between communities of cloud computing and high-performance computing.

We also compare these systems with respect to different file sizes, as shown in Fig. 8.4. Our results show that starting from 1 MB, file size affects little to the write performance on all systems. However, we observe two dramatical extremes on 1 KB

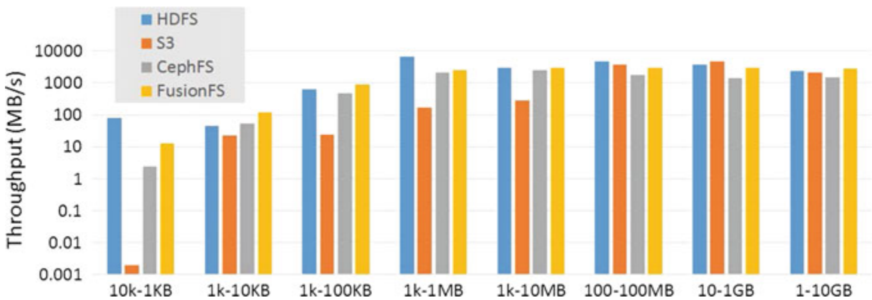

Fig. 8.4 Write throughput of different file sizes

files: HDFS achieves an impressing overall throughput on these small files while S3 is extremely slow. This indicates that for applications where small files dominate, HDFS is in favor with regards to performance.

8.4 Localized File Read

File read throughput is an important metric and is often underestimated since a lot of effort is put on data write as discussed in Sect. 8.3. When a VM is booted or restarted, the image needs to be loaded into the memory and this is becoming a challenging problem in many cloud platforms [25, 26]. Therefore a scalable read throughput is highly desirable for the cloud storage, which urges us to revisit the conventional architecture where files are typically read from remote shared file systems. In HPC this means that the remote parallel file system such as GPFS and Lustre, and in cloud platforms such as Amazon EC2 it implies the remote S3 storage, or the S3FS file system.

We compare the read performance of all the file systems on the m3.large instance of Amazon EC2. Similarly, we scale the experiment from four nodes to 16 nodes, and to 64 nodes in a weak-scaling manner. Every node read a hundred of 100 MB files from the respective file system. Figure 8.5 shows that all these systems scale well up to 64 nodes.

We also compare the systems of interest with respect to their block sizes. In Fig. 8.6, we let each system read different number of files of various sizes, all on 64 Amazon EC2 m3.large instances. For example, "1 k–100 KB" means the system writes 1,000 files of 100 KB.

We observe that for all systems, once the file size is 1 MB and beyond, the read throughput is relatively stable, meaning the I/O bandwidth is saturated. We also note that S3 performs significantly worse than others for files of size 1 KB, although it quickly catches up others at 10 KB and beyond. This suggests that S3 would not be an ideal medium for applications where small files dominate.

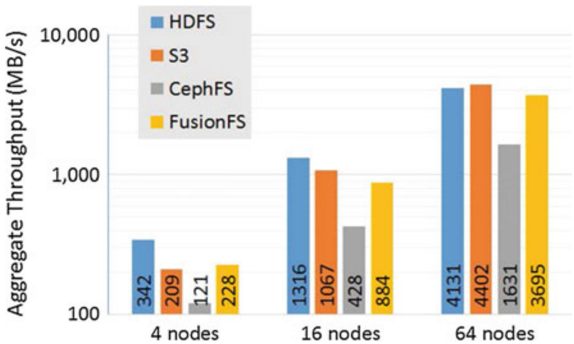

Fig. 8.5 Scalable read throughput on Amazon EC2

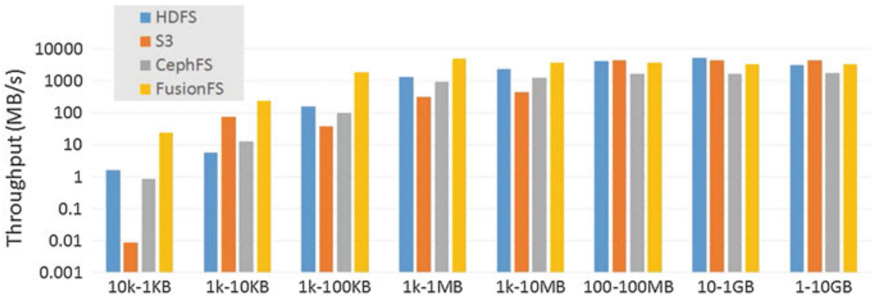

Fig. 8.6 Read throughput of various file sizes

8.5 Put It Altogether: The FusionFS Filesystem

In the previous three sections we discuss three main design criteria for the next-generation HPC storage system on the cloud. This section will present a real system, namely FusionFS, that implements all the aforementioned designs as well as its unique features such as cooperative caching, GPU acceleration, dynamic compression, lightweight provenance, and parallel serialization.

8.5.1 Metadata Management

FusionFS has different data structures for managing regular files and directories. For a regular file, the field *addr* stores the node where this file resides. For a directory, there is a field *filelist* to record all the entries under this directory. This *filelist* field is particularly useful for providing an in-memory speed for directory read such as "ls/mnt/fusionfs". Nevertheless, both regular files and directories share some common fields, such as timestamps and permissions, which are commonly found in traditional i-nodes.

The metadata and data on a local node are completely decoupled: a regular file's location is independent of its metadata location. This flexibility allows us to apply different strategies to metadata and data management, respectively. Moreover, the separation between metadata and data has the potential to plug in alternative components to metadata or data management, making the system more modular.

Besides the conventional metadata information for regular files, there is a special flag in the value indicating if this file is being written. Specifically, any client who requests to write a file needs to set this flag before opening the file, and will not reset it until the file is closed. The atomic compare-swap operation supported by DHT [27, 28] guarantees the file consistency for concurrent writes.

Another challenge on the metadata implementation is on the large-directory performance issues. In particular, when a large number of clients write many small files on the same directory concurrently, the value of this directory in the key-value pair gets incredibly long and responds extremely slowly. The reason is that a client needs to update the entire old long string with the new one, even though the majority of the old string is unchanged. To fix that, we implement an atomic append operation that asynchronously appends the incremental change to the value. This approach is similar to Google File System [3], where files are immutable and can only be appended. This gives us excellent concurrent metadata modification in large directories, at the expense of potentially slower directory metadata read operations.

8.5.2 File Write

Before writing to a file, the process checks if the file is being accessed by another process. If so, an error number is returned to the caller. Otherwise the process can do one of the following two things: If the file is originally stored on a remote node, the file is transferred to the local node in the *fopen()* procedure, after which the process writes to the local copy. If the file to be written is right on the local node, or it is a new file, then the process starts writing the file just like a system call.

