SLoG: Large-Scale Logging Middleware for HPC and Big Data Convergence

Pierre Matri, Philip Carns, Robert Ross, Alexandru Costan, María Pérez, Gabriel Antoniu

SLoG: Large-Scale Logging Middleware for HPC and Big Data Convergence

Pierre Matri,* Philip Carns,† Robert Ross,† Alexandru Costan,‡§ María S. Pérez,* Gabriel Antoniu‡

*Universidad Politécnica de Madrid, Madrid, Spain, {pmatri, mperez}@fi.upm.es
†Argonne National Laboratory, Lemont, IL, USA, {carns, ross}@mcs.anl.gov
‡Inria, Rennes, France, {alexandru.costan, gabriel.antoniu}@inria.fr
§IRISA / INSA Rennes, Rennes, France, alexandru.costan@irisa.fr

***Abstract*—Cloud developers traditionally rely on purpose-specific services to provide the storage model they need for an application. In contrast, HPC developers have a much more limited choice, typically restricted to a centralized parallel file system for persistent storage. Unfortunately, these systems often offer low performance when subject to highly concurrent, conflicting I/O patterns. This makes difficult the implementation of inherently concurrent data structures such as distributed shared logs. Yet, this data structure is key to applications such as computational steering, data collection from physical sensor grids, or discrete event generators. In this paper we tackle this issue. We present SLoG, shared log middleware providing a shared log abstraction over a parallel file system, designed to circumvent the aforementioned limitations. We evaluate SLoG's design on up to 100,000 cores of the Theta supercomputer: the results show high append velocity at scale while also providing substantial benefits for other persistent backend storage systems.**

I. Introduction

Traditionally, cloud system designers have used purpose-specific storage services for their data storage needs. These include key-value stores [1]–[3], wide-column databases [4]–[6], or streaming message brokers [7], [8]. On the contrary, high-performance computing (HPC) platforms rest on on a much more constrained set of storage primitives, typically limited to parallel file systems [9], [10] or transient burst buffers [11], [12]. The low availability of local storage on the compute nodes of most supercomputers today and the lack of administrative access give little opportunity for users to deploy the storage system they need.

As the boundaries between HPC and big data analytics (BDA) continue to blur [13], new challenges arise. A critical objective set in this convergence context is to foster application portability across platforms [14]. Let us consider an application running in a cloud context, which uses a specialized storage service such as a distributed shared log [15]. Porting this application to HPC (e.g., to leverage specific hardware capabilities) is challenging. Indeed, deploying cloud-oriented shared log services on HPC is often not possible because of the unique specificities of these platforms. While one could consider a shared log abstraction over the available parallel file system, providing the illusion of a storage paradigm atop another is extremely difficult considering conflicting sets of constraints and APIs between the two models [16], [17].

Shared log storage is indeed one of these storage models that are both unavailable and difficult to implement on HPC platforms using the available storage primitives. Yet, in scientific applications, distributed logs could play many roles, for example for in situ visualization of large data streams, collection of telemetry events for computational steering, or data aggregation from arrays of physical sensors. A shared log enables multiple processes to append data at the end of a single byte stream. Unfortunately, in such a case, the write contention at the tail of the log is among the worst-case scenarios for parallel file systems, yielding problematically low append performance [18]. Thus, efficient streaming log middleware is needed that can leverage the primitives available on HPC platforms for persistence.

We present the SLoG[1] shared log middleware. It provides high append velocity over distributed file systems by leveraging I/O parallelism and proxying, circumventing the inherent limitations of HPC file systems. It features pluggable backends that enable it to leverage other storage models such as object stores or to transparently forward the requests to a shared log storage system when available (e.g., on cloud platforms). SLoG abstracts this complexity away from the developer, fostering application portability between platforms. We evaluated SLoG's performance at scale on a leadership-class supercomputer, using up to 100,000 cores. We measured append velocities peaking at 174 million appends per second, far beyond the capabilities of any shared log storage implementation on HPC platforms. For these reasons, we envision that SLoG could fuel convergence between HPC and big data. Our contributions are as follows.

