Accelerating Big Data Infrastructure and Applications (Ongoing collaboration)

Kevin Brown*, Tianqi Xu*, Keita Iwabuchi*, Kento Sato†, Adam Moody†, Kathryn Mohror†, Nikhil Jain†, Abhinav Bhatele†, Martin Schulz†, Roger Pearce†, Maya Gokhale† and Satoshi Matuoka*
*Tokyo Institute of Technology
2-12-1 W8-33 Oo-okayama, Meguro-ku, Tokyo 152-8550, Japan
Email: brown.k.aa@m.titech.ac.jp
†Lawrence Livermore National Laboratory
7000 East Ave., Livermore, CA 94550-9234, U.S.A.

***Abstract*—High-performance computing (HPC) systems are increasingly being used for data-intensive, or "Big Data", workloads. However, since traditional HPC workloads are compute-intensive, the HPC-Big Data convergence has created many challenges with optimizing data movement and processing on modern supercomputers. Our collaborative work addresses these challenges using a three-pronged approach: (i) measuring and modeling extreme-scale I/O workloads, (ii) designing a low-latency, scalable, on-demand burst-buffer solution, and (iii) optimizing graph algorithms for processing Big Data workloads. We describe the three areas of our collaboration and report on their respective developments.**

1. Introduction

Extreme-scale supercomputing is converging with Big Data in order to cope with the proliferation of exabyte-and-beyond scale data. Large-scale data-intensive computing brings new challenges and dimensions of complexity into performance modeling and performance engineering [1]. This issue is significant for Big Data applications, where data movement can easily dominate the overall application performance.

Large-scale parallel I/O operations involve moving data across many levels of the I/O subsystem hierarchy and every layer introduces parameters/data transfer patterns that affect the performance. Therefore, understanding the performance factors of an I/O operation requires surveying the entire breadth of the system and all variables connected with the I/O subsystem.

I/O bottlenecks are possible due to contention for shared network resources and, especially, contention for shared parallel file systems (PFS) [2], [3], [4]. In contrast to the rapid growth in computational resource and network bandwidth, PFS performance growth has been modest and continues to widen the compute-I/O performance gap [5]. This creates a trend of compute-bound applications becoming I/O-bound at scale, and the effects of multi-user I/O contention and bursty I/O patterns at the PFS level are becoming their main bottlenecks [6], [7].

The handling of irregular data structures such as graphs, especially those that change over time, presents a similar problem, in that data movement becomes the dominant part of the workload. Graphs are at the core of many Big Data applications in the fields of Computer Science, Biology, Chemistry, and Social Sciences. Hence, designing algorithms and infrastructures to optimize data movement for less-structured data within large-scale graph analytics would support an entire class of essential workloads.

To address the aforementioned issues, we have been conducting collaborative work between Tokyo Institute of Technology (Tokyo Tech) and Lawrence Livermore National Laboratory (LLNL) from the perspective of I/O modeling, I/O system software, and a data-intensive analytics algorithm. The work done under this collaboration satisfies these goals by analysing the performance of data-intensive workloads on HPC systems and designing of optimized infrastructure and algorithms for Big Data processing. We are modeling the behaviour of data-intensive applications at extreme-scale to capture the variability in I/O performance due to resource failure and contention, particular when joined with other types of workloads. For parallel I/O-bound applications, we create a scalable, elastic burst-buffer solution that provides on-demand acceleration of I/O-intensive applications. And for accelerating data movement in the processing of large-scale unstructured graphs, we create a novel graph store that provides high spatial and sequential locality and supports rapidly changing dynamic graphs.

2. Measuring and Modeling I/O Performance

Parallel I/O operations traverse multiple components of the system, therefore, holistic analysis of multiple level of the system's software and hardware stack is necessary for exposing bottlenecks in data movement and storage. I/O performance analysis of production workloads further requires consideration of the operating environment and all associated factors that can affect I/O throughput at runtime.