The aggregate write throughput is obviously optimal because file writes are associated with local I/O throughput and avoids the following two types of cost: (1) the procedure to determine to which node the data will be written, normally accomplished by pinging the metadata nodes or some monitoring services, and (2) transferring the data to a remote node. It should be clear that FusionFS works at the file level, thus chunking the file is not an option. Nevertheless, we will support chunk-level data movement in the next release of FusionFS. The downside of this file write strategy is the poor control on the load balance of compute node storage. This issue could be addressed by an asynchronous re-balance procedure running in the background, or by a load-aware task scheduler that steals tasks from the active nodes to the more idle ones.

When the process finishes writing to a file that is originally stored in another node, FusionFS does not send the newly modified file back to its original node. Instead, the metadata of this file is updated. This saves the cost of transferring the file data over the network.

8.5.3 File Read

Unlike file write, it is impossible to arbitrarily control where the requested data reside for file read. The location of the requested data is highly dependent on the I/O pattern. However, we could determine which node the job is executed on by the distributed workflow system such as Swift [29]. That is, when a job on node A needs to read some data on node B, we reschedule the job on node B. The overhead of rescheduling the job is typically smaller than transferring the data over the network, especially for data-intensive applications. In our previous work [30], we detailed this approach, and justified it with theoretical analysis and experiments on benchmarks and real applications.

Indeed, remote readings are not always avoidable for some I/O patterns such as merge sort. In merge sort, the data need to be joined together, and shifting the job cannot avoid the aggregation. In such cases, we need to transfer the requested data from the remote node to the requesting node.

8.5.4 Hybrid and Cooperative Caching

When the node-local storage capacity is limited, remote parallel filesystems should coexist with FusionFS to store large-sized data. In some sense, FusionFS is regarded as a caching middleware between the main memory and remote parallel filesystems. We are interested in what placement policies (i.e., caching strategies) are beneficial to HPC workloads.

Our first attempt is a user-level caching middle where on every compute node, assuming a memory-class device (for example, SSD) is accessible along with a conventional spinning hard drive. That is, each compute node is able to manipulate data on hybrid storage systems. The middleware, named HyCache [14], speeds up HDFS by up to 28 %.

Our second attempt is a cooperative caching mechanism across all the compute nodes, called HyCache+ [13]. HyCache+ extends HyCache in terms of network storage support, higher data reliability, and improved scalability. In particular, a two-stage scheduling mechanism called 2-Layer Scheduling (2LS) is devised to explore the data locality of cached data on multiple nodes. HyCache+ delivers two orders of magnitude higher throughput than the remote parallel filesystems, and 2LS outperforms conventional LRU caching by more than one order of magnitude.

8.5.5 Accesses to Compressed Data

Conventional data compression embedded in filesystems naively applies the compressor to either the entire file or every block of the file. Both methods have limitations on either inefficient data accesses or degraded compression ratio. We introduce a new concept called virtual chunks, which enable efficient random accesses to the compressed files while retaining high compression ratio.

The key idea [16] is to append additional references to the compressed files so that a decompression request could start at an arbitrary position. Current system prototype [15] assumes the references are equidistant, and experiments show that virtual chunks improve random accesses by $2\times$ speedup.

8.5.6 Space-Efficient Data Reliability

The reliability of distributed filesystems is typically achieved through data replication. That is, a primary copy serves most requests, and there are a number of backup copies (replicas) that would become the primary copy upon a failure.

One concern with the conventional approach is its space efficiency; for example, two replicas imply poor 33 % space efficiency. On the other hand, erasure coding has been proposed to improve the space efficiency; unfortunately it is criticized on its computation overhead. We integrated GPU-accelerated erasure coding to FusionFS and report the performance in [11]. Results showed that erasure coding could improve FusionFS performance by up to 1.82×.

8.5.7 Distributed Data Provenance

The traditional approach to track application's provenance is through a centralized database. To address this performance bottleneck on large-scale systems, in [31] we propose a lightweight database on every compute node. This allows every participating node to maintain its own data provenance, and results in highly scalable aggregate I/O throughput. Admittedly, an obvious drawback of this approach is on the interaction among multiple physical databases: the provenance overhead becomes unacceptable when there is heavy traffic among peers.

To address the above drawback, we explore the feasibility of tracking data provenance in a completely distributed manner in [12]. We replace the database component by a graph-like hashtable data structure, and integrate it into the FusionFS filesystem. With a hybrid granularity of provenance information on both block- and file-level, FusionFS achieves over 86 % system efficiency on 1,024 nodes. A query interface is also implemented with small performance overhead as low as 5.4 % on 1,024 nodes.

8.5.8 Parallel Serialization

We have explored how to leverage modern computing systems' multi-cores to improve the serialization and deserialization speed of large objects. [17] Rather than proposing new serialization algorithms, we tackle the problem from a system's perspective. Specifically, we propose to leverage multiple CPU cores to split a large object into smaller sub-objects, so to be serialized in parallel. While data parallelism is not a new idea in general, it has never been applied to data serialization and poses new problems. For instance, serializing multiple chunks of a large object incurs additional overhead such as metadata maintenance, thread and process synchronization, resource contention. In addition, the granularity (i.e., the number of sub-objects) is a machine-dependent choice: the optimal number of concurrent processes and threads might not align with the available CPU cores.

In order to overcome these challenges and better understand whether the proposed approach could improve the performance of data serialization of large objects, we provide detailed analysis on the system design, for example, how to determine the sub-object's granularity for optimal performance and how to ensure that the performance gain is larger than the cost. To demonstrate the effectiveness of our proposed approach, we implemented a system prototype called parallel protocol buffers (PPB) by extending a widely used open-source serialization utility (Google's Protocol Buffers [32]). We have evaluated PPB on a variety of test beds: a conventional Linux server, the Amazon EC2 cloud, and an IBM Blue Gene/P supercomputer. Experimental results confirm that the proposed approach could significantly accelerate the serialization process. In particular, PPB could accelerate the metadata interchange 3.6× faster for FusionFS.

8.6 Related Work

Conventional storage in HPC systems for scientific applications are mainly remote to compute resources. Popular systems include GPFS [23], Lustre [24], PVFS [33]. All these systems are typically deployed on a distinct cluster from compute nodes. The architecture with separated compute- and storage-resources, which was designed decades ago, has shown its limitation for modern applications that are becoming increasingly data-intensive [7].

Cloud computing, on the other hand, is built on the commodity hardware where local storage is typically available for virtual computing machines. The de facto node-local file system (Google File System [3], HDFS [1]), however, can be hardly leveraged by scientific applications out of the box due to the concerns on small file accesses, POSIX interface, and so forth. Another category of storage in the cloud is similar to the conventional HPC solution—a remote shared storage such as Amazon S3. A POSIX-compliant file system built on S3 is also available named S3FS [4]. Unfortunately its throughput performance usually becomes a bottleneck of the applications and thus limits its use in practice.