- We leverage several use cases for distributed logs, (Section II), and **present the design of SLoG** and the diverse techniques supporting its scalability (Section III).
- **We evaluate SLoG on up to 100,000 cores of the Theta supercomputer** [19], exhibiting high append velocities on HPC platforms using both file system and object storage backends (Section IV).

We review the related work (Section V) and conclude with a discussion of future research directions and perspectives (Section VI).

[1]SLoG stands for shared log gateway

II. The case for shared log middleware

Distributed logging is a simple storage model that naturally serves a wide variety of applications. On HPC systems it allows the collection of metadata, telemetry, or monitoring events generated by simulation that can be used either live for computational steering or offline for analytics or visualization. In the case of non-bulk-synchronous applications such as discrete event simulators, shared logging is a natural model for storing the generated events that can arrive in no predefined order. Shared logs also prove critical for real-world BDA use cases, such as large-scale monitoring systems [20].

In telemetry event management, the events generated by an application carry data related to its state, its performance, or other application-specific information. Analyzing these events while the application is running can be useful, for instance to verify that the application behaves as expected or to run in situ visualization [21]. This information can also be used for in-situ analytics [22] or computational steering [23]–[25]. The latter can in turn influence the simulation in real time, for example increasing its resolution on a particular zone or rerunning part of the simulation with different parameters. We illustrate this in Figure 1. In this paper we focus on the application of telemetry event management to computational steering.

The case for a shared log on HPC systems. In order to deliver its real-time promise, computational steering requires collecting live telemetry data from all the nodes running the simulation. In parallel, one or multiple processes retrieve, analyze, and act on the gathered events *in chronological order*. This chronological order is critical for the accuracy of computational steering and is the key motivation for a distributed shared log. Moreover, a distributed shared log between the simulation and the steering helps mitigate bursting generation rates and decouple these two components.

Why are shared logs challenging? Parallel file systems are well known to be generally bad at handling concurrent, conflicting writes originating from different clients [18], [26]–[28]. Unfortunately, distributed logs are by design a highly concurrent data structure in which multiple processes essentially compete to write to the tail of a unique object. Because distributed logging is not available on HPC platforms today, few or no applications are actually built around this data structure. Yet, we advocate that distributed logging could become a relevant storage model for HPC applications in the next years, as strong as it is today for BDA.

Hence, we posit that efficient distributed shared log middleware would foster the development of applications taking advantage of computational steering at scale. It needs at the same time to evade the limited write concurrency permitted by these storage systems, while abstracting this complexity away from the developer. Moreover, when other storage models are available (e.g., object storage [29]), it should take advantage of their different capabilities to further enable performance without requiring any application modification.

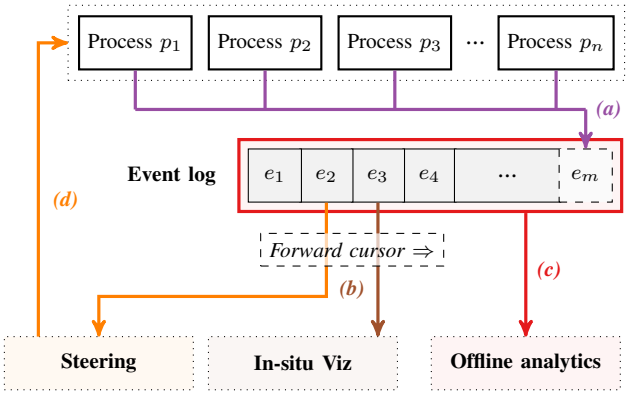


Fig. 1: Telemetry event collection. Simulation ranks push events to a log *(a)*. Such events are used for *online* tasks (e.g., in situ visualization, computational steering) *(b)*, or are stored for further offline processing *(c)*. Computational steering in turn controls the simulation *(d)*.

