Making the case for reforming the I/O software stack of extreme-scale systems

Florin Isaila, Javier Garcia, Jesus Carretero

University Carlos III

Rob Ross, Dries Kimpe

Argonne National Laboratory

Abstract

The ever-increasing data needs of scientific and engineering applications require novel approaches to managing and exploring huge amounts of information in order to advance scientific discovery. To achieve this goal, one of the main priorities of the international scientific community is addressing the challenges of performing scientific computing on exascale machines within the next decade. Future exascale platforms will be likely, characterized by a three to four orders of magnitude increase in concurrency, a substantially larger storage capacity, and a deepening of the storage hierarchy. In the face of these foreseeable evolutions, the current development model of independently applying optimizations at each layer of the system I/O software stack will not scale to the new levels of concurrency, storage hierarchy, and capacity. In this article we discuss the current system software development model for the I/O software stack of high-end computing platforms. We identify the challenges of improving scalability, performance, energy efficiency, and resilience of the I/O software stack, while accessing a deepening hierarchy of volatile and non-volatile storage. We advocate for radical new approaches to reforming the I/O software stack in order to advance toward exascale.

Keywords:
storage, I/O software stack, data locality, energy efficiency, resilience

1. Introduction

The ever-increasing data needs of scientific and engineering applications require novel approaches and larger computing infrastructures to manage and explore huge amounts of information [21]. To achieve this goal, one of the main priorities of the international scientific community is addressing the challenges of performing scientific computing on Exaflop machines by 2020. One of the most critical scalability challenges is scaling the current software I/O software stack [10]. Understanding the limitations of the I/O software stack in petascale systems and proposing radical solutions to address them have become crucial for reaching the goal of building exascale systems, as they are expected to have a high impact on increasing the rate of scientific productivity.

In this article we argue that the current uncoordinated development model of independently applying optimizations at each layer of the system software I/O software stack will not scale to the new levels of concurrency, storage hierarchy, and capacity [10]. Radical new approaches to reforming the I/O software stack are needed in order to enable holistic system software optimizations that can address cross-cutting issues such as power, resiliency, and performance. The key insight is to investigate cross-layer control mechanisms and run-times, seeking to unify the access to several layers of volatile and non-volatile memories and to improve the scalability, performance, and resilience of the I/O software stack.

The remainder of the paper is organized as follows. Section 2 presents an overview of data-related challenges related to exascale architectures. The subsequent sections discuss five main issues to be addressed in order to prepare the I/O software stack for new levels of scalability: storage I/O data path coordination (Section 3), cross-layer load balance (Section 4), exposing and exploring data locality (Section 5), energy efficiency (Section 6), and resilience (Section 7). Section 8 concludes with a brief summary.

2. Data-related challenges

Increasingly the researchers are recognizing that the evolution to exascale will require overcoming several system-wide data-related challenges. In this section we present an overview of the main data-related challenges classified in four categories as shown in Figure 1: applications, platform, I/O software stack, and cross-cutting issues.