Fortunately, researchers have made a significant amount of effort [34, 35] to bridge the gap between two extremes (HPC and cloud computing) of storage paradigms, particularly in terms of scalability [36]. We observe more and more node-local and POSIX-compliant storage systems (Ceph [6], FusionFS [7]) being tested on the cloud.

We will briefly review the unique features provided by FusionFS as follows.

8.6.1 Filesystem Caching

To the best of our knowledge, HyCache is the first user-level POSIX-compliant hybrid caching for distributed file systems. Some of our previous work [30, 37] proposed data caching to accelerate applications by modifying the applications and/or their workflow, rather than at the filesystem level. Other existing work requires modifying OS kernel, or lacks of a systematic caching mechanism for manipulating files across multiple storage devices, or does not support the POSIX interface. Any of the these concerns would limit the system's applicability to end users. We will give a brief review of previous studies on hybrid storage systems.

Some recent work reported the performance comparison between SSD and HDD in more perspectives [38, 39]. Hystor [40] aims to optimize of the hybrid storage of SSDs and HDDs. However it requires to modify the kernel which might cause some issues. A more general multitiering scheme was proposed in [41] which helps decide the needed numbers of SSD/HDDs and manage the data shift between SSDs and HDDs by adding a "pseudo device driver," again, in the kernel. iTransformer [42] considers the SSD as a traditional transient cache in which case data needs to be written to the spinning hard disk at some point once the data is modified in the SSD. iBridge [43] leverages SSD to serve request fragments and bridge the performance gap between serving fragments and serving large sub-requests. HPDA [44] offers a mechanism to plug SSDs into RAID in order to improve the reliability of the disk array. SSD was also proposed to be integrated to the RAM level which makes SSD as the primary holder of virtual memory [45]. NVMalloc [46] provides a library to explicitly allow users to allocate virtual memory on SSD. Also for extending virtual memory with Storage Class Memory (SCM), SCMFS [47] concentrates more on the management of a single SCM device. FAST [48] proposed a caching system to pre-fetch data in order to quicken the application launch. In [49] SSD is considered as a read-only buffer and migrate those random-writes to HDD.

A thorough review of classical caching algorithms on large-scale data-intensive applications is recently reported in [50]. HyCache+ is different from the classical cooperative caching [51] in that HyCache+ assumes persistent underlying storage and manipulates data at the file level. As an example of distributed caching for distributed file systems, Blue Whale Cooperative Caching (BWCC) [52] is a read-only caching system for cluster file systems. In contrast, HyCache+ is a POSIX-compliant I/O storage middleware that transparently interacts with the underlying parallel file systems. Even though the focus of this chapter lies on the 2-layer hierarchy of a local cache (e.g., SSD) and a remote parallel file system (e.g., GPFS [23]), the approach presented in HyCache+ is applicable to multitier caching architecture as well. Multilevel caching gains much research interest, especially in the emerging age of cloud computing where the hierarchy of (distributed) storage is being redefined with more layers. For example, Hint-K [53] caching is proposed to keep track of the last K steps across all the cache levels, which generalizes the conventional LRU-K algorithm concerned only on the single-level information.

There are extensive studies on leveraging data locality for effective caching. Block Locality Caching (BLC) [54] captures the backup and always uses the latest locality information to achieve better performance for data deduplication systems. The File Access corRelation Mining and Evaluation Reference model (FARMER) [55] optimizes the large-scale file system by correlating access patterns and semantic attributes. In contrast, HyCache+ achieves data locality with a unique mix of two principles: (1) write is always local, and (2) read locality depends on the novel 2LS mechanism which schedules jobs in a deterministic manner followed by a local heuristic replacement policy.

While HyCache+ presents a pure software solution for distributed cache, some orthogonal work focuses on improving caching from the hardware perspective. In [56], a hardware design is proposed with low overhead to support effective shared caches in multicore processors. For shared last-level caches, COOP [57] is proposed to only use one bit per cache line for re-reference prediction and optimize both locality and utilization. The REDCAP project [58] aims to logically enlarge the disk cache using a small portion of main memory, so that the read time could be reduced. For Solid-State Drive (SSD), a new algorithm called lazy adaptive replacement cache [59] is proposed to improve the cache hit and prolong the SSD lifetime.

Power-efficient caching has drawn a lot of research interests. It is worth mentioning that HyCache+ aims to better meet the need of high I/O performance for HPC systems, and power consumption is not the major consideration at this point. Nevertheless, it should be noted that power consumption is indeed one of the toughest challenges to be overcome in future systems. One of the earliest work [60] tried to minimize the energy consumption by predicting the access mode and allowing cache accesses to switch between the prediction and the access modes. Recently, a new caching algorithm [61] was proposed to save up to 27 % energy and reduce the memory temperature up to 5.45 °C with negligible performance degradation. EEVFS [62] provides energy efficiency at the file system level with an energy-aware data layout and the prediction on disk idleness.

While HyCache+ is architected for large-scale HPC systems, caching has been extensively studied in different subjects and fields. In cloud storage, Update-batched Delayed Synchronization (UDS) [63] reduces the synchronization cost by buffering the frequent and short updates from the client and synchronizing with the underlying infrastructure in a batch fashion. For continuous data (e.g., online video), a new algorithm called Least Waiting Probability (LWP) [64] is proposed to optimize the newly defined metric called user waiting rate. In geoinformatics, the method proposed in [65] considers both global and local temporal-spatial changes to achieve high cache hit rate and short response time.

The job scheduler proposed in this work takes a greedy strategy to achieve the optimal solution for the HyCache+ architecture. A more general, and more difficult, scheduling problem could be solved in a similar heuristic approach [66, 67]. For an even more general combinatorial optimization problem in a network, both precise and bound-proved low-degree polynomial approximation algorithms were reported in [68, 69]. Some incremental approaches [70–72] were proposed to efficiently retain the strong connectivity of a network and solve the satisfiability problem with constraints.

8.6.2 Filesystem Compression

While the storage system could be better designed to handle more data, an orthogonal approach is to address the I/O bottleneck by squeezing the data with compression techniques. One example where data compression gets particularly popular is checkpointing, an extremely expensive I/O operation in HPC systems. In [73], it showed that data compression had the potential to significantly reduce the checkpointing file sizes. If multiple applications run concurrently, a data-aware compression scheme [74] was proposed to improve the overall checkpointing efficiency. Recent study [75] shows that combining failure detection and proactive checkpointing could improve 30 % efficiency compared to classical periodical checkpointing. Thus data compression has the potential to be combined with failure detection and proactive checkpointing to further improve the system efficiency. As another example, data compression was also used in reducing the MPI trace size, as shown in [76]. A small MPI trace enables an efficient replay and analysis of the communication patterns in large-scale machines.