III. SLoG design and implementation

In this section we describe the key principles that drive the design and implementation of SLoG, large-scale shared log middleware that tackles the aforementioned challenges. SLoG adapts best practices from CORFU [15] or Kafka [7] to solve the limitations of HPC parallel file systems.

The two key design principles behind SLoG are *stream partitioning* (Section III-A) and *write proxying* (Section III-B). The former reduces write contention by distributing events to multiple partitions. The latter eliminates concurrency, channeling appends through intermediate processes each mapped to a partition. Readers are organized in *groups*, obtaining events from a dedicated *read proxy* (Section III-C). Figure 2 shows this architecture. SLoG also acts as a pass-through interface on platforms where shared log implementations are available, fueling cross-platform application portability.

A. Stream partitioning

In order to reduce write concurrency, a log is split into multiple *partitions*. This concept is similar to that of Kafka. When a writer generates an event, it is directed to one and only one partition. Each partition hence receives only a subset of all append operations. A partition is mapped to a single, dedicated file on the parallel file system, providing persistence as well as fault tolerance. The possible write throughput for a partition is hence determined by the maximum throughput the underlying file system can support for a single file. Increasing the number of partitions for a log increases the total throughput supported by the log by increasing write parallelism. The number of partitions for each log is set by the user at the time the log is created, and that number cannot be changed at a later time.

Multiple strategies are possible for determining to which partition each write is directed. These include uniform distribution (e.g., round robin) or a static mapping of writers to specific partitions. In practice, uniform distribution is the default on most systems [7]. We adopted the same default for SLoG, thus ensuring that with p partitions, each will receive n/p events out of a total of n.

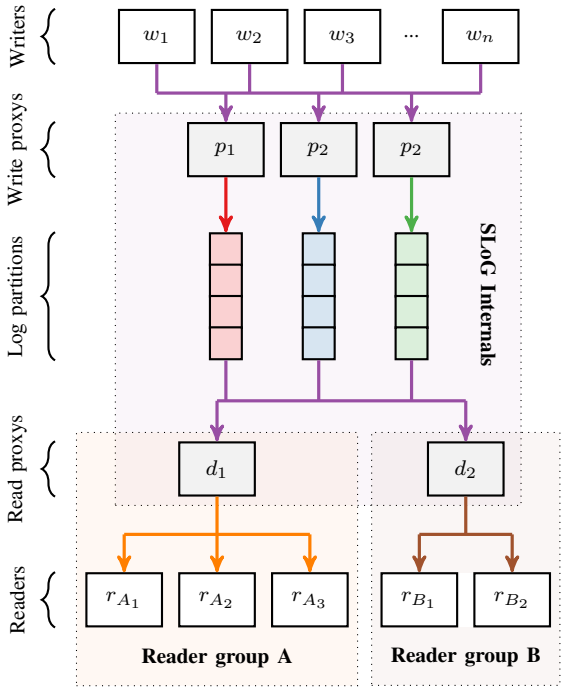
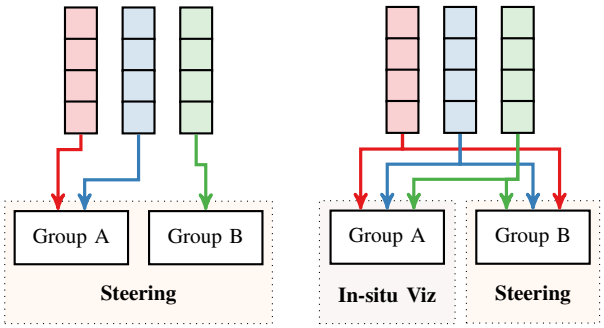


Fig. 2: SLoG high-level architecture.



(a) Increased parallelism (b) Multiple apps

Fig. 3: SLoG reader grouping strategies.