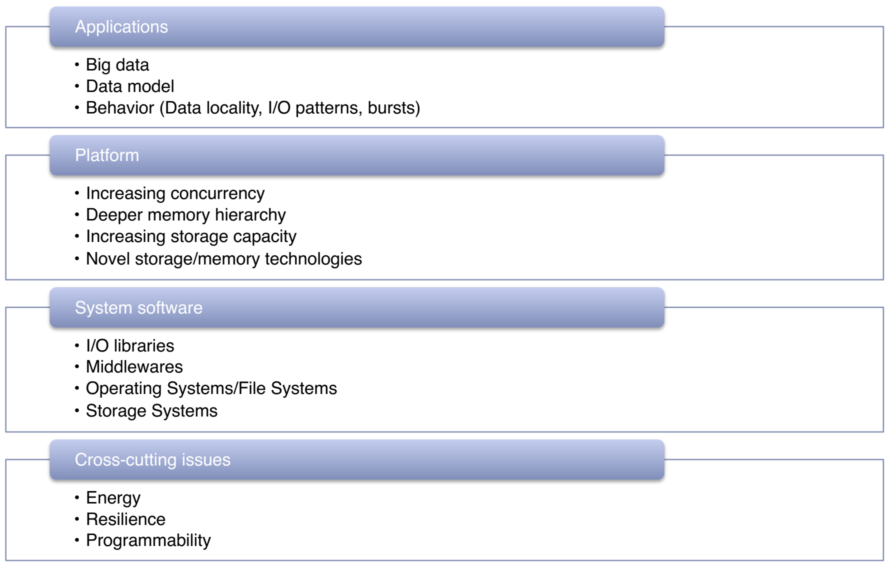

Figure 1: Data-related exascale challenges

2.1. Applications

Many scientific applications running on today's large-scale supercomputers are data intensive. Table 1 shows examples of data requirements for some applications at Argonne National Laboratory. These numbers applications commonly generate tens of terrabytes in a single simulation. Additionally, all the applications from this sample require at least 1 TByte of on-line data.

The data requirements of scientific applications are widely expected to become larger in the next few years, increasing the pressure on the storage I/O system. Thus, it becomes increasingly important to better understand application data models and to be able to efficiently map them on the underlying storage. High-level I/O libraries such as HDF5 [6], Parallel-NetCDF [25], and ADIOS [29] facilitate the design of application data models and find increasing acceptance among computational scientists, but they still face important challenges. In particular, mapping application models on storage models is still inefficient, as no mechanisms are available for system-wide optimization of storage I/O access. For instance, the common POSIX storage model has significant shortcomings that limit its usability in large-scale environments: strong consistency requirements strangle performance and scalability, it lacks support for data locality and flexible data layout, and it does not have a failure model. Because of these limitations, most high-level I/O libraries (i.e.,

Table 1: Data requirements for selected 2011 INCITE applications at Argonne Leadership Computing Facility (ALCF) - Argonne National Laboratory.

Project	On-Line Data (TBytes)	Off-Line Data (TBytes)
Combustion in Gaseous Mixtures	1	17
Protein Structure	1	2
Laser-Plasma Interactions	60	60
Type Ia Supernovae	75	3000
Nuclear Structure and Reactions	6	15
Fast Neutron Reactors	100	100
Lattice Quantum Chromodynamics	300	70
Fracture Behavior in Materials	12	72
Engineering Design of Fluid Systems	3	200
Multimaterial Mixing	215	100
Turbulent Flows	10	20
Earthquake Wave Propagation	1000	1000
Fusion Reactor Design	50	100

HDF5, PnetCDF, ADIOS) pursue more relaxed consistency models for performance reasons and have defined their data models different from POSIX.

Application behaviour plays an important role in efficiently using the I/O storage system. Many scientific applications have bursty access patterns, alternating periods of intense storage I/O requests with periods of no I/O at all [13]. These patterns can cause either contention or underutilization. Novel approaches are necessary for absorbing peak demands and distributing them over periods of inactivity. Further, data access locality of applications needs to be better exploited in order to reduce communication.

2.2. Platforms

The architecture of most of today's leadership HEC systems is hierarchical [7], as shown in the right-hand side of Figure 2. For better scalability, computation is performed on a large number of strongly interconnected cores and is segregated from storage resources. The storage I/O functionality is typically offloaded to dedicated I/O nodes. The I/O nodes are connected over a high-performance network to storage servers. The storage servers

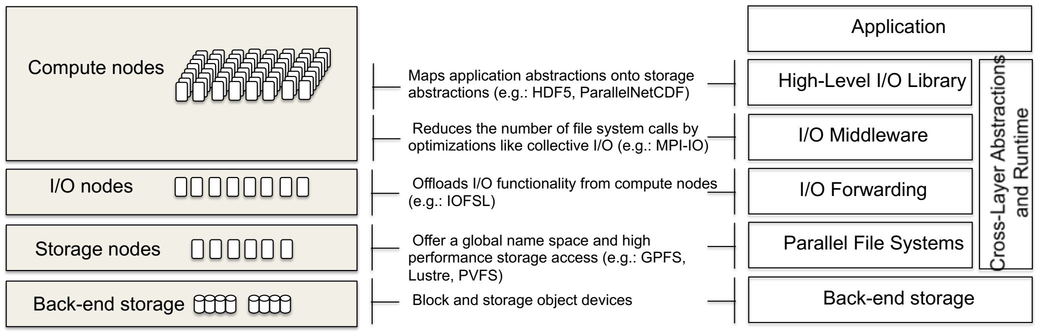
Compute nodes
I/O nodes
Storage nodes
Back-end storage
Maps application abstractions onto storage abstractions (e.g.: HDF5, ParallelNetCDF)
Reduces the number of file system calls by optimizations like collective I/O (e.g.: MPI-IO)
Offloads I/O functionality from compute nodes (e.g.: IOFSL)
Offer a global name space and high performance storage access (e.g.: GPFS, Lustre, PVFS)
Block and storage object devices
Application
High-Level I/O Library
I/O Middleware
I/O Forwarding
Parallel File Systems
Back-end storage
Cross-Layer Abstractions and Runtime