2.1. Workload Characterization and System-wide Measurements

Measuring the performance of the data-intensive applications and their target systems is the first step towards identifying the significant performance factors that must be considered for these I/O workloads. Full-system metrics collection projects such as TOKIO [8] and SIOX [9] and works on full stack profiling such as [10] are driven by this imperative. Projects like TOKIO and SIOX produce streams of performance data points for applications and system component, which requires aggregation and correlated to arrive at useful insights. However, the challenges related to such analysis are non-trivial when dealing with multiple concurrent jobs running on a shared system [11].

We believe that system-wide analysis must be complemented with fine-grained experiments that measure and characterize the performance of data intensive workloads. The knowledge from these experiments will help to explain the performance variations seen in the analysis of system-wide metrics for production workloads. The characterization and analysis being done in our collaboration incorporates HPC, Big Data, and deep learning workloads in order to correctly understand and model the requirements of future systems.

2.2. Modeling Realistic Operating Environments

Modeling the I/O performance is necessary for designing scalable data-intensive application and responsive I/O subsystems [1]. Recent I/O modeling frameworks, such as CODES [12], have matured to include detailed models of subcomponents such as local disks and packet-level accurate networks. While existing I/O models have been useful for performance prediction and capacity planning, e.g. [13], their studies are typically based on the premise that HPC systems are static and/or that system resources are dedicated to the studied workload. Failure analysis research have confirmed otherwise; HPC systems are dynamic in the sense of the progression of failing components [14], and these systems are shared by multiple users and multiple jobs.

To accurately model the I/O performance of a system in production requires the inclusion of all of the aforementioned items: (i) the significant performance factors for the system and the workload, (ii) models for the failure rates and patterns, and (iii) the nature and effect of resource contention in the I/O subsystem. We are currently building a comprehensive model in CODES that incorporate all of these items, as shown in Figure 1, leveraging the validated network and storage models that are distributed with CODES. Simultaneously, we are collecting real-world system failure statistics and interference patterns that will be used to train our model. Our resulting model will capture the instability and performance variability of production systems. Among other things, we will use this model to explain the performance of parallel-I/O and data-intensive workloads in production as well as to design architectures for future workloads.

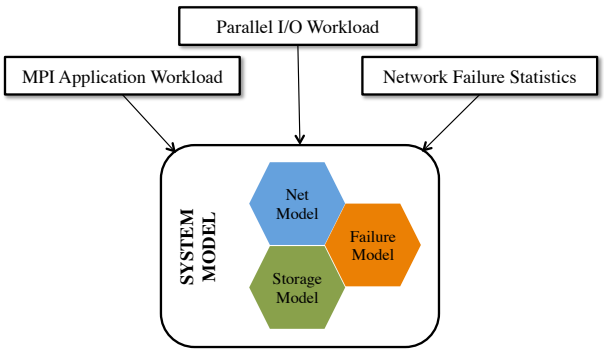


Figure 1. System Model that Incorporates Network, Storage, and Failure

3. Software-Level Burst Buffer on HPC Systems

One significant bottleneck for Big Data applications is the parallel file system (PFS) since the growth in computational performance continues to exceed the improvements in PFS performance [5]. Moreover, congestions due to PFS sharing further degrades the I/O performance of data-intensive applications [2], [3], [4].

To solve this problem, we extend the burst buffer technology [15], and propose a highly scalable I/O acceleration system called HuronFS (Hierarchical, user-level and on-demand filesystem) to accelerate parallel I/O performance. HuronFS is an extension of our previous work CloudBB (Cloud-based Burst Buffer) [16] on cloud. HuronFS builds a two-level storage hierarchy by using compute nodes to provide on-demand caching and buffering space (level-1 storage) in front of existing parallel file systems (level-2 storage). We also propose a novel fault-tolerant multi-metadata-server architecture, thereby enabling HuronFS to handle burst I/O workloads efficiently at scale.

We implement HuronFS by using CCI (Common Communication Interface) [17], a high-performance communication framework, allowing HuronFS to fully utilize the high-performance network architectures such as InfiniBand. HuronFS is built on top of the FUSE [18] filesystem, achieving portable usability since existing applications can benefit from improved I/O performance without any code modification.