B. Write proxying

Unfortunately, enabling clients to append directly to the files that back each partition of the system would yield problematically low performance, because of the inherent low level of concurrency permitted by parallel file systems. Indeed, while these systems excel at dealing with intensive loads of isolated writes, they are known to poorly handle conflicting operations. Yet, efficient conflict management is essential for fast file appends. The POSIX specification defines an `O_APPEND` mode for this purpose, but it is inherently more difficult to implement in a distributed environment considering the strong consistency semantics of POSIX I/O, and hence it has not been the focus of parallel file system optimization.

Append proxying in SLoG circumvents this issue. This principle is inherited from best practices for HPC I/O optimization. With each partition, SLoG associates a dedicated process named *append proxy*. Through this process all append operations are channeled from the writers to a specific partition. This eliminates the costly file system access concurrency at the expense of a small increase in append latency.

The proxy process receives append operations from the writers. It maintains internally an exclusive cursor on the file that hosts the partition it is associated with, as well as an offset counter. For each append operation, it generates a `write` to the file at the next free offset, along with the current timestamp, and increments the counter by the size of the event. For optimization, incoming requests can be batched and written in bulk to the file system. Also, the proxy can be used to transparently apply further data transformations (e.g., data compression).

C. Read proxying

SLoG distributes clients in *reader groups*. For each reader group, a dedicated *read proxy* process distributes the events to the readers of this group while ensuring a total delivery order between events within a group.

Reader groups serve the principal purpose of increasing the read parallelism permitted by the system. A number of partitions are associated with each reader group. The events from these partitions are distributed to all the readers the group contains. This distribution is performed observing the order of the timestamp associated with each event in the log.

Such an approach permits great flexibility in terms of grouping strategies. For example, read parallelism can be increased by mapping distinct groups to disjoint sets of log partitions. With this configuration, a partial order of the messages is guaranteed inside a reader group. It is illustrated by Figure 3a. Mapping different groups to a common set of partitions enables multiple readers to access the same events in parallel. This is the case when multiple applications access the same log simultaneously (e.g., computational steering and in situ visualization). It is illustrated by Figure 3b. Combinations of these strategies are possible in order to increase parallelism while using multiple applications.

Read proxies serve a purpose symmetric to that of write proxies. With each reader group a single read proxy process. Its role is twofold: (1) coordinate reads from the file system for all the partitions the group is mapped to and (2) distribute the events to the readers of the group. It does so guaranteeing partial delivery ordering of the events.

A read proxy holds a cursor to all files that hold each of the partitions the group is associated with. Read requests from the clients are served in timestamp order, advancing all the file cursors in parallel. As an optimization to significantly reduce read latency, the proxy reads from the file system in chunks that are cached in its memory.

If the write proxy is configured to apply data transformations such as compression, the read proxy is responsible for applying the reverse operation, hence rendering these transformations completely transparent to the client.

IV. Evaluating SLoG at scale

In this section we discuss early experiments proving the relevance of the design of SLoG. We focus on *append throughput*, which is harder to scale than read throughput.

A. Leveraging a high-end experimental platform

All experiments carried out in this paper are based on the Theta supercomputer [19], hosted at the Argonne Leadership Computing Facility (ALCF) of Argonne National Laboratory. Theta is a 9.63 petaflops system, ranked 16 in the TOP500 as of June 2017. It is composed of 3,624 nodes, each containing a 64-core Knights Landing Xeon Phi CPU with 16 GB of high-bandwidth in-package memory (MCDRAM), 192 GB of DDR4 RAM, and a 128 GB SSD. Networking leverages a Cray Aries interconnect with Dragonfly configuration.

The Theta storage system is based on Lustre. Its 10 TB parallel file system spans 170 storage servers. Theta is also connected to the GPFS file system of the Mira supercomputer colocated in the same facility using high-speed interconnects. This configuration allows for experiments on both Lustre and GPFS from the same experimental platform.

The minimal API required for computational steering is straightforward to benchmark. Indeed a single operation, `append`, is performed by all event generator ranks. We develop a simple MPI-based benchmark that sequentially generates events at maximum rate from a number of generator ranks. These 1 KB events are appended to the log using that operation. The computational steering process is simulated by a single dedicated rank that reads the logs concurrently, repeatedly reading events from the log. Since we focus only on storage I/O, the data is discarded immediately by this process. We advocate that this benchmark, although simple, is well suited for evaluating the maximum performance achievable by the system. The `append` operation causes a very high write contention in distributed logging systems, a large number of processes concurrently push data at maximum rate to a single log puts a very high pressure on the storage system.

B. Evaluated persistence backends

Besides the Lustre and GPFS file systems, we experimented using SLoG with two additional backends:

Zlog [30], which implements the CORFU [15] log design over the Ceph file system [31]. At its core lies a network sequencer, an atomic counter initialized with the size of the file and incremented by the size of each write.

Týr [32], which is an object store targeted at coping with highly concurrent I/O patterns. It features a lightweight transaction protocol that natively enables atomic operations such as appends to be performed at high speed.

These two systems provide a baseline performance compared with (1) a purpose-built shared log storage system (i.e., Zlog) and (2) an object storage system built for concurrent I/O patterns (i.e., Týr).

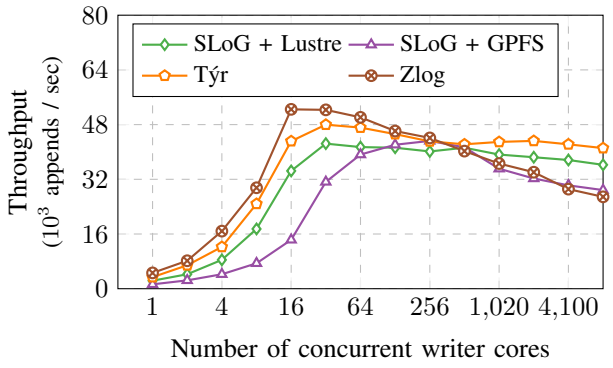


Fig. 4: SLoG single-partition append performance, using up to 8,192 append client cores, with 1 KB appends.

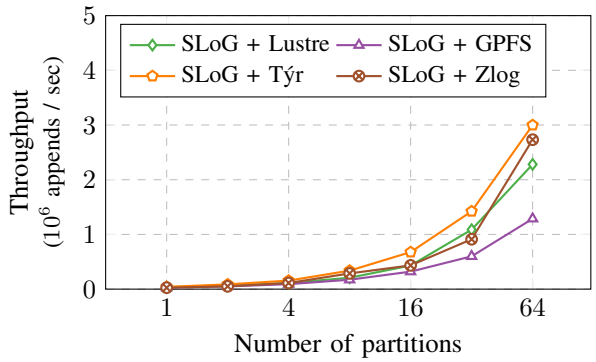


Fig. 5: SLoG multi-partition append performance, using 8,192 append client cores, with 1 KB appends.

C. Baseline append performance with a single partition

We start evaluating the baseline performance of SLoG using a single log. This demonstrates the relevance and efficiency of write proxying.

Theta is one of the few supercomputing platforms offering access to the local SSD drives of the compute nodes, we can compare Týr and Zlog on the machines. The application uses between 1 and 8,192 cores as append clients. We deploy Týr and Zlog using between 64 and 1,536 cores for the storage servers, on different machines from the append clients (we measured this ratio to be optimal for our setup). We measure the number of appends per process per core over 10 minutes. We do not use any write batching or compression. Týr and Zlog are configured with a replication factor of 3.

We plot the results on Figure 4. Overall, they show that the performance achieved with SLoG is competitive with that of Týr or Zlog when the number of concurrent writers is low, despite the overhead of write proxying and the limitations of the underlying file system. As soon as the I/O capabilities of a single write proxy are saturated, however, the write performance offered by SLoG with both Lustre and GPFS reaches a plateau, slightly decreasing as more writers are added. Týr and Zlog exhibit similar behavior, which is due to the high write contention at the tail of the log.

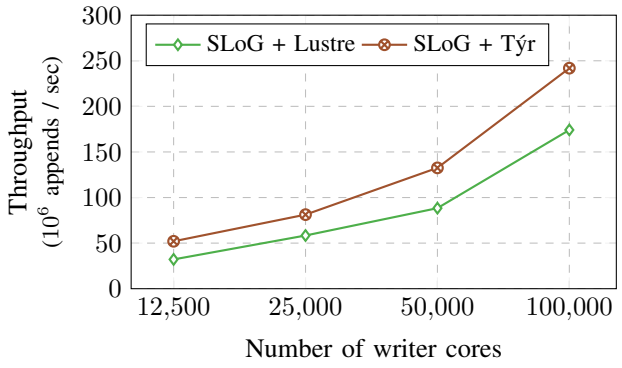


Fig. 6: SLoG horizontal scalability up to 100,000 append client cores, using up to 6,250 SLoG partitions, with 1 KB appends.

D. Varying the number of partitions

The performance decrease observed with a single log when the write proxy gets saturated is the main motivation driving the parallelism model of SLoG. By evenly distributing the writes across partitions, we increase the number of proxies, hence increasing the total write capacity of the log.

Since neither Zlog nor Týr offers such a parallelism model to support high logging velocity, we implement for each a SLoG backend. For Zlog, each partition is mapped to a Zlog log. For Týr, we map a partition to a blob. Since thee systems are built for concurrent I/O, the SLoG client library accesses the storage systems directly, bypassing the write proxy.

In Figure 5 we plot the results. We show how increasing the number of partitions impacts the performance of the storage. We use the same setup as in the preceding section with 8,192 clients, varying the number of partitions between 1 and 64. The results show a clear increase in the throughput per client. As expected, doubling the number of partitions roughly doubles the achieved throughput on the log by evenly splitting the load across multiple files, reducing hot spots at the tail of the log and consequently increasing throughput.

We also measure the achieved read performance during this experiment. We do not note any substantial performance degradation when increasing the number of partitions, with only a 5,3% performance drop between 1 and 8,192 partitions.

E. Scaling up to 100,000 append cores

We demonstrate the scalability of the system by varying the number of client cores from 12,500 to 100,000. Based on our previous experiments, 16 client cores per proxy yield the best overall performance. We hence vary the number of partitions proportionally between 780 and 6,250. We use both Týr and Lustre as backends, using the same configuration as in the preceding section. Týr is deployed on 20,000 storage cores.

Overall, the results depicted in Figure 6 confirm the near-horizontal scalability that can be achieved with SLoG by increasing the number of concurrent partitions, using either file- or object-based storage as the backend. We measure a throughput of up to 174 million appends per second on Lustre and up to 241 million appends per second with Týr.

V. Related work

SLoG seeks to enable HPC applications to leverage distributed shared logs over the storage primitives provided by the platform. This tackles the challenges posed by large-scale computational steering. At smaller scale, however, researchers have used comparable approaches on HPC platforms.

In [33], Agelastos et al. propose a lightweight distributed service able to collect live metrics on the Blue Waters system. However, this tool is specifically built for system monitoring. Knüpfer et al. propose a comparable contribution in [34], describing a large-scale monitoring infrastructure capable of collecting performance metrics from a large number of compute nodes. In contrast with these two contributions, SLoG tackles a more general problem with significantly higher throughput requirements.

The challenges of parallel I/O to a single file on HPC systems have been tackled by Frings et al. in [35]. Indeed, their design of SIONlib enables a large number of processes to concurrently collaborate on a single file. While that work is well suited for a number of use cases, its goal is not to provide a total ordering across the events that is crucial for distributed logging. Actually, while not the target of this paper, SIONlib could be seen as complementary to SLoG by enabling all partitions to share a single file.

Also relevant to the distributed logging field are the excellent publications describing state-of-the-art streaming data systems, including CORFU [15], Kafka [7], and BookKeeper [8]. SLoG builds on the best practices learned from these systems to cope with the hard specificities of HPC storage systems. SLoG does not intend to replace such systems but to complement them by providing an alternative for platforms on which they are not available.

VI. Conclusion and perspectives

Distributed shared logs provide a relevant storage model for a number of applications (e.g., computational steering, in situ visualization). However, they are currently not available on HPC platforms, typically offering parallel file systems for persistent storage and only limited possibilities for installing specialized storage services such as distributed shared logs.

In this paper we introduce the SLoG middleware, proposing a distributed shared log abstraction over a parallel file system. To provide high write throughput, we circumvent the low write concurrency of parallel file systems using write proxying, while at the same time partitioning the log to ensure high horizontal scalability. We show on up to 100,000 cores of the Theta supercomputer that SLoG delivers its promise of high append performance with multiple persistence backends, which include object stores in addition to parallel file systems.

We foresee that such an interface could serve in the context of HPC and big data convergence, providing users with a high-level abstraction of distributed logging that can adapt to a variety of backend systems depending on the primitives available on each platform. SLoG could therefore prove to be a strong building block for distributed logging, fueling application portability between HPC and big data platforms.

ACKNOWLEDGMENT

This work is part of the "BigStorage: Storage-based Convergence between HPC and Cloud to handle Big Data" project (H2020-MSCA-ITN-2014-642963), funded by the European Commission within the Marie Skłodowska-Curie Actions framework. This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (Contract DE-AC02-06CH11357). It is also supported by the ANR OverFlow project (ANR-15-CE25-0003). We thank Quincey Koziol (LBNL), Noah Watkins (UC Santa Cruz), Robert Jacobs (ANL), and Jay Lofstead (SNL) for their expert assistance. The experiments presented were carried out by using resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility (Contract DE-AC02-06CH11357).

REFERENCES

[1] G. DeCandia, D. Hastorun, M. Jampani *et al.*, "Dynamo: Amazon's highly available key-value store," in *ACM SIGOPS operating systems review*, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[2] V. Srinivasan, B. Bulkowski, W.-L. Chu *et al.*, "Aerospike: Architecture of a real-time operational dbms," *Proceedings of the VLDB Endowment*, vol. 9, no. 13, pp. 1389–1400, 2016.

[3] R. Escriva, B. Wong, and E. G. Sirer, "HyperDex: A distributed, searchable key-value store," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 25–36.

[4] A. Khetrapal and V. Ganesh, "HBase and Hypertable for large scale distributed storage systems," *Dept. of Computer Science, Purdue University*, pp. 22–28, 2006.

[5] F. Chang, J. Dean, S. Ghemawat *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[6] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[7] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, 2011, pp. 1–7.

[8] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: Wait-free coordination for internet-scale systems." in *USENIX annual technical conference*, vol. 8, no. 9. Boston, MA, USA, 2010.

[9] P. Schwan, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the 2003 Linux symposium*, vol. 2003, 2003, pp. 380–386.

[10] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for linux clusters," in *Proceedings of the 4th annual Linux showcase and conference*, 2000, pp. 317–328.

[11] N. Liu, J. Cope, P. Carns *et al.*, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*. IEEE, 2012, pp. 1–11.

[12] T. Wang, S. Oral, Y. Wang, B. Settlemyer, S. Atchley, and W. Yu, "Burstmem: A high-performance burst buffer system for scientific applications," in *2014 IEEE International Conference on Big Data*. IEEE, 2014, pp. 71–79.