Figure 2: Mapping of the I/O software stack (right-hand side) on the architecture of current large-scale high-end computing systems (left-hand side)

offer access to the back-end storage (i.e., NVRAM, disks, and tape).

Future Exaflop platforms will be likely characterized by a three to four orders of magnitude increase in concurrency, a substantially larger storage capacity, and a deepening of the storage hierarchy [18]. Building exascale systems will require architectural and technological shifts for overcoming the current scalability barriers, while staying within an target energy budget of 20 MW.

Non-volatile RAM (NVRAM) is considered one of the main memory technologies that will have a significant impact on energy-efficiency, performance, access semantics, and resilience of the future exascale systems [18]. Recent studies have focused on architectural solutions integrating RAM and NVRAM [14], redefining system primitives to accommodate the integration of NVRAM in the memory hierarchy [33], examining the role of NVRAM to absorb I/O bursts of scientific applications [28], and analyzing the impact of NVRAM on scientific applications [24]. However, the impact of introducing the NVRAM technologies in the storage stack remains an open problem, with direct implications for all I/O stack levels, while the technology evolves. Recently, researchers have expressed increasing agreement that scaling the current I/O stack will require NVRAM storage burst buffer capability for staging data in and out of an exascale machine [10]. However, various open questions remain such as: where the burst buffers will be located (on the compute nodes or I/O nodes), what is going to manage the burst buffer (file system or middleware), which mechanisms and APIs are needed for controlling the burst buffers, and how the burst buffers will be connected (into the interconnect or on a data channel such as SAS).

2.3. I/O software stack

Figure 2 shows the mapping of the current I/O software stack on the architecture of high-end computing systems [16]. High-level I/O libraries such as HDF5 [6] and Parallel-NetCDF [25] run on compute nodes on top of middleware. The middleware layer (e.g., MPI-IO [2]) runs as well on compute nodes and provides access optimizations such as collective I/O [38] in order to reduce the number of I/O operations sent to I/O nodes and file systems. However, since many applications do not use collective I/O, I/O delegation has been proposed as a middleware mechanism for coalescing and optimizing independent I/O operations [32]. High-level I/O library and middleware layers are optional; many applications access the storage through the POSIX interface [13]. The recently developed I/O Scalability Forwarding Layer [8] reduces the I/O load on compute nodes by offloading I/O functionality to dedicated I/O nodes. The parallel file system layer provides a logical name space and parallel access to back-end storage through servers running on storage nodes. Parallel file systems such as GPFS [36], Lustre [1], and Orange/PVFS [3] have been designed for supporting tens of thousands of clients, but they will not scale for the expected concurrency increase of three to four orders of magnitude. File system servers access and manage data and metadata stored on the back-end storage.

The model for the I/O software stack described above suffers from a number of problems:

1. As the depth of the storage hierarchy increases, one of the biggest concerns is the I/O software stack programmability, defined as the facility of developing global performance optimizations by system programmers. I/O system optimizations are often applied independently at each system layer. However, this approach can cause mismatches between the requirements of different layers (for instance in terms of load, locality, consistency) [10]. Because of this uncoordinated development, understanding the interplay between optimizations at different layers has become difficult even for parallel I/O researchers [26].
2. The path through the I/O software stack lacks mechanisms for adapting to unexpected events such as sudden data volume variations and failures. However, managing data volume variability and failures requires cross-layer adaptive control mechanisms, which are unfortunately not available in the state-of-the-art I/O software stack for HEC platforms.