It should be noted that a compression method does not necessarily need to restore the absolutely original data. In general, compression algorithms could be categorized into to two groups: lossy algorithms and lossless algorithms. A lossy algorithm might lose some (normally a small) percentage of accuracy, while a lossless one has to ensure the 100 % accuracy. In scientific computing, studies [77, 78] show that lossy compression could be acceptable, or even quite effective, under certain circumstances. In fact, lossy compression is also popular in other fields, e.g. the most widely compatible lossy audio and video format MPEG-1 [79]. This section presents virtual chunks mostly by going through a delta-compression example based on XOR, which is a lossless compression. It does not imply that virtual chunks cannot be used in a lossy compression. Virtual chunk is not a specific compression algorithm, but a system mechanism that is applicable to any splittable compression, no matter if it is lossy or lossless.

Some frameworks are proposed as middleware to allow applications call high-level I/O libraries for data compression and decompression, e.g., [80–82]. None of these techniques take consideration of the overhead involved in decompression by assuming the chunk allocated to each node would be requested as an entirety. In contrast, virtual chunks provide a mechanism to apply flexible compression and decompression.

There is much previous work to study the file system support for data compression. Integrating compression to log-structured file systems was proposed decades ago [83], which suggested a hardware compression chip to accelerate the compressing and decompressing. Later, XDFS [84] described the systematic design and implementation for supporting data compression in file systems with BerkeleyDB [85].

MRAMFS [86] was a prototype file system to support data compression to leverage the limited space of nonvolatile RAM. In contrast, virtual trunks represent a general technique applicable to existing algorithms and systems.

Data deduplication is a general inter-chunk compression technique that only stores a single copy of the duplicate chunks (or blocks). For example, LBFS [87] was a networked file system that exploited the similarities between files (or versions of files) so that chunks of files could be retrieved in the client's cache rather than transferring from the server. CZIP [88] was a compression scheme on content-based naming that eliminated redundant chunks and compressed the remaining (i.e., unique) chunks by applying existing compression algorithms. Recently, the metadata for the deduplication (i.e., file recipe) was also slated for compression to further save the storage space [89]. While deduplication focuses on inter-chunk compressing, virtual chunk focuses on the I/O improvement within the chunk.

Index has been introduced to data compression to improve the compressing and query speed, e.g., [90, 91]. The advantage of indexing is highly dependent on the chunk size: large chunks are preferred to achieve high compression ratios in order to amortize the indexing overhead. Large chunks, however, would cause potential decompression overhead as explained earlier in this chapter. Virtual chunk overcomes the large-chunk issue by logically splitting the large chunks with fine-grained partitions while still keeping the physical coherence.

8.6.3 GPU Acceleration

Recent GPU technology has drawn much research interest of applying these many-cores for data parallelism. For example, GPUs are proposed to parallelize the XML processing [92]. In high-performance computing, a GPU-aware MPI was proposed to enable the inter-GPU communication without changing the original MPI interface [93]. Nevertheless, GPUs do not necessarily provide superior performance; GPUs might suffer from factors such as small shared memory and weak single-thread performance as shown in [94]. Another potential drawback of GPUs lies in the dynamic instrumentation that introduces runtime overhead. Yet, recent study [95] shows that the overhead could be alleviated by information flow analysis and symbolic execution. In this paper, we leverage GPUs in key-value stores—a new domain for many-cores.

8.6.4 Filesystem Provenance

As distributed systems become more ubiquitous and complex, there is a growing emphasis on the need for tracking provenance metadata along with file system metadata. A thorough review is presented in [96]. Many Grid systems like Chimera [97] and the Provenance Aware Service Oriented Architecture (PASOA) [98] provide provenance tracking mechanisms for various applications. However these systems are very domain-specific and do not capture provenance at the filesystem level. The Distributed Provenance Aware Storage System (DPASS) tracks the provenance of files in a distributed file system by intercepting filesystem operations and sending this information via a netlink socket to user-level daemon that collects provenance in a database server [99]. The provenance is however, collected in a centralized fashion, which is a poor design choice for distributed file systems meant for extreme scales. Similarly in efficient retrieval of files, provenance is collected centrally [100].

PASS describes global naming, indexing, and querying in the context of sensor data [101], which is a challenging problem also from system's perspective [36]. PA-NFS [102] enhances NFS to record provenance in local area networks but does not consider distributed naming explicitly. SPADE [103] addresses the issue using storage identifiers for provenance vertices that are unique to a host and requiring distributed provenance queries to disambiguate vertices by referring to them by the host on which the vertex was generated as well as the identifier local to that host.

Several storage systems have been considered for storing provenance. ExSPAN [104] extends traditional relational models for storing and querying provenance metadata. SPADE supports both graph and relational database storage and querying. PASS has explored the use of clouds [101]. Provbase uses Hbase to store and query scientific workflow provenance [105]. Further compressing provenance [104], indexing [106] and optimization techniques [107] have also been considered. However, none of these systems have been tested for exascale architectures. To give adequate merit to the previous designs, we have integrated FusionFS with SPADE as well as considered FusionFS's internal storage system for storing audited provenance.

8.6.5 Data Serialization

Many serialization frameworks are developed to support transporting data over distributed systems. XML [108] represents a set of rules to encoding documents or text-based files. Another format, namely JSON [109], is treated as a lightweight alternative to XML in web services and mobile devices as well. While XML and JSON are the most widely used data serialization format for text-based files, binary format is also gaining its popularity. A binary version of JSON is available called BSON [110]. Two other famous binary data serialization frameworks are Google's Protocol Buffers [32] and Apache Thrift [111]. Both frameworks are designed to support lightweight and fast data serialization and deserialization, which could substantially improve the data communication in distributed systems. The key difference between Thrift and Protocol Buffers is that the former has the built-in support for RPC.

Many other serialization utilities are available at the present. Avro [112] is used by Hadoop for serialization. Internally, it uses JSON [109] to represent data types and protocols and improves the performance of the Java-based framework. Etch [113] supports more flexible data models (for example, trees), but it is slower and generates larger files. BERT [114] supports data format compatible with Erlang's binary serialization format. Message Pack [115] allows both binary data and non UTF-8 encoded strings. Hessian [116] is a binary web service protocol that is 2× faster than the Java serialization with significantly smaller compressed data size. ICE [117] is a middleware platform that supports object-oriented RPC and data exchange. CBOR [118] is designed to support extremely small message size.

None of the aforementioned systems, however, support data parallelism. Thus they suffer the low efficiency problem when multiple CPU cores are available particularly when the data is large in size. PPB, on the other hand, takes advantage of the idle cores and leverage them for parallelizing the compute-intensive process of data serialization

Many frameworks are recently developed for parallel data processing. MapReduce [119] is a programming paradigm and framework that allows users to process terabytes of data over massive-scale architecture in a matter of seconds. Apache Hadoop [120] is one of the most popular open-source implementations of MapReduce framework. Apache Spark [121] is an execution engine which supports more types of workload than Hadoop and MapReduce.

Several parallel programming models and paradigms have been existing for decades. Message Passing Interface (MPI) a standard for messages exchange between processes. It greatly reduces the burden from developers who used to consider detailed protocols in multiprocessing programs and tries to optimize the performance in many scenarios. The major implementation includes MPICH [122] and Open MPI [123]. OpenMP [124] is a set of compiler directives and runtime library routines that enable the parallelization of code's execution over shared memory multiprocessor computers. It supports different platforms and processor architectures, programming languages, and operating systems. Posix Threads (Pthread) is defined as a set of C programming types and function calls. It provides standardized programming interface to create and manipulate threads, which allow developers to take full advantage of the capabilities of threads. Microsoft's Parallel Patterns Library (PPL) [125] gives an imperative programming model that introduces parallelism to applications and improves scalability.

Numerous efforts have been devoted to utilizing or improving data parallelism in cluster and cloud computing environment. Jeon et al. [126, 127] proposed adaptive parallelization and prediction approaches for search engine query. Lee et al. [128] presented how to reduce data migration cost and improve I/O performance by incorporating parallel data compression on the client side. Klasky et al. [129] proposed a parallel data-streaming approach with multi-threads to migrate terabytes of scientific data across distributed supercomputer centers. Some work [130–133] proposed data-parallel architectures and systems for large-scale distributed computing. In [134, 135], authors exploited the data parallelism in a program by dynamically executing sets of serialization codes concurrently.

Unfortunately, little study exists on data parallelism for data serialization, mainly because large messages are usually not the dominating cost by convention. PPB [17] for the first time identifies that large message is a challenging problem from our observations on real-world applications at Google. We hope our PPB experience could provide the community insights for designing the next-generation data serialization tools.

8.7 Conclusion

This chapter envisions the characteristics of future cloud storage systems for scientific applications that used to be running on HPC systems. Based on the literature and our own FusionFS experience, we believe the key designs of future storage system comprise the fusion of compute and storage resources as well as completely distributed data manipulation (both metadata and data), namely (1) distributed metadata accesses, (2) optimized data write, and (3) localized file read.

To make matters more concrete, we then detail the design and implementation of FusionFS, whose uniqueness lies in its highly scalable metadata and data throughput. We also discuss its integral features such as cooperative caching, efficient accesses to compressed data, space-efficient data reliability, distributed data provenance, and parallel data serialization. All the aforementioned features enable FusionFS to nicely bridge the storage gap between scientific big data applications and cloud computing.

References

1. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: Proceedings of IEEE Symposium on Mass Storage Systems and Technologies (2010)
2. Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J., Ludwig, T.: Small-file access in parallel file systems. In: Proceedings of IEEE International Symposium on Parallel and Distributed Processing (2009)
3. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: ACM Symposium on Operating Systems Principles (2003)
4. S3FS: https://code.google.com/p/s3fs/. Accessed 6 March 2015
5. FUSE: http://fuse.sourceforge.net. Accessed 5 Sept 2014
6. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C.: Ceph: a scalable, high-performance distributed file system. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation (2006)
7. Zhao, D., Zhang, Z., Zhou, X., Li, T., Wang, K., Kimpe, D., Carns, P., Ross, R., Raicu, I.: FusionFS: Toward supporting data-intensive scientific applications on extreme-scale distributed systems. In: Proceedings of IEEE International Conference on Big Data, pp. 61–70 (2014)
8. Zhao, D., Liu, N., Kimpe, D., Ross, R., Sun, X.H., Raicu, I.: Towards exploring data-intensive scientific applications at extreme scales through systems and simulations. IEEE Trans. Parallel Distrib. Syst. 1–14 (2015). doi:10.1109/TPDS.2015.2456896
9. Weil, S.A., Brandt, S.A., Miller, E.L., Maltzahn, C.: Crush: controlled, scalable, decentralized placement of replicated data. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (2006)

10. Zhao, D., Raicu, I.: Distributed file systems for exascale computing. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC '12), doctoral showcase (2012)
11. Zhao, D., Burlingame, K., Debains, C., Alvarez-Tabio, P., Raicu, I.: Towards high-performance and cost-effective distributed storage systems with information dispersal algorithms. In: IEEE International Conference on Cluster Computing (2013)
12. Zhao, D., Shou, C., Malik, T., Raicu, I.: Distributed data provenance for large-scale data-intensive computing. In: IEEE International Conference on Cluster Computing (2013)
13. Zhao, D., Qiao, K., Raicu, I.: Hycache+: towards scalable high-performance caching middleware for parallel file systems. In: Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 267–276 (2014)
14. Zhao, D., Raicu, I.: HyCache: a user-level caching middleware for distributed file systems. In: Proceedings of IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (2013)
15. Zhao, D., Yin, J., Qiao, K., Raicu, I.: Virtual chunks: on supporting random accesses to scientific data in compressible storage systems. In: Proceedings of IEEE International Conference on Big Data, pp. 231–240 (2014)
16. Zhao, D., Yin, J., Raicu, I.: Improving the i/o throughput for data-intensive scientific applications with efficient compression mechanisms. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC '13), poster session (2013)
17. Zhao, D., Qiao, K., Zhou, Z., Li, T., Zhou, X., Wang, K., Raicu, I.: Exploiting multi-cores for efficient interchange of large messages in distributed systems. Concurrency Comput.: Pract. Experience 2015 (accepted)
18. Kodiak: https://www.nmc-probe.org/wiki/Machines:Kodiak. Accessed 5 Sept 2014
19. Amazon EC2: http://aws.amazon.com/ec2. Accessed 6 March 2015
20. Welch, B., Noer, G.: Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions. In: IEEE 29th Symposium on Mass Storage Systems and Technologies (2013)
21. Nagle, D., Serenyi, D., Matthews, A.: The Panasas activescale storage cluster: delivering scalable high bandwidth storage. In: Proceedings of ACM/IEEE Conference on Supercomputing (2004)
22. Zhao, D., Zhang, D., Wang, K., Raicu, I.: Exploring reliability of exascale systems through simulations. In: Proceedings of the 21st ACM/SCS High Performance Computing Symposium (HPC) (2013)
23. Schmuck, F., Haskin, R.: GPFS: a shared-disk file system for large computing clusters. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies (2002)
24. Schwan, P.: Lustre: building a file system for 1,000-node clusters. In: Proceedings of the Linux Symposium (2003)
25. Wu, H., Ren, S., Garzoglio, G., Timm, S., Bernabeu, G., Chadwick, K., Noh, S.-Y.: A reference model for virtual machine launching overhead. IEEE Trans. Cloud Comput. (pp. 99), 1–1 (2014)
26. Wu, H., Ren, S., Garzoglio, G., Timm, S., Bernabeu, G., Noh, S.-Y.: Modeling the virtual machine launching overhead under fermicloud. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), May 2014
27. Li, T., Zhou, X., Brandstatter, K., Zhao, D., Wang, K., Rajendran, A., Zhang, Z., Raicu, I.: ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table. In: Proceedings of IEEE International Symposium on Parallel and Distributed Processing (2013)
28. Li, T., Ma, C., Li, J., Zhou, X., Wang, K., Zhao, D., Raicu, I.: Graph/z: a key-value store based scalable graph processing system. In: IEEE International Conference on Cluster Computing (2015)
29. Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., Wilde, M.: Swift: Fast, reliable, loosely coupled parallel computation. In: IEEE Congress on Services (2007)
30. Raicu, I., Foster, I.T., Zhao, Y., Little, P., Moretti, C.M., Chaudhary, A., Thain, D.: The quest for scalable support of data-intensive workloads in distributed systems. In: Proceedings of ACM International Symposium on High Performance Distributed Computing (2009)

31. Shou, C., Zhao, D., Malik, T., Raicu, I.: Towards a provenance-aware distributed filesystem. In: 5th Workshop on the Theory and Practice of Provenance (TaPP) (2013)
32. Protocol Buffers: http://code.google.com/p/protobuf/. Accessed 5 Sept 2014
33. Carns, P.H., Ligon, W.B., Ross, R.B., Thakur, R.: PVFS: a parallel file system for linux clusters. In: Proceedings of the 4th Annual Linux Showcase and Conference (2000)
34. Li, T., Zhou, X., Wang, K., Zhao, D., Sadooghi, I., Zhang, Z., Raicu, I.: A convergence of key-value storage systems from clouds to supercomputer. Concurrency Comput.: Pract. Experience (2016)
35. Zhao, D., Yang, X., Sadooghi, I., Garzoglio, G., Timm, S., Raicu, I.: High-performance storage support for scientific applications on the cloud. In: Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud) (2015)
36. Li, T., Keahey, K., Wang, K., Zhao, D., Raicu, I.: A dynamically scalable cloud data infrastructure for sensor networks. In: Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud) (2015)
37. Raicu, I., Zhao, Y., Foster, I.T., Szalay, A.: Accelerating large-scale data exploration through data diffusion. In: Proceedings of the 2008 International Workshop on Data-aware Distributed Computing (2008)
38. Li, S., Huang, H.H.: Black-box performance modeling for solid-state drives. In: 2010 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS) (2010)
39. Rizvi, S., Chung, T.-S.: Flash SSD vs HDD: High performance oriented modern embedded and multimedia storage systems. In: 2nd International Conference on Computer Engineering and Technology (ICCET) (2010)
40. Chen, F., Koufaty, D.A., Zhang, X.: Hystor: making the best use of solid state drives in high performance storage systems. In: Proceedings of the International Conference on Supercomputing (2011)
41. Guerra, J., Pucha, H., Glider, J., Belluomini, W., Rangaswami, R.: Cost effective storage using extent based dynamic tiering. In: Proceedings of the 9th USENIX Conference on File and Stroage Technologies (2011)
42. Zhang, X., Davis, K., Jiang, S.: iTransformer: using SSD to improve disk scheduling for high-performance I/O. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (2012)
43. Zhang, X., Ke, L., Davis, K., Jiang, S.: iBridge: improving unaligned parallel file access with solid-state drives. In: Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium (2013)
44. Mao, B., Jiang, H., Feng, D., Wu, S., Chen, J., Zeng, L., Tian, L.: HPDA: a hybrid parity-based disk array for enhanced performance and reliability. In: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS) (2010)
45. Badam, A., Pai, V.S.: SSDAlloc: hybrid SSD/RAM memory management made easy. In: Proceedings of the 8th USENIX Conference on Networked systems design and implementation (2011)
46. Wang, C., Vazhkudai, S.S., Ma, X., Meng, F., Kim, Y., Engelmann, C.: Nvmalloc: exposing an aggregate ssd store as a memory partition in extreme-scale machines. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (2012)
47. Wu, X., Narasimha Reddy, A.L.: SCMFS: a file system for storage class memory. In: Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (2011)
48. Joo, Y., Ryu, J., Park, S., Shin, K.G.: FAST: quick application launch on solid-state drives. In: Proceedings of the 9th USENIX Conference on File and Stroage Technologies (2011)
49. Yang, Q., Ren, J.: I-CASH: intelligently coupled array of SSD and HDD. In: Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture (2011)
50. Fares, R., Romoser, B., Zong, Z., Nijim, M., Qin, X.: Performance evaluation of traditional caching policies on a large system with petabytes of data. In: 2012 IEEE 7th International Conference on Networking, Architecture and Storage (NAS) (2012)

51. Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Comput. Surv. **35**(4) (2003)
52. Shi, L., Liu, Z., Xu, L.: Bwcc: a fs-cache based cooperative caching system for network storage system. In: Proceedings of the 2012 IEEE International Conference on Cluster Computing (2012)
53. Wu, C., Xubin, H., Qiang, C., Changsheng, X., Shenggang, W.: Hint-k: an efficient multi-level cache using k-step hints. IEEE Trans. Parallel Distrib. Syst. **99** (2013)
54. Meister, D., Kaiser, J., Brinkmann, A.: Block locality caching for data deduplication. In: Proceedings of the 6th International Systems and Storage Conference (2013)
55. Xia, P., Feng, D., Jiang, H., Tian, L., Wang, F.: Farmer: a novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing (2008)
56. Lin, J., Lu, Q., Ding, X., Zhang, Z., Zhang, X., Sadayappan, P.: Enabling software management for multicore caches with a lightweight hardware support. In: Proceedings of the 2009 ACM/IEEE Conference on Supercomputing (2009)
57. Zhan, D., Jiang, H., Seth, S.C.: Locality & utility co-optimization for practical capacity management of shared last level caches. In: Proceedings of the 26th ACM International Conference on Supercomputing (2012)
58. Gonzalez-Ferez, P., Piernas, J., Cortes, T.: The ram enhanced disk cache project (redcap). In: Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies (2007)
59. Huang, S., Wei, Q., Chen, J., Chen, C., Feng, D.: Improving flash-based disk cache with lazy adaptive replacement. In: 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST) (2013)
60. Zhu, Z., Zhang, X.: Access-mode predictions for low-power cache design. IEEE Micro **22**(2) (2002)
61. Yue, J., Zhu, Y., Cai, Z., Lin, L.: Energy and thermal aware buffer cache replacement algorithm. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (2010)
62. Manzanares, A., Ruan, X., Yin, S., Xie, J., Ding, Z., Tian, Y., Majors, J., Qin, X.: Energy efficient prefetching with buffer disks for cluster file systems. In: Proceedings of the 2010 39th International Conference on Parallel Processing (2010)
63. Li, Z., Wilson, C., Jiang, Z., Liu, Y., Zhao, B., Jin, C., Zhang, Z.L., Dai, Y.: Efficient batched synchronization in dropbox-like cloud storage services. In: Proceedings of the 14th International Middleware Conference (2013)
64. Xu, Y., Xing, C., Zhou, L.: A cache replacement algorithm in hierarchical storage of continuous media object. In: Advances in Web-Age Information Management: 5th International Conference (2004)
65. Li, R., Guo, R., Xu, Z., Feng, W.: A prefetching model based on access popularity for geospatial data in a cluster-based caching system. Int. J. Geogr. Inf. Sci. **26**(10) (2012)
66. Qiao, K., Tao, F., Zhang, L., Li, Z.: A ga maintained by binary heap and transitive reduction for addressing psp. In: 2010 International Conference on Intelligent Computing and Integrated Systems (ICISS) (2010)
67. Tao, F., Qiao, K., Zhang, L., Li, Z., Nee, A.: GA-BHTR: an improved genetic algorithm for partner selection in virtual manufacturing. Int. J. Prod. Res. **50**(8) (2012)
68. Calinescu, G., Qiao, K.: Asymmetric topology control: exact solutions and fast approximations. In: IEEE International Conference on Computer Communications (INFOCOM '12) (2012)
69. Calinescu, G., Kapoor, S., Qiao, K., Shin, J.: Stochastic strategic routing reduces attack effects. In: Global Telecommunications Conference (GLOBECOM 2011), 2011. IEEE (2011)
70. Zhao, D,, Yang, L.: Incremental isometric embedding of high-dimensional data using connected neighborhood graphs. IEEE Trans. Pattern Anal. Mach. Intell. **31**(1), 86–98 (2009)
71. Lohfert, R., Lu, J., Zhao, D.; Solving sql constraints by incremental translation to sat. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (2008)

72. Zhao, D., Yang, L.: Incremental construction of neighborhood graphs for nonlinear dimensionality reduction. In: Proceedings of 18th International Conference on Pattern Recognition, vol. 3, pp. 177–180 (2006)
73. Ferreira, K.B., Riesen, R., Arnold, D., Ibtesham, D., Brightwell, R.: The viability of using compression to decrease message log sizes. In: Proceedings of International Conference on Parallel Processing Workshops (2013)
74. Zerin Islam, T., Mohror, K., Bagchi, S., Moody, A., de Supinski, B.R., Eigenmann, R.: McrEngine: a scalable checkpointing system using data-aware aggregation and compression. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC) (2012)
75. Slim Bouguerra, M., Gainaru, A., Gomez, L.B., Cappello, F., Matsuoka, S., Maruyam, N.: Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing. In: IEEE International Symposium on Parallel Distributed Processing (2013)
76. Noeth, M., Marathe, J., Mueller, F., Schulz, M., de Supinski, B.: Scalable compression and replay of communication traces in massively parallel environments. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC) (2006)
77. Laney, D., Langer, S., Weber, C., Lindstrom, P., Wegener, A.: Assessing the effects of data compression in simulations using physically motivated metrics. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (2013)
78. Lakshminarasimhan, S., Jenkins, J., Arkatkar, I., Gong, Z., Kolla, H., Ku, S.-H., Ethier, S., Chen, J., Chang, C.S., Klasky, S., Latham, R., Ross, R., Samatova, N.F.: ISABELA-QA: query-driven analytics with ISABELA-compressed extreme-scale scientific data. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11) (2011)
79. MPEG-1: http://en.wikipedia.org/wiki/MPEG-1. Accessed 5 Sept 2014
80. Bicer, T., Yin, J., Chiu, D., Agrawal, G., Schuchardt, K.: Integrating online compression to accelerate large-scale data analytics applications. In: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS) (2013)
81. Schendel, E.R., Pendse, S.V., Jenkins, J., Boyuka, D.A., II, Gong, Z., Lakshminarasimhan, S., Liu, Q., Kolla, H., Chen, J., Klasky, S.,Ross, R., Samatova, N.F.: Isobar hybrid compression-i/o interleaving for large-scale parallel i/o optimization, In: Proceedings of International Symposium on High-Performance Parallel and Distributed Computing (2012)
82. Jenkins, J., Schendel, E.R., Lakshminarasimhan, S., Boyuka, D.S., II, Rogers, T., Ethier, S., Ross, R., Klasky, S., Samatova, N.F.: Byte-precision level of detail processing for variable precision analytics. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC) (2012)
83. Burrows, M., Jerian, C., Lampson, B., Mann, T.: On-line data compression in a log-structured file system. In: Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (1992)
84. Joshua, P.: MacDonald. File system support for delta compression. Technical report, University of California, Berkley (2000)
85. Olson, M.A., Bostic, K., Seltzer M.: db. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference (1999)
86. Edel, N.K., Tuteja, D., Miller, E.L., Brandt S.A.: Mramfs: a compressing file system for non-volatile ram. In: Proceedings of the the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS) (2004)
87. Muthitacharoen, A., Chen, B., Mazières, D.: A low-bandwidth network file system. In: Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP) (2001)
88. Park, K.S., Ihm, S., Bowman, M., Pai, V.S.: Supporting practical content-addressable caching with czip compression. In: 2007 USENIX Annual Technical Conference (2007)

89. Meister, D., Brinkmann, A., Süß, T.: File recipe compression in data deduplication systems. In: Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST) (2013)
90. Lakshminarasimhan, S., Boyuka, D.A., Pendse, S.V., Zou, X., Jenkins, J., Vishwanath, V., Papka, M.E., Samatova, N.F.: Scalable in situ scientific data encoding for analytical query processing. In: Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing (HPDC) (2013)
91. Gong, Z., Lakshminarasimhan, S., Jenkins, J., Kolla, H., Ethier, S., Chen, J., Ross, R., Klasky, S., Samatova, N.F.: Multi-level layout optimization for efficient spatio-temporal queries on isabela-compressed data. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS) (2012)
92. Shnaiderman, L., Shmueli, O.: A parallel twig join algorithm for XML processing using a GPGPU. In: International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures (2012)
93. Wang, H., Potluri, S., Bureddy, D., Rosales, C., Panda, D.K.: Gpu-aware mpi on rdma-enabled clusters: design, implementation and evaluation. IEEE Trans. Parallel Distrib. Syst. **25**(10) (2014)
94. Bordawekar, R., Bondhugula, U., Rao. R.: Believe it or not!: mult-core cpus can match gpu performance for a flop-intensive application! In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT '10, (2010)
95. Farooqui, N., Schwan, K., Yalamanchili, S.: Efficient instrumentation of gpgpu applications using information flow analysis and symbolic execution. In: Proceedings of Workshop on General Purpose Processing Using GPUs, GPGPU-7 (2014)
96. Muniswamy-Reddy, K.-K.: Foundations for provenance-aware systems (2010)
97. Foster, I.T., Vckler, J.S., Wilde, M., Zhao, Y.: The virtual data grid: a new model and architecture for data-intensive collaboration. In: CIDR'03 (2003)
98. Provenance aware service oriented architecture. http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/WebHome. Accessed 6 July 2015
99. Parker-Wood, A., Long, D.D.E., Miller, E.L., Seltzer, M., Tunkelang, D.: Making sense of file systems through provenance and rich metadata. Technical Report UCSC-SSRC-12-01, University of California, Santa Cruz, March 2012
100. Muniswamy-Reddy, K.-K., Holland, D.A., Braun, U., Seltzer, M.: Provenance-aware storage systems. In: Proceedings of the annual conference on USENIX '06 Annual Technical Conference (2006)
101. Muniswamy-Reddy, K.-K., Macko, P., Seltzer, M.: Making a cloud provenance-aware. In: 1st Workshop on the Theory and Practice of Provenance (2009)
102. Muniswamy-Reddy, K.-K., Braun, U., Holland, D.A., Macko, P., Maclean, D., Margo, D., Seltzer, M., Smogor, R.: Layering in provenance systems. In: Proceedings of the 2009 USENIX Annual Technical Conference (2009)
103. Gehani, A., Tariq, D.: SPADE: support for provenance auditing in distributed environments. In: Proceedings of the 13th International Middleware Conference (2012)
104. Zhou, W., Sherr, M., Tao, T., Li, X., Thau Loo, B., Mao, Y.: Efficient querying and maintenance of network provenance at internet-scale. In: Proceedings of the 2010 International Conference on Management of Data, pp. 615–626 (2010)
105. Abraham, J., Brazier, P., Chebotko, A., Navarro, J., Piazza, A.: Distributed storage and querying techniques for a semantic web of scientific workflow provenance. In: 2010 IEEE International Conference on Services Computing (SCC), pp. 178–185. IEEE (2010)
106. Malik, T., Gehani, A., Tariq, D., Zaffar, F.: Sketching distributed data provenance. In: Data Provenance and Data Management in eScience, pp. 85–107 (2013)
107. Heinis, T., Alonso, G.: Efficient lineage tracking for scientific workflows. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1007–1018 (2008)
108. Extensible Markup Language (XML): http://www.w3.org/xml/. Accessed 13 Dec 2014
109. JSON: http://www.json.org/. Accessed 8 Dec 2014

110. Binary JSON: http://bsonspec.org/. Accessed 13 Dec 2014
111. Apache Thrift: https://thrift.apache.org/. Accessed 8 Dec 2014
112. Apache Avro: http://avro.apache.org/. Accessed 13 Dec 2014
113. Apache Etch: https://etch.apache.org/. Accessed 13 Dec 2014
114. BERT: http://bert-rpc.org/. Accessed 13 Dec 2014
115. Message Pack: http://msgpack.org/. Accessed 13 Dec 2014
116. Hessian: http://hessian.caucho.com/. Accessed 13 Dec 2014
117. ICE: http://doc.zeroc.com/display/ice34/home. Accessed 13 Dec 2014
118. CBOR: http://cbor.io/. Accessed 13 Dec 2014
119. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. In: Proceedings of USENIX Symposium on Opearting Systems Design & Implementation (2004)
120. Apache Hadoop: http://hadoop.apache.org/. Accessed 5 Sept 2014
121. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (2010)
122. MPICH: http://www.mpich.org/. Accessed 10 Dec 2014
123. Open MPI: http://www.open-mpi.org/. Accessed 10 Dec 2014
124. OpenMP: http://openmp.org/wp/. Accessed 9 Dec 2014
125. PPL: http://msdn.microsoft.com/en-us/library/dd492418.aspx. Accessed 13 Dec 2014
126. Jeon, M., He, Y., Elnikety, S., Cox, A.L., Rixner, S.: Adaptive parallelism for web search. In: Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13 (2013)
127. Jeon, M., Kim, S., Hwang, S., He, Y., Elnikety, S., Cox, A.L., Rixner, S.: Predictive parallelization: taming tail latencies in web search. In: Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR '14 (2014)
128. Lee, J., Winslett, M., Ma, X., Yu, S.: Enhancing data migration performance via parallel data compression. In: Proceedings of the 16th International Parallel and Distributed Processing Symposium, IPDPS '02 (2002)
129. Klasky, S., Ethier, S., Lin, Z., Martins, K., McCune, D., Samtaney, R.: Grid-based parallel data streaming implemented for the gyrokinetic toroidal code. In: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, SC '03 (2003)
130. Warneke, D., Kao, O.: Nephele: efficient parallel data processing in the cloud. In: Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers, MTAGS '09 (2009)
131. Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P.K., Currey, J.: Dryadlinq: a system for general-purpose distributed data-parallel computing using a high-level language. In: Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08 (2008)
132. Ronnie, C., Bob, J., Per-Ake, L., Bill, R., Darren, S., Simon, W., Jingren, Z.: Scope: easy and efficient parallel processing of massive data sets. Proc. VLDB Endow. **1**(2), 1265–1276 (2008)
133. Ahrens, J., Brislawn, K., Martin, K., Geveci, B., Charles Law, C., Papka, M.: Large-scale data visualization using parallel data streaming. In: Computer Graphics and Applications. IEEE, **21**(4), July 2001
134. Allen, M.D., Sridharan, S., Sohi, G.S.: Serialization sets: a dynamic dependence-based parallel execution model. In: Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '09 (2009)
135. Voss, M., Eigenmann, R.: Reducing parallel overheads through dynamic serialization. In: Proceedings of the 13th International Symposium on Parallel Processing and the 10th Symposium on Parallel and Distributed Processing, IPPS '99/SPDP '99 (1999)