[13] D. A. Reed and J. Dongarra, "Exascale computing and Big Data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[14] G. Fox, J. Qiu, S. Jha, S. Ekanayake, and S. Kamburugamuve, "Big Data, simulations and HPC convergence," in *Big Data Benchmarking*. Springer, 2015, pp. 3–17.

[15] M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobber, M. Wei, and J. D. Davis, "CORFU: A shared log design for flash clusters," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 1–1.

[16] H. Greenberg, J. Bent, and G. Grider, "MDHIM: A parallel key/value framework for HPC." in *HotStorage*, 2015.

[17] P. Kalmegh and S. B. Navathe, "Graph database design challenges using HPC platforms," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*. IEEE, 2012, pp. 1306–1309.

[18] Y. Tsujita, K. Yoshinaga, A. Hori, M. Sato, M. Namiki, and Y. Ishikawa, "Multithreaded two-phase I/O: Improving collective MPI-IO performance on a Lustre file system," in *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb 2014, pp. 232–235.

[19] K. Harms, T. Leggett, B. Allen, S. Coghlan, M. Fahey, C. Holohan, G. McPheeters, and P. Rich, "Theta: Rapid installation and acceptance of an XC40 KNL system," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 1, 2018.

[20] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, "MonALISA: A distributed monitoring service architecture," *arXiv preprint cs/0306096*, 2003.

[21] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro, "Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework," in *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*. IEEE, 2013, pp. 67–75.

[22] S. Ziegeler, C. Atkins, A. Bauer, and L. Pettey, "In situ analysis as a parallel I/O problem," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, ser. ISAV2015. New York, NY, USA: ACM, 2015, pp. 13–18.

[23] H. Yi, M. Rasquin, J. Fang, and I. A. Bolotnov, "In-situ visualization and computational steering for large-scale simulation of turbulent flows in complex geometries," in *2014 IEEE International Conference on Big Data*, Oct 2014, pp. 567–572.

[24] S. Ko, J. Zhao, J. Xia *et al.*, "VASA: Interactive computational steering of large asynchronous simulation pipelines for societal infrastructure," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 1853–1862, Dec 2014.

[25] D. Butnaru, G. Buse, and D. Pflüger, "A parallel and distributed surrogate model implementation for computational steering," in *2012 11th International Symposium on Parallel and Distributed Computing*, June 2012, pp. 203–210.

[26] R. McLay, D. James, S. Liu, J. Cazes, and W. Barth, "A user-friendly approach for tuning parallel file operations," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2014, pp. 229–236.

[27] J. Logan and P. Dickens, "Towards an understanding of the performance of MPI-IO in Lustre file systems," in *2008 IEEE International Conference on Cluster Computing*, Sept. 2008, pp. 330–335.

[28] F. Tessier, V. Vishwanath, and E. Jeannot, "TAPIOCA: An I/O library for optimized topology-aware data aggregation on large-scale supercomputers," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept. 2017, pp. 70–80.

[29] P. Matri, Y. Alforov, A. Brandon, M. Kuhn, P. Carns, and T. Ludwig, "Could blobs fuel storage-based convergence between HPC and big data?" in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 81–86.

[30] "Zlog," https://cruzdb.github.io/zlog/, 2017, accessed: 2017-11-28.

[31] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[32] P. Matri, A. Costan, G. Antoniu, J. Montes, and M. S. Pérez, "Týr: Blob storage meets built-in transactions," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 573–584.

[33] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC14*. IEEE, 2014, pp. 154–165.

[34] A. Knüpfer, C. Rössel, D. an Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony *et al.*, "Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir," in *Tools for High Performance Computing 2011*. Springer, 2012, pp. 79–91.

[35] W. Frings, F. Wolf, and V. Petkov, "Scalable massively parallel I/O to task-local files," in *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*. IEEE, 2009, pp. 1–11.