3. As the system scale increases, exposing and exploring data locality will become a key factor for improving both application performance and energy efficiency. The expected deepening of the storage hierarchy brings novel opportunities to exploit locality at different layers of the system architecture (e.g., compute node, I/O node, storage nodes). However, existing storage I/O interfaces such as POSIX lack the capability of exposing and exploiting data locality.

These are critical problems for the promises that exascale platforms hold. In this paper we argue that they need to be addressed in a synergistic manner, through radical approaches that enable cross-layer optimizations of the I/O software stack in exascale platforms.

Cross-layer mechanisms have already been successfully proposed in networking research in order to optimize system performance [27]. In storage I/O most cross-layer optimizations are applied ad-hoc and involve two adjacent layers [26, 22, 35]. The need for an I/O middleware for efficiently matching the applications requirements to storage has been identified as well by the exascale I/O workgroup (EIOW) [5].

2.4. Cross-cutting issues

In The International exascale Software Project Roadmap, the authors identify a number of cross-cutting issues that will affect all aspects of system software and applications at exascale [18]. Here we discuss the relationship of three of these aspects (energy, resilience, and programmability) with the redesign of the I/O software stack.

While energy efficiency can be improved to a substantial extent through technological and architectural innovations, it has been shown that an increasing amount of energy consumption is due to moving data inside a large scale system [30]. In this respect, supporting the development of I/O software through cross-layer mechanisms could enhance energy efficiency by reducing the communication through a better exploitation of data locality and layout.

Resilience requires a system-wide approach. However, the current I/O software stack lacks a failure model, since no mechanisms are available for detecting, isolating, and/or recovering from failures of different stack layers. In our opinion, a cross-layer mechanism with clear failure semantics can support the design of resilient techniques and optimizations at different stack layers.

Further, the design of cross-layer mechanisms and abstractions needs to contribute to enhancing programmability, defined as the ease of developing applications. The I/O software stack has to be redesigned to easily expose and exploit data locality and concurrency, to hide the latencies, to avoid congestion, and to identify and bypass failures in an early stage.

3. Storage I/O data path coordination

One of the most important shortcomings of the system software in the HEC machines is the lack of mechanisms for coordinating storage I/O policies across the various layers of the I/O stack. This is likely to limit the scalability of the storage hierarchy on future machines.

In our opinion novel mechanisms and abstractions are needed for controlling the I/O flow on given paths between compute nodes and back-end storage, for instance, for rerouting the I/O flow when encountering traffic congestions or failures. We mention here three main aspects. First, research is needed to identify information and mechanisms that have to be offered at each stack layer in order to support the cross-layer control of the storage I/O path. Second, hiding the access latency inside the I/O software stack can be improved through proactive multiple-layer data-staging policies, for instance, by leveraging the predictable patterns of data-intensive applications for controlling write-back policy for writes and prefetching policy for reads. Third, adaptive data staging can help change the data flow policy for unexpected events such as unpredicted bursts. A possible solution to these issues can be based on integrating an adaptive buffering mechanism into each I/O software stack layer. For instance, a minimal adaptive mechanism will permit buffers to dynamically shrink/grow, change the write-behind/prefetching strategy, and change next hop in the data path from compute nodes to storage. An adaptive buffering mechanism can coordinate the I/O data path based on the cross-layer control mechanisms, which will continuously monitor and distribute run-time information such as failures, performance, and resource utilization metrics at various layers. The results of one of our studies integrating write-back and prefetching policies at two different levels of I/O software stack of a Blue Gene/P system indicate that I/O coordination of the I/O data flow may bring a substantial performance benefit to applications [22].

4. Cross-layer load balance in the I/O software stack

The continuous evolution of application requirements and platforms will likely require more flexibility for shifting the load along the layers of the I/O stack in order to improve the resource utilization, while reducing the contention. Causes of storage I/O overload include data aggregation, processing of data for exploiting locality, network transfers, and OS jitter.

For instance, the I/O forwarding layer has been proposed to off-load I/O functions such as file system calls from compute nodes to I/O nodes [8]. This approach increases the scalability of the computation on current HEC systems by reducing the operating system jitter on application nodes. However, the expected concurrency growth of three to four orders of magnitude will increase the pressure on the I/O nodes and will likely require a rebalance of I/O functionality. Current I/O forwarding mechanisms rely on function shipping, are employed unidirectionally, and are used for bridging only two architectural layers (i.e., forward I/O from compute nodes to I/O nodes).

In our opinion the I/O software stack needs a flexible, cross-layer I/O load-balancing mechanism. One possible solution is to build a channel for bidirectional off-loading of I/O functionality by transporting hints, feedback, and notifications and by shipping I/O functionality across layers. This channel will facilitate devising novel experiments for evaluating critical aspects such as integration of NVRAM at various software stack layers, in-situ processing vs. post-processing, techniques for absorbing peak demands and staging progressively, hierarchical aggregation of data, and hierarchical exploitation of data locality.

5. Exposing and exploiting data locality

In future exascale systems most energy is expected to be spent moving data [18]. Hence, efficiently managing and exploiting data locality will become critical for system scalability within a limited energy budget.

In the current state-of-the-art software stack, data locality cannot be fully controlled on the data path from application to the storage, which may cause inefficient access. In particular, the standard MPI-IO data operations are not locality-aware, no mechanisms exist that allow data accesses to take advantage of data layout in the file system (e.g. I/O caching, data distribution over servers, block alignment). On the other hand, most parallel file systems strive to comply with the POSIX requirements. However, one of the main critiques of POSIX is that it does not expose data locality [34]. Other scalable file systems such as HDFS [20] have dropped the POSIX interface and have been codesigned with processing frameworks such as Map-Reduce [17]. While these frameworks may simplify programming of a class of embarrassingly parallel data-intensive computations and include best-effort strategies to colocate computation and data, they are not general enough to efficiently address the needs of scientific applications and have not been widely adopted on the HEC infrastructure.

Exploiting data locality is directly linked to data layout awareness and control. Existing solutions either offer the applications mechanisms for controlling data layout of the file system (e.g., PVFS distributions [3], Clusterfile logical and physical layout [23]) or propose intermediary layers for optimally exploiting given data layouts through collective I/O [38], block alignment [26], or mapping of application data models on a file system optimized layout [11].

Additionally, a limited number of works have leveraged topology-awareness for improving the performance of storage I/O stack. Topology-awareness has been used for automatically selecting the number of data aggregators for collective I/O operations [15]. Son et al. [37] present an approach for leveraging topology in I/O caching for shared storage systems. However, their approach focuses on a particular topology: a two-dimensional mesh network. File systems such as PanFS [4] and Ceph [42] offer infrastructure topology awareness, but to the best of our knowledge these mechanisms have not been explored for building topology-aware storage services or for improving the scalability of the I/O software stack.

The lack of coordination of data locality management can cause inefficiencies due to factors such as data redundancies, unnecessary data reorganization, unaligned access, lost locality-aware optimization opportunities, unnecessary consistency constraints. To the best of our knowledge, there exist currently no abstractions and mechanisms for coordinating data locality management across the I/O software stack. These cross-layer abstractions and mechanisms can substantially improve the programmability of system-wide data locality control (e.g., by taking into consideration data layout, architectural characteristics such as topology or storage hierarchy). While we do not advocate writing the I/O stack from scratch, we believe that a proper restructuring based on cross-layer feedback will significantly increase the capacity to optimize individual I/O stack layers and rebalance the I/O stack functionality across layers depending on the architectural and technological advancements.

6. Energy efficiency

Energy is the one of the most prominent barriers in achieving exascale. According to opinion leaders two orders of magnitude improvement in energy efficiency is required for building cost-efficient exascale systems within the next decade. A large share of this improvement corresponds to data, as energy cost of moving data has exceeded the energy cost of computing [19].

As a cross-cutting aspect, energy efficiency of the storage I/O stack can be improved through both software and hardware solutions. With respect to hardware, non-volatile RAM (NVRAM) is considered one of the storage technologies that will have a significant impact on energy-efficiency, performance, access semantics, and resilience of the future exascale systems [18].

With respect to software, exploiting data locality across the whole I/O stack and storage hierarchy, as discussed in Section 5, is expected to bring the highest benefit in terms of energy efficiency. However, other directions may offer promising results. We believe that, if properly balanced, even the current storage systems can support larger-scale concurrency, as it remains underutilized for significantly large time windows because of the bursty I/O behavior [13]. Absorbing bursts in low-energy NVRAM devices and gradually spilling them to disks [28] is a promising technique. However, the role of the burst buffers inside the storage hierarchy and the I/O software stack is still an open research issue. Other techniques developed for data centers such as power-aware layout [9], gear-levelling [41, 39], and write-offloading [31, 40] can be further useful for reducing the energy spent by disks in periods of low activity, but their application in HEC systems needs to be investigated.

7. Resilience

The expected scale of the future systems will increase the probability of failures. As we advance toward exascale, it becomes critical to minimize the interruption of system operation in the presence of failures. Current experience shows that storage software is one of the main causes of application interruptions. For instance, Table 2 shows the root causes of interrupt events on the Intrepid Blue Gene/P system at Argonne National Laboratory during 2010 and 2011. The majority of problems arise from misbehaving clients that propagated through the entire system. In this case this propagation was made possible by the tight coupling among distributed components of the GPFS file system. File system state is shared with clients in order to enforce POSIX storage abstraction and to enable coherent client caching.

Table 2: Root causes of interrupt events on the Argonne Intrepid BG/P system for 2010-2011. Thanks to C. Goletz and B. Allcock (ANL).

Component	2010	2011
GPFS	101	77
Machine	79	35
Myrinet HW/SW	28	32
Scheduler	14	42
Service node (DB)	29	8
PVFS	15	7
Admin Error	8	7
DDN	6	0
Service network	2	0

One key for improving resilience is to decouple the health of the components of the storage system based on stateless protocols (e.g., NFSv3, PVFS) and through techniques such as sandboxing. However, achieving failure isolation across the storage software stack is nontrivial: it has direct implications on the abstractions offered by the storage system, especially when caching and buffering are done at several system levels. In this context, reducing the complexity of resilience problem will most likely require: (1) pushing strong consistency support up into the libraries or applications, (2) providing efficient mechanisms for cross-layer information dissemination in order to support the adaptation process, and (3) providing mechanisms for redirecting traffic based on failure and for balancing client traffic with reconstruction.

Achieving a high degree of resilience still faces many open problems. Novel storage abstractions are needed, because the popular POSIX abstraction does not include a failure model, while its strong consistency requirements make the implementation of resilient protocols difficult. Further, research is needed for better understanding which failures the storage system should automatically protect against and which should be reported to the users and require their intervention. In general, failure traceability and early reaction capacity have to be substantially improved in order to adequately respond to the expectations of future exascale systems.

8. Conclusions

In this paper we have made the case for reforming the I/O software stack of extreme scale systems. We advocate for cross-layer control mechanisms based on application hints, feedback dissemination, notification, and I/O functionality shipping throughout the I/O software stack. These mechanisms are currently not available for supporting the development of storage I/O optimizations. We claim that novel approaches of providing adaptive control of data staging throughout the I/O stack are needed. We are not aware of any current approach of coordinating this data path along the vertical dimension from application to storage. We believe that I/O researchers should focus on improving the data path from applications to storage through data models that will smooth evolution from the currently popular POSIX to novel approaches that better map complex data to the storage system. As energy is directly related to data locality, action is needed for addressing the current lack of support for exposing and exploiting data locality throughout the I/O software stack. We believe that timely reforming the I/O stack will smooth the way toward reaching exascale, by facilitating the scaling to expected increases in concurrency and storage hierarchy depth [12].

Acknowledgements

This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

This research has been partially funded by the Spanish Ministry of Science and Innovation under grant TIN2010-16497 "Input/Output techniques for distributed and high-performance computing environments". The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 328582.

References

[1] Lustre file system. http://www.lustre.org., 2013.

[2] MPI forum. http://www.mpi-forum.org/, 2013.

[3] OrangeFS/PVFS. http://www.pvfs.org., 2013.

[4] PANFS web page. http://www.panasas.com/products/panfs, 2013.

[5] The Exascale I/O initiative (EIOW). Available at http://www.eiow.org/., 2013.

[6] The HDF group. http://www.hdfgroup.org/HDF5/, 2013.

[7] Top 500 Supercomputing Sites. Available at http://www.top500.org., 2013.

[8] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, P. Sadayappan, Scalable I/O Forwarding Framework for high-performance computing systems, in: Proceedings of IEEE Conference on Cluster Computing, New Orleans, LA.

[9] H. Amur, J. Cipar, V. Gupta, G. Ganger, Robust and flexible power-proportional storage, ACM Symposium on Cloud Computing (SOCC) (2010).

[10] M. Bancroft, J. Bent, E. Felix, G. Grinder, J. Nunez, S. Poole, R. Ross, E. Salmon, L. Ward, HEC File Systems and I/O (HEC FSIO) Workshop Document. http://institute.lanl.gov/hec-fsio/docs/., Technical Report, 2011.

[11] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, M. Wingate, PLFS: a checkpoint filesystem for parallel applications, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, ACM, New York, 2009, pp. 21:1–21:12. URL: `http://doi.acm.org/10.1145/1654059.1654081`.

[12] F. Cappello, B. Mohr, Working Group report on software eco-system., Technical Report, European Exascale Software Initiative., 2011.

[13] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, R. Ross, Understanding and improving computational science storage access through continuous characterization, Transactions on Storage 7 (2011) 8:1–8:26. URL: `http://doi.acm.org/10.1145/2027066.2027068`.

[14] A.M. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R.K. Gupta, A. Snavely, S. Swanson, Understanding the impact of emerging non-volatile memories on high-performance, I/O-intensive computing, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, Washington, DC, pp. 1–11.

[15] M. Chaarawi, E. Gabriel, Automatically selecting the number of aggregators for collective I/O operations, in: CLUSTER, pp. 428–437.

[16] J. Cope, K. Iskra, D. Kimpe, R. Ross, Bridging HPC and grid file I/O with IOFSL, in: Proceedings of the 10th international conference on Applied Parallel and Scientific Computing - Volume 2, PARA'10, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 215–225.

[17] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (2008) 107–113. URL: `http://doi.acm.org/10.1145/1327452.1327492`.

[18] J. Dongarra, P. Beckman, et al., "the international exascale software roadmap", International Journal of High Performance Computer Applications 25 (2011) 3–60.

[19] M. Duranton, S. Yehia, D.B. Schaffer, K. de Bosschere, J. Maebe, The HiPEAC vision for advanced computing in Horizon 2020, HiPEAC Project, 2013.

[20] S. Ghemawat, H. Gobioff, S.T. Leung, The Google file system, in: Proceedings of the nineteenth ACM Symposium on Operating Systems Principles, SOSP '03, ACM, New York, NY, 2003, pp. 29–43. URL: `http://doi.acm.org/10.1145/945445.945450`.

[21] T. Hey, S. Tansley, K. Tolle (Eds.), The Fourth Paradigm: Data-Intensive Scientific Discovery, Microsoft Research, Redmond, Washington, 2009. URL: `http://research.microsoft.com/en-us/collaboration/fourthparadigm/`.

[22] F. Isaila, J. Garcia Blas, J. Carretero, R. Latham, R. Ross, Design and evaluation of multiple-level data staging for Blue Gene systems, IEEE Trans. Parallel Distrib. Syst. 22 (2011) 946–959.

[23] F. Isaila, W. Tichy, Clusterfile: A flexible physical layout parallel file system, Concurrency and Computation: Practice and Experience 15 (2003) 653–679.

[24] D. Li, J.S. Vetter, G. Marin, C. McCurdy, C. Cira, Z. Liu, W. Yu, Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications, in: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS '12, IEEE Computer Society, Washington, DC, 2012, pp. 945–956.

[25] J. Li, W.k. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, M. Zingale, Parallel netCDF: A high-performance scientific I/O interface, in: Proceedings of the 2003 ACM/IEEE conference on Supercomputing, SC '03, ACM, New York, 2003, pp. 39–. URL: `http://doi.acm.org/10.1145/1048935.1050189`.

[26] W.k. Liao, A. Choudhary, Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols, in: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08, IEEE Press, Piscataway, NJ, 2008, pp. 3:1–3:12. URL: `http://dl.acm.org/citation.cfm?id=1413370.1413374`.

[27] X. Lin, N.B. Shroff, R. Srikant, A tutorial on cross-layer optimization in wireless networks, IEEE Journal on Selected Areas in Communications 24 (2006) 1452–1463.

[28] N. Liu, J. Cope, P.H. Carns, C.D. Carothers, R.B. Ross, G. Grider, A. Crume, C. Maltzahn, On the role of burst buffers in leadership-class storage systems, in: MSST, pp. 1–11.

[29] J.F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, C. Jin, Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS), in: Proceedings of the 6th international workshop on Challenges of large applications in distributed environments, CLADE '08, ACM, New York, 2008, pp. 15–24. URL: `http://doi.acm.org/10.1145/1383529.1383533`.

[30] I. Manousakis, M. Marazakis, A. Bilas, FDIO : A feedback driven controller for minimizing energy in I/O-intensive applications., in: HotStor-

age'13: The 5th USENIX Workshop on Hot Topics in Storage and File Systems., USENIX, New York, 2013.

[31] D. Narayanan, A. Donnelly, A. Rowstron, Write off-loading: Practical power management for enterprise storage, Trans. Storage 4 (2008) 10:1–10:23. doi:10.1145/1416944.1416949.

[32] A. Nisar, W.k. Liao, A. Choudhary, Delegation-based I/O mechanism for high performance computing systems, IEEE Trans. Parallel Distrib. Syst. 23 (2012) 271–279.

[33] X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, D.K. Panda, Beyond block I/O: Rethinking traditional storage primitives, in: Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, HPCA '11, IEEE Computer Society, Washington, DC, 2011, pp. 301–311. URL: `http://dl.acm.org/citation.cfm?id=2014698.2014867`.

[34] I. Raicu, I.T. Foster, P. Beckman, Making a case for distributed file systems at Exascale, in: Proceedings of the third International Workshop on Large-scale System and Application Performance, LSAP '11, ACM, New York, 2011, pp. 11–18.

[35] E. Santos-Neto, S. Al-Kiswany, N. Andrade, S. Gopalakrishnan, M. Ripeanu, Enabling cross-layer optimizations in storage systems with custom metadata, in: Proceedings of the 17th International Symposium on High Performance Distributed Computing, HPDC '08, ACM, 2008, pp. 213–216. URL: `http://doi.acm.org/10.1145/1383422.1383451`.

[36] F. Schmuck, R. Haskin, GPFS: A shared-disk file system for large computing clusters, in: Proceedings of the 1st USENIX Conference on File and Storage Technologies, FAST '02, USENIX Association, Berkeley, CA, 2002. URL: `http://dl.acm.org/citation.cfm?id=1083323.1083349`.

[37] S.W. Son, M.T. Kandemir, Y. Zhang, R. Garg, Topology-aware i/o caching for shared storage systems, in: ISCA PDCCS, pp. 143–150.

[38] R. Thakur, W. Gropp, E. Lusk, Data Sieving and Collective I/O in ROMIO, in: Proceedings of the The 7th Symposium on the Frontiers

of Massively Parallel Computation, FRONTIERS '99, IEEE Computer Society, Washington, DC, 1999, pp. 182–189.

[39] E. Thereska, A. Donnelly, D. Narayanan, Sierra: Practical power-proportionality for data center storage, in: EuroSys '11: Proceedings of the sixth Conference on Computer Systems, ACM Request Permissions, 2011.

[40] A. Verma, R. Koller, L. Useche, R. Rangaswami, SRCMap: energy proportional storage using dynamic consolidation, Proceedings of the 8th USENIX Conference on File and Storage Technologies (2010) 20–20.

[41] C. Weddle, M. Oldham, J. Qian, A.i.A. Wang, PARAID: The gearshifting power-aware RAID, in: In Proceedings of the USENIX Conference on File and Storage Technologies, USENIX Association, 2007, pp. 245–260.

[42] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, C. Maltzahn, Ceph: A scalable, high-performance distributed file system, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06, USENIX Association, Berkeley, CA, 2006, pp. 307–320. URL: `http://dl.acm.org/citation.cfm?id=1298455.1298485`.