3.1. Architecture

PFS need to ensure a high level consistency since multiple processes create, write, update, and read concurrently to shared files. Some PFSs use a Master-Worker model consisting of a metadata server to managing the consistency of files on the PFS's data store nodes; while other PFS such as *object store* use a Key-Value model where objects (such as chunks, files, directories, etc.) are referenced by unique hashes. In the Master-Worker model, the metadata server can easily become a bottleneck at large-scale, and in

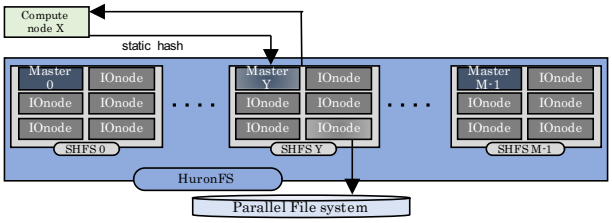


Figure 2. The overview of HuronFS

TABLE 1. EXPERIMENT ENVIRONMENT FOR HURONFS

Network	Mellanox Infiniband 4X FDR 56 Gb/sec
CPU	Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz
Memory	251 GiB

the Key-Value model, the control and configuration can be challenging.

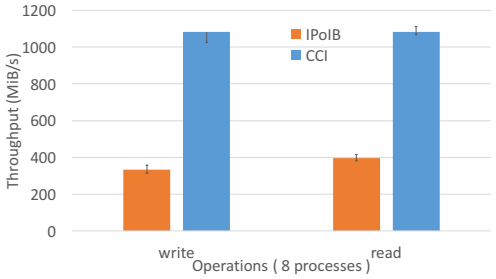
To achieve scalability as well as controllability, In CloudBB and HuronFS we propose a hybrid architecture of a Master-Worker and a Key-Value model [16], i.e., Multi-Master-Worker. Figure 2 shows the architecture of the hybrid model. The model consists of multiple sub-HuronFS (*SHFSs*), and SHFS consists of a single *Master Node* (*MN*) and multiple *IOnodes* (*IONs*). In our model, multiple MNs manage file metadata in a distributed manner so that a single MN does not become a bottleneck.

An MN is in charge of managing file metadata and controlling all the IONs belonging to the same SHFS. In order to reduce the potential for bottlenecks, I/O requests from CNs are distributed across different SHFSs by using a hash function. When a CN issues an I/O request, the CN computes a hash value from the file path of the requesting file (`file_path`). For example, given a set of SHFSs ($SHFS_0$, $SHFS_1$, ..., $SHFS_{M-1}$) and `file_path`, $SHFS_i$ is selected by the CN such that $i =$ `hash_function(file_path)` mod M. M is the total number of SHFSs. By using this architecture, we are able to hide ION from users, enabling us to dynamically change the number of IONs according to the workload.

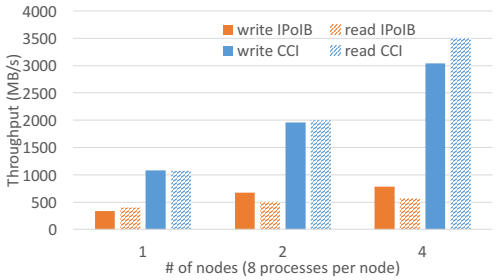
3.2. Evaluation

We evaluate the sequential I/O and metadata performance of HuronFS on a local cluster with InfiniBand network using two different communication layers: the CCI framework and the IPoIB protocol. The specification of the system is shown in Table 1. During the experiments, We use one single Master, a single I/O node, and up to four clients. For our experiments, we use the IOR [19] benchmarker running eight processes per node.

Figure 3a shows the read and write performance from single client. As we can see, we can achieve approximately 1 GiB/s from one client by using CCI, on the other hand, by using IPoIB, we can only get around 400 MiB/s. Figure 3b shows the read and write performance when we vary the number of clients using the single I/O node. As we can see,



(a) Single-client:Single-ION Performance



(b) Multi-client:Single-ION Performance

Figure 3. Throughput of HuronFS Using Different Communication Layers

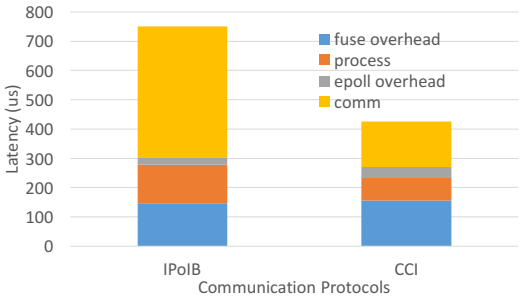


Figure 4. HuronFS Metadata Operation Performance

by increasing the number of clients, the performance scales linearly. By using 4 compute nodes, we can achieve up to 3 GiB/s with CCI and around 1 GiB/s with IPoIB.

We also measure the metadata performance by measuring the latency of `getattr`, a command that gets the attribute of a file. Figure 4 shows the latency of metadata operation on HuronFS using different communication layers. The latencies of `getattr` are approximately 400 us and 800 us with CCI and IPoIB, respectively. With CCI, we can achieve 100% performance improvement over IPoIB.

4. Locality-aware Large-scale Dynamic Graph Data Store

In many graph applications, the structure of the graph changes dynamically over time and may require real time analysis. However, most prior work for processing large graphs on HPC platforms has not focused on dynamic

graphs, rather static only. To address this issue, we have been developing data structures and the infrastructure necessary to support dynamic graph analysis at large scale on distributed HPC platforms, including next generation supercomputers which have locally attached NVRAM (non-volatile random-access memory).

4.1. DegAwareRHH

We have developed DegAwareRHH [20], a novel high-performance large-scale dynamic graph data store that leverages a linear probing and open addressing compact hash table that exhibits high spatial and sequential locality. DegAwareRHH is *degree aware*, and uses separated compact data structures for low-degree vertices to reduce their storage and search overheads especially on NVRAM. In addition, we extend DegAwareRHH for distributed-memory using an *asynchronous visitor queue abstraction* [21], [22] aiming for localizing remote communication in the area where graph update occurred.

4.2. Performance Evaluations on Dynamic Graph Processing Workloads

We use the Catalyst cluster at LLNL; each node has 12-core Intel(R) Xeon(R) processors (2 sockets) and 128 GB of DRAM.

4.2.1. Dynamic Graph Construction. Figure 5 shows the graph construction (unique edge insertions and deletions) performance of DegAwareRHH and STINGER [23], a state-of-art dynamic graph processing framework, for performance comparison. We use a synthetic graph that has 1B edges and 5% additional edge deletes. As can be seen, DegAwareRHH outperforms STINGER by 212.2 times.

4.2.2. Dynamic Graph Coloring. Next, we evaluate the performance of a dynamic graph analytics algorithm on DegAwareRHH. We run the edge-centric massive-scale dynamic graph-colouring algorithm developed by Sallinen et al. [24].

We use PowerGraph [25] for performance comparison. We run their static version of graph colouring implementation, whose execution time does not including graph construction time.

The execution time of the graph colouring algorithm on SK2005 [26] graph are shown in Figure 6. Surprising, PowerGraph performance degrades as the number of compute nodes increases. In contrast, DegAwareRHH scales well and outperforms PowerGraph by 9.7x at 64 compute nodes.

5. Summary

Our efforts in measuring and modeling extreme-scale I/O workloads target realistic operating environment, considering both component failures as well as interference from non-I/O workloads over shared resources. For optimizing data-intensive workloads, we have developments

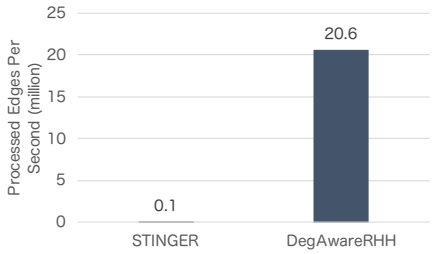


Figure 5. Dynamic Graph Construction Performance (in-core, single-node).

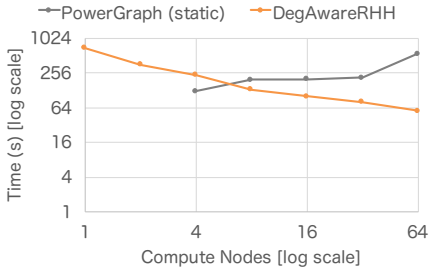


Figure 6. Run-time for the Dynamic Graph Coloring (in-core)

of HuronFS: a scalable, low-latency user-space burst-buffer solution for HPC systems; and DegAwareRHH: a large scale dynamic graph store for accelarating graph processing in Big Data applications.

Acknowledgments

This research was supported by JST, CREST (Research Area: Advanced Core Technologies for Big Data Integration).

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-727471).

References

[1] S. Snyder, P. Carns, R. Latham, M. Mubarak, R. Ross, C. Carothers, B. Behzad, H. V. T. Luu, S. Byna, and Prabhat, "Techniques for modeling large-scale hpc i/o workloads," in *Proceedings of the 6th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computing Systems*, ser. PMBS '15, 2015, pp. 5:1–5:11.

[2] A. Kougkas, M. Dorier, R. Latham, R. Ross, and X. H. Sun, "Leveraging burst buffer coordination to prevent i/o interference," in *2016 IEEE 12th International Conference on e-Science (e-Science)*, Oct 2016, pp. 371–380.

[3] B. Xie, J. Chase, D. Dillow, O. Drokin, S. Klasky, S. Oral, and N. Podhorszki, "Characterizing output bottlenecks in a supercomputer," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[4] J. Lofstead and R. Ross, "Insights for exascale io apis from building a petascale io api," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 87:1–87:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503238

[5] V. Vishwanath, M. Hereld, V. Morozov, and M. E. Papka, "Topology-aware data movement and staging for i/o acceleration on blue gene/p supercomputing systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 19:1–19:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063409

[6] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, S. Gupta, D. T. S. S. Vazhkudai, J. H. Rogers, D. Dillow, G. M. Shipman, and A. S. Bland, "Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems," in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2014, pp. 217–228.

[7] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross, "Understanding and improving computational science storage access through continuous characterization," *Transactions on Storage (TOS*, vol. 7, no. 3, pp. 8:1–8:26, Oct. 2011.

[8] N. E. R. S. C. C. (NERSC), "Tokio: Total knowledge of i/o," http://www.nersc.gov/research-and-development/tokio/, 2016.

[9] J. M. Kunkel, M. Zimmer, N. Hübbe, A. Aguilera, H. Mickler, X. Wang, A. Chut, T. Bönisch, J. Lüttgau, R. Michel, and J. Weging, *The SIOX Architecture – Coupling Automatic Monitoring and Optimization of Parallel I/O*. Springer International Publishing, pp. 245–260.

[10] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular hpc i/o characterization with darshan," in *2016 5th Workshop on Extreme-Scale Programming Tools (ESPT)*, Nov 2016, pp. 9–17.

[11] O. DeMasi, T. Samak, and D. H. Bailey, "Identifying hpc codes via performance logs and machine learning," in *Proceedings of the First Workshop on Changing Landscapes in HPC Security*, ser. CLHS '13, 2013, pp. 23–30.

[12] J. Cope, N. Liu, S. Lang, P. Carns, C. Carothers, and R. Ross, "Codes: Enabling co-design of multi-layer exascale storage architectures," in *Proceedings of the Workshop on Emerging Supercomputing Technologies 2011*, 2011.

[13] S. El Sayed, M. Bolten, and D. Pleiter, "Using file system counters in modelling parallel i/o architectures," *SIGOPS Oper. Syst. Rev.*, vol. 50, no. 3, pp. 37–46, Jan. 2017.

[14] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer, "Lessons learned from the analysis of system failures at petascale: The case of blue waters," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 610–621.

[15] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, April 2012, pp. 1–11.

[16] T. Xu, K. Sato, and S. Matsuoka, "Cloudbb: Scalable i/o accelerator for shared cloud storage," in *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, Dec 2016, pp. 509–518.

[17] O. R. N. Laboratory, "Common communication interface," https://www.olcf.ornl.gov/center-projects/common-communication-interface/.

[18] "FUSE," http://fuse.sourceforge.net/.

[19] "IOR HPC benchmark," http://sourceforge.net/projects/ior-sio/, LLNL.

[20] K. Iwabuchi, S. Sallinen, