Towards Scalable I/O Architecture for Exascale Systems

Yong Chen
Department of Computer Science
Texas Tech University
yong.chen@ttu.edu

ABSTRACT
High performance computing (HPC) has crossed the Petaflop mark and is reaching the Exaflop range quickly. The exascale system is projected to have millions of nodes, with thousands of cores for each node. At such an extreme scale, the substantial amount of concurrency can cause a critical contention issue for I/O system. The contention can destroy the request locality, increase the access latency, and waste the precious I/O interconnection bandwidth. This study proposes a *dynamically coordinated I/O architecture* for exascale systems. The fundamental idea is to coordinate I/O accesses according to access pattern, network topology, interconnection condition, and data distribution on storage devices to reduce the contention and regain the locality. The preliminary results have shown the promise of a dynamically coordinated I/O architecture. It has a potential to manage the substantial amount of I/O concurrency and provides a scalable I/O architecture for exascale systems.

Categories and Subject Descriptors
C.4 [**Performance of Systems**]: Design studies. D.4.3 [**File Systems Management**]: Access methods.

General Terms
Performance

Keywords
Exascale systems; parallel I/O; parallel file systems; scalable I/O architecture; storage; many-task computing; high performance computing

1. INTRODUCTION
Many scientific and engineering simulations in critical areas of research, such as nanotechnology, astrophysics, climate, bioinformatics, and high-energy physics, are highly data intensive [DBMA11, IBCL11, DOSW09, SACC09, RaFZ08, Brya07, DICI]. These applications contain a large number of I/O accesses, where large amounts of data are stored to and retrieved from storage. The application teams are beginning to process terabytes or tens of terabytes of data in a single simulation. For example, a turbulence simulation code GTC (Gyrokinetic Toroidal Code) running on 29K cores of the Jaguar machine at the Oak Ridge Leadership Computing Facility (OLCF) of Oak Ridge National Laboratory (ORNL) generated over 54 terabytes of data in a 24-hour period [GTC, Jaguar]. As discussed in [RLUW09], 12 out of 20 INCITE applications run at the Argonne Leadership Computing Facility (ALCF) of Argonne National Laboratory (ANL) generated datasets in the terabyte range and store them on-line [INCITE]. The volume of the data and the pressure on the I/O system capability substantially increases over the time.

The exascale system is projected to appear around the year of 2018. By that time, application teams are predicted to process hundreds of terabytes or even petabytes of data in a single simulation run [DBMA11]. Such an extreme-scale system is projected to have millions of nodes, with thousands of cores for each node [DBMA11, ShDM10]. Table 1 shows a potential exascale HPC system design and a comparison with current HPC systems in terms of factor changes for the peak performance, system size, etc. [VTYR08]. It should be noted that *the most significant change is the total concurrency* with a factor change of 4,444. This change of the total concurrency is anticipated as the most challenging issue for HPC systems to achieve

Table 1 Potential Exascale Computer Design and Its Relationship to Current HPC designs [VTYR08]

	2010	2018	Factor Change
System Peak	2 Pf/s	1 Ef/s	500
Power	6 MW	20 MW	3
System Memory	0.3 PB	10 PB	33
Node Performance	0.125 Tf/s	10 Tf/s	80
Node Memory BW	25 GB/s	400 GB/s	16
Node Concurrency	12 CPUs	1000 CPUs	83
Interconnect BW	1.5 GB/s	50 GB/s	33
System Size (nodes)	20 K nodes	1 M nodes	50
Total Concurrency	225 K	1 B	4444
Storage	15 PB	300 PB	20
Input/Output Bandwidth	0.2 TB/s	20 TB/s	100

MTAGS'11, November 14th, 2011, Seattle, WA, USA.

an exascale. Such a factor change means that an exascale system will have billions of processes running on millions of nodes in order to achieve the anticipated exaflop performance. The exascale system brings critical challenges than ever for the I/O system to be able to meet billions of processes' demand of processing hundreds of terabytes of data simultaneously. The I/O system performance is predicted as one of the most critical challenges presented by the exascale HPC systems [DBMA11, LPGK11, ABBD10, SACC09, BBFG09, RaFZ08]. The limited I/O system capability could considerably lower the sustained performance of exascale systems. There is a great research need in visiting the I/O system challenges, and designing and developing an I/O architecture that is able to achieve such an extreme scale and to meet the applications' demand on exascale systems.

Various methods have been developed to improve HPC I/O system performance. These methods have been focused on two directions in general, *exploring parallelism* and *exploring locality*. Both parallelism and locality are not easily achieved for exascale systems. At an exascale, the substantial amount of concurrency can cause a critical contention issue, at multiple levels including node level, card level, plane level, and storage level. The contention at various levels can destroy the locality that generations of researchers have been striving to achieve, because the contention causes I/O requests compete and interfere with each other. Many critical challenges presented by exascale systems have to be investigated and well addressed for us to enter an exaflop era. The current HPC I/O architecture has been designed as one-set-for-all and has been static. Such a one-set-for-all and static I/O architecture does not manage the concurrency intelligently and limits the scalability and the potential of I/O systems at an extreme scale.

This study proposes a *dynamically coordinated I/O architecture* (or *coordinated I/O* in short) for exascale systems. We argue that coordinating I/O accesses according to access pattern, network topology, and data distribution on storage devices to reduce the contention and regain the locality is critical and fundamental to exascale systems. The goal of the dynamically coordinated I/O architecture is to *manage the substantial amount of concurrency with coordination among I/O requests on the fly to achieve parallelism but avoid the critical contention issue at an extreme scale*. It should be an important component for many-task computing paradigm [RaFZ08], where many tasks and processes can cause serious contention in I/O accesses without a proper coordination. The preliminary tests conducted for both independent I/O and collective I/O have shown the promise of a dynamically coordinated I/O architecture. It has a real potential to manage the enormous amount of I/O concurrency and provide a scalable I/O architecture for exascale systems.

2. RELATED WORK

There has been significant amount of research efforts in improving I/O performance and providing scalable I/O solutions at various levels. At the library level, notable solutions include collective I/O [MayJ01, ThGL99], two-phase and extended two-phase I/O [BoRC93, ThCh96], server-direct I/O [SCJJ95], disk-directed I/O [Kotz97], partitioned collective I/O [YuVe08], and resonance I/O [ZhJD09]. Parallel file systems (PFS), such as Lustre [Cfsi00], GPFS [ScHa02], PanFS [WUAG08], PVFS/PVFS2 [CLRR00], and PPFS2 [TrRe04], enable concurrent I/O accesses from multiple clients to files. All these file systems provide high bandwidth for large, well-formed parallel I/O requests. Recent studies that improve the I/O system performance for petascale/exascale machines include data staging services [IBCL11, AWEK10, AEWS11], I/O forwarding (either a hardware or software solution) that ships the I/O calls to dedicated I/O nodes to improve the performance [VHIK10, ACIK09, IRYB08], latent I/O asynchrony approach [WPBW09, WWAM11], interference removal with a data replication approach [ZhJi10], and caching and prefetching optimizations [NiLC08, EHHN10, BCST08, CBST08a, ZLMZ08]. These existing studies have shown the great need of improving the I/O system performance, which is crucial for exascale systems. This study focuses on one critical I/O challenge for exascale systems, the contention issue, and addresses it with a dynamically coordinated I/O architecture.

3. DYNAMICALLY COORDINATED I/O ARCHITECTURE

This study proposes to design and develop a *dynamically coordinated I/O architecture* to address challenges presented by exascale systems. The design contains two major components: *dynamic data coordinator (DDC)* and *dynamic request analyzer (DRA)* as shown in Figure 1. The purpose of the DRA component is to obtain three types of information: the *data distribution* on storage servers, the *network topology and I/O interconnect condition*, and the *I/O access pattern* from the application. The data distribution on storage servers (including the data layout strategy, striping width, and striping factor) can be obtained via parallel file systems API. The obtaining of data distribution knowledge happens at the first time an I/O occurs. The data distribution information can then be cached in runtime system. Such a caching is safe as the data layout of a specific file is determined when it is created and will be static except deleted or explicitly changed. The network topology is static and can be revealed to the DRA component as well. The DRA obtains the I/O interconnection usage information to assist the coordination. Such information can be periodically sampled and provided. The DRA component obtains the application's I/O requests and analyzes access pattern as well, which is used to direct the coordination. With the data distribution, I/O interconnect usage, and access pattern of applications provided by the DRA component, the DDC component coordinates the I/O

accesses to manage the substantial amount of concurrency and to mitigate the contention issue on exascale systems. The DDC component orchestrates the I/O requests in both independent I/O and collective I/O operations, which have been observed with serious contention issue at a large scale [CSTS10, JiCS10]. The dynamic coordination for both independent and collective I/O is discussed in the following subsections.

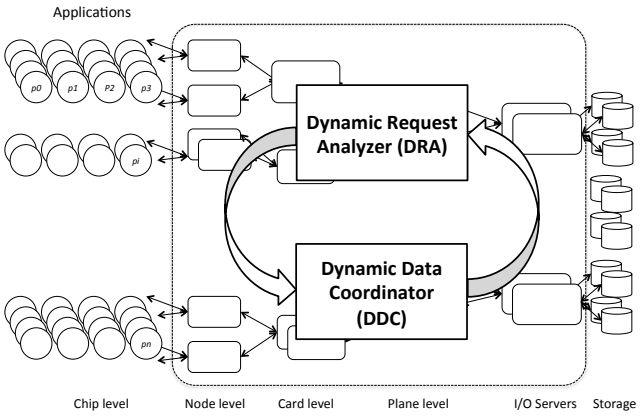

Figure 1. Dynamically Coordinated I/O Architecture

Independent I/O and collective I/O are two important forms of I/O, and both of them remain critical for exascale systems. The independent I/O can be performed by any individual process or any subset of processes of a parallel application. The advantage of the independent I/O is that users have the freedom to perform any I/O; however, the downside is that the I/O system has no idea of what other processes might do and therefore have to service the I/O requests of each process individually. Collective I/O addresses this limitation by having a group of processes participating together in I/O activities, merging the reads and writes with the knowledge of all participating processes' requests, and carrying out them more efficiently.

3.1 Dynamic Coordination for Independent I/O

Due to the independent nature, the existing independent I/O strategy merely utilizes the underlying file system calls to realize it, without coordinating with other processes. Such an approach is fine if the amount of concurrency is not high and the simultaneous requests do not exceed the I/O server capability and the bandwidth. However, this approach is problematic when the concurrency level is high and the amount of simultaneous independent I/O requests exceeds the bandwidth and the I/O server capability. The ignorance of other processes can introduce contention, destroy the locality of requests due to the interference, and increase the latency drastically. The dynamic coordination can be critical for the independent I/O on exascale systems. Without a dynamically coordinated I/O, we not only lose the potential benefit of exploring the correlation among accesses as in the collective I/O, but also deteriorate the I/O performance because of the increased contention and diminishing locality. The dynamically coordinated I/O considers the amount of concurrency and orchestrates the substantial amount of simultaneous requests so that the request from a specific process is serviced continuously, instead of interrupted randomly by other processes, as shown in Figure 2. The coordination takes the amount of concurrency, the amount of available bandwidth and the server capability into consideration. Even though the coordinated I/O does not combine and optimize accesses with other processes as in the collective I/O case, it avoids and reduces the performance loss due to the contention of other processes substantially. Thus it can improve the performance effectively and is more scalable as demonstrated through preliminary tests (Section 4.1).

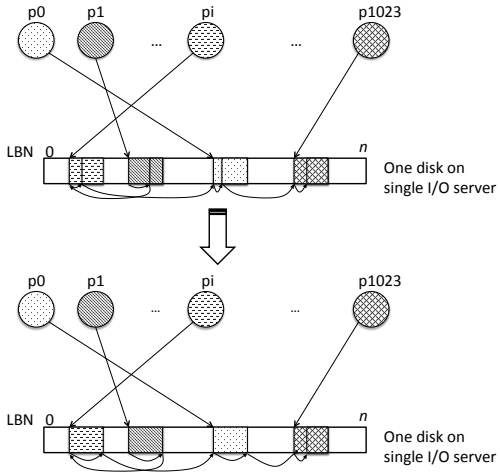

Figure 2. Dynamically Coordinated I/O

3.2 Dynamic Coordination for Collective I/O

This study explores dynamically coordinated I/O for collective I/O to manage the concurrency and reduce the contention as well. The DDC component obtains the data layout, the network condition, and the I/O requests via the DRA component similar as in the case of handling independent I/O requests. The DDC leverages this information to orchestrate and rearrange these requests to reduce the contention. Figure 3 demonstrates such a coordinated I/O. Without dynamic coordination, the collective I/O strategy can cause extensive network traffic and contention when accessing data, as shown in Figure 3 (a). The coordinated I/O rearranges the partitions of file domains (at the logical view) in collective I/O and the requests of aggregators such that: 1) the requests are grouped and need as few file servers as possible to reduce access contention and explore better parallelism; 2) the requests are reordered to be physically contiguous as much as possible to explore better locality. Figure 3 (b) demonstrates how file domain partitions and access requests are rearranged following the dynamic coordination with the example in Figure (a). Note that the coordination here is to change the requests that each aggregator carries out on behalf of the processes. In other words, the coordination changes the responsible portion of the aggregators and the way the aggregators access data. It is critical to note that the

coordination does not exchange data themselves among aggregators. The dynamically coordinated I/O is also designed for both I/O reads and writes. Figure 3 (a) and (c) show the communication and I/O pattern of the collective I/O strategy without and with dynamic coordination. It can be observed that the dynamically coordinated I/O groups accesses with the consideration of concurrency and contention and results in the accesses in a matched and neater way.

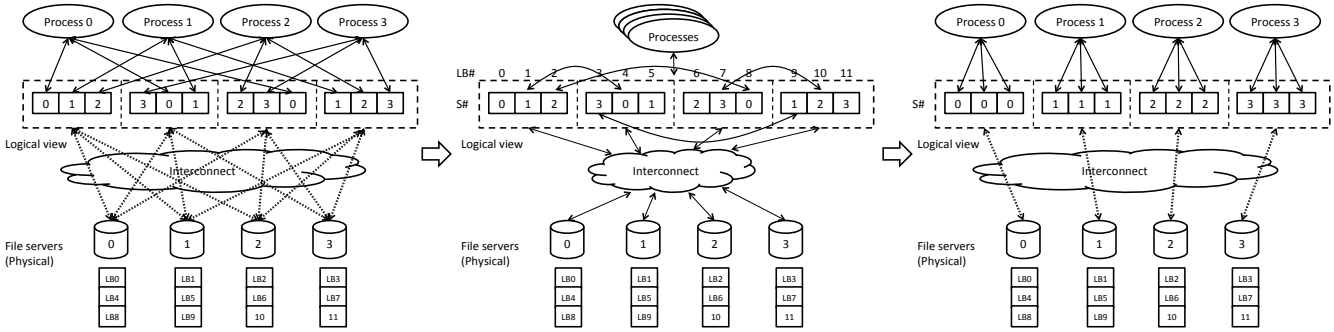

Figure 3. Dynamically Coordinated I/O

4. PRELIMINARY RESULTS

A set of experimental tests was carried out on a 65-node Linux-based cluster test bed. Even though the current evaluation scale is far less than the exascale systems, the experiments conducted on the test bed demonstrated the issues of I/O contention, reduced locality, and increased latency well. The experiments have also verified the potential of the dynamically coordinated I/O. The experiments were conducted with MPICH2-1.0.5p3 release and PVFS 2.8.1 file system on Ubuntu 4.3.3-5 system with kernel 2.6.28.10. The IOR-2.10.2 benchmark and one user-level checkpointing/restart application were used for tests.

4.1 Results of Dynamic Coordination for Independent I/O

Figure 4 reports the results of the dynamically coordinated I/O for a user-level checkpointing application. The application's performance was nearly doubled with the dynamic coordination in the case of 128 concurrent processes conducting simultaneous I/O. In this application, all processes follow a coordinated checkpointing protocol by reaching a global consistent state first, then issue application-level checkpointing by writing the runtime data into persistent data storage. We varied the number of client processes to test the performance under different scenarios, but keep the same total image size of the whole application in all cases. For instance, the first set of bars in the figure represents the case with 8 processes and each process writes an image of 2000MB. Even though the total amount of I/O requests is the same for all cases, the execution time was increased when the number of processes increased. This time increase is primarily due to the contention from many processes and the degraded locality because of the contention. With the coordinated I/O, the execution time was decreased by 35.2% on average. Furthermore, it can be observed that dynamic coordination achieved stable performance under various cases, which demonstrates that it explores better locality and reduces contention. The coordinated I/O is scalable for increased system sizes.

4.2 Results of Dynamic Coordination for Collective I/O

The preliminary evaluation has confirmed the promising benefits of dynamic coordination for collective I/O as well. Figure 5 reports the average bandwidth improvement of the coordinated I/O with the IOR benchmark for a set of

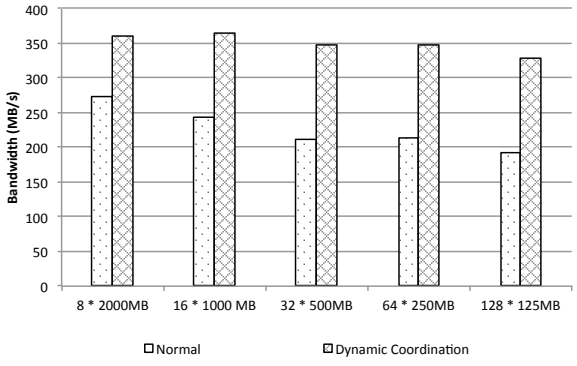

Figure 4. Bandwidth Comparison with and without Dynamic Coordination

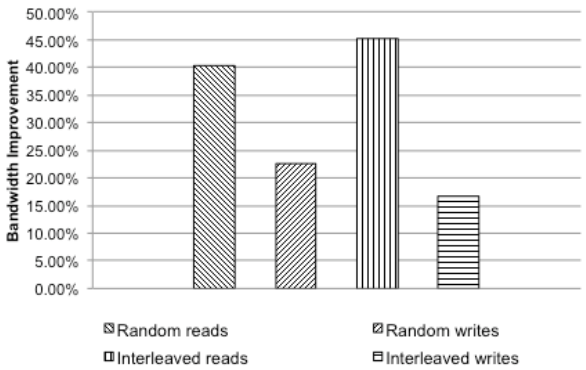

Figure 5. Bandwidth Improvement of Dynamic Coordination

random reads, random writes, interleaved reads, and interleaved writes patterns. The average bandwidth was considerably improved, and was up to 46% in the interleaved reads case.

5. ONGOING WORK

The coordinated I/O is an access-aware, topology-aware, and layout-aware I/O architecture. This awareness is achieved via the DRA and DDC components that analyze data accesses, network topology, and data layout to direct I/O. It can manage the growing amount of I/O concurrency that causes critical I/O contention and diminished locality issues and has demonstrated a potential through preliminary tests. While the coordinated I/O is under further development and exploration, we are also working on the integration with parallel programming models. The intention is to let users better represent the I/O activities and access patterns, which can assist to perform the dynamic coordination in an even better way. In addition, we are in the process of modeling the dynamic coordination and analyze the potential in theory. The theoretical analysis can help better understandings of I/O challenges and issues in the post-petascale era and in the coming exascale era. We are carrying out evaluations at a scale of O(10K-100K) processes as well.

6. CONCLUSION

With the exascale systems near the horizon, it is critical to design and develop a scalable I/O architecture for such ultra large scale systems. The exascale HPC systems present critical challenges to the I/O architecture in terms of substantial amount of concurrency and contention, and reduced locality and increased latency in I/O requests. The proposed *coordinated I/O* intends to address these issues, which will meet the need of exascale systems and the growing demand of data-intensive science and simulations that will be run on exascale HPC systems. The preliminary studies have shown a promise of the coordinated I/O. In the near future, we will continue the design and development of dynamically coordinated I/O and the research exploration along this direction. The long-term goal of this research is to provide a scalable I/O architecture for extreme scale systems with access-aware, topology-aware, and layout-aware solutions.

7. REFERENCES

[RaFZ08] I. Raicu, I. Foster, Y. Zhao. Many-Task Computing for Grids and Supercomputers. IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08), 2008.

[ABBD10] K. Alvin, B. Barrett, R. Brightwell, S. S. Dosanjh, A. Geist, K. S. Hemmert, M. A. Heroux, D. Kothe, R. C. Murphy, J. Nichols, R. Oldfield, A. Rodrigues and J. S. Vetter. On the Path to Exascale. IJDST 1(2): 1-22 (2010)

[ACIK09] N. Ali, P. H. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. B. Ross, L. Ward, P. Sadayappan. Scalable I/O Forwarding Framework for High-performance Computing Systems. Proceedings of the 2009 IEEE International Conference on Cluster Computing, 2009.

[AEWS11] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan and S.Klasky. Just in time: adding value to the IO pipelines of high performance applications with JITStaging. HPDC 2011: 27-36

[AWEK10] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan and F. Zheng. DataStager: scalable data staging services for petascale applications. Cluster Computing 13(3): 277-290 (2010)

[BBFG09] M. Bancroft, J. Bent, E. Felix, G. Grider, J. Nunez, S. Poole, R. Ross, E. Salmon, L. Ward. HEC FSIO 2008 Workshop Report. High End Computing Interagency Working Group (HECIWG) Sponsored File Systems and I/O Workshop HEC FSIO 2009

[BCST08] S. Byna, Y. Chen, X.-H. Sun, R. Thakur and W. Gropp. Parallel I/O Prefetching Using MPI File Caching and I/O Signatures. In Proc. of the ACM/IEEE SuperComputing Conference (SC'08), Nov. 2008.

[BoRC93] R. Bordawekar, J. M. d. Rosario, A. N. Choudhary: Design and Evaluation of primitives for Parallel I/O. SC 1993: 452-461

[Brya07] R. E. Bryant. Data-intensive supercomputing: The case for DISC. In Tech Report CMU-CS-07-128, Carnegie Mellon University School of Computer Science, 2007.

[CBST08a] Y. Chen, S. Byna, X.-H. Sun, R. Thakur, W. Gropp. "Exploring Parallel I/O Concurrency with Speculative Prefetching," in Proc. 37th International Conference on Parallel Processing (ICPP'08), Sept. 2008.

[Cfsi00] Cluster File Systems Inc. Lustre: A scalable, high performance file system. Whitepaper, http://www.lustre.org/docs/whitepaper.pdf

[CLRR00] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A Parallel File System For Linux Clusters'. Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000, pp. 317-327.

[CSTS10] Y. Chen, X.-H. Sun, R. Thakur, H. Song and H. Jin. Improving Parallel I/O Performance with Data Layout Awareness. In the Proc. of the IEEE International Conference on Cluster Computing 2010 (Cluster'10), 2010.

[DBMA11] J.Dongarra, P. H. Beckman, et. al. The International Exascale Software Project roadmap. IJHPCA 25(1): 3-60 (2011)

[DOSW09] D. Donofrio, L. Oliker, J. Shalf, M. F. Wehner, C. Rowen, J. Krueger, S. Kamil and M. Mohiyuddin. Energy-Efficient Computing for Extreme-Scale Science. IEEE Computer 42(11): 62-71 (2009)

[DICI] Data-Intensive Computing Initiative. http://dicomputing.pnl.gov/.

[EHHN10] M. Eshel, R. L. Haskin, D. Hildebrand, M. Naik, F. B.Schmuck and R. Tewari. Panache: A Parallel File System Cache for Global File Access. In Proc. of the 8th USENIX Conference on File and Storage Technologies, 2010.

[GTC] Gyrokinetic Particle Simulations Gyrokinetic Toroidal Code (GTC) http://w3.pppl.gov/theory/proj_gksim.html

[INCITE] DOE Innovative and Novel Computational Impact on Theory and Experiment program, http://hpc.science.doe.gov/

[IRYB08] K. Iskra, J. W. Romein, K. Yoshii, and P. Beckman. ZOID: I/O Forwarding Infrastructure for Petascale Architectures. In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 153 -162, 2008.

[IBCL11] F. Isaila, J. G. Blas, J. Carretero, R. Latham, R. B. Ross. Design and Evaluation of Multiple-Level Data Staging for Blue Gene Systems. IEEE Trans. Parallel Distrib. Syst. 22(6): 946-959, 2011.

[Jaguar] Jaguar supercomputer at the Oak Ridge Leadership Computing Facility of Oak Ridge National Laboratory, http://www.olcf.ornl.gov/computing-resources/jaguar/

[JiCS10] H. Jin, Y. Chen and X.-H. Sun. Optimizing HPC Fault-Tolerant Environment: An Analytical Approach. In the Proc. of the 39th International Conference on Parallel Processing (ICPP'10), 2010.

[Kotz97] D. Kotz: Disk-Directed I/O for MIMD Multiprocessors. ACM Trans. Comput. Syst. 15(1): 41-74 (1997)

[LPGK11] J. F. Lofstead, M.Polte, G. A. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf and Q.Liu. Six degrees of scientific data: reading patterns for extreme scale science IO. HPDC 2011: 49-60

[MayJ01] J. May. Parallel I/O For High Performance Computing. Morgan Kaufmann Publishing, 2001.

[NiLC08] A. Nisar, W.-K. Liao, A. Choudhary. Scaling Parallel I/O Performance through I/O Delegate and Caching System. SC 2008.

[SACC09] V. Sarkar, S. Amarasinghe, et. al. ExaScale Software Study: Software Challenges in Extreme Scale Systems. ExaScale Computing Study, DARPA IPTO, 2009

[ScHa02] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In First USENIX Conference on File and Storage Technologies, pages 231--244. USENIX, Jan. 2002.

[SCJJ95] K. E. Seamons, Y. Chen, P. Jones, J. Jozwiak and M. Winslett. Server-Directed Collective I/O in Panda. SC'95.

[ShDM10] John Shalf, Sudip S. Dosanjh and John Morrison. Exascale Computing Technology Challenges. VECPAR 2010: 1-25

[ThGL99] R. Thakur, W. Gropp, and E. Lusk. Data Sieving and Collective I/O in ROMIO. In Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, February 1999, pp. 182-189.

[ThCh96] R. Thakur, and A. Choudhary. An Extended Two-Phase Method for Accessing S ections of Out-of-Core Arrays. Scientific Programming, (5)4:301-317, 1996.

[TrRe04] N. Tran and D. A. Reed. Automatic ARIMA Time Series Modeling for Adaptive I/O Prefetching. IEEE Trans. Parallel Distrib. Syst. 15(4): 362-377 (2004).

[VHIK10] V.Vishwanath, M.Hereld, K.Iskra, D. Kimpe, V. Morozov, M.E. Papka, R. B. Ross and K. Yoshii. Accelerating I/O Forwarding in IBM Blue Gene/P Systems. SC 2010: 1-10

[VTYR08] J. S. Vetter, V. Tipparaju, W. Yu and P. C. Roth. HPC Interconnection Networks: The Key to Exascale Computing. High Performance Computing Workshop 2008: 95-106

[WPBW09] P. M. Widener, M. Payne, P. G. Bridges, M. Wolf, H. Abbasi, S.McManus and K. Schwan. Exploiting Latent I/O Asynchrony in Petascale Science Applications. ICPP Workshops 2009: 105-112

[WUAG08] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B.Mueller, J.Small, J. Zelenka, and B. Zhou. Scalable Performance of the Panasas Parallel File System. In Proc. of the 6th USENIX Conference on File and Storage Technologies, 2008.

[WWAM11] P. Widener, M.Wolf, H. Abbasi, S. McManus, M. Payne, M. J. Barrick, J. Pulikottil, P. G. Bridges and K. Schwan. Exploiting Latent I/O Asynchrony in Petascale Science Applications. IJHPCA 25(2): 161-179 (2011)

[YuVe08] W. Yu and J. S. Vetter. ParColl: Partitioned Collective I/O on the Cray XT. ICPP 2008: 562-569

[ZhJD09] X. Zhang, S. Jiang, and K. Davis. Making Resonance a Common Case: A High-performance Implementation of Collective I/O on Parallel File Systems. In Proc. of the 23rd IEEE International Symposium on Parallel and Distributed Processing, 2009.

[ZhJi10] X. Zhang and S. Jiang. Iternterference Removal: Removing Interference of Disk Access for MPI Programs through Data Replication. In Proceedings of the 24th International Conference on Supercomputing, 2010, pp. 223–232.

[ZLMZ08] Z. Zhang, K. Lee, X. Ma and Y. Zhou. PFC: Transparent Optimization of Existing Prefetching Strategies for Multi-Level Storage Systems. ICDCS 2008: 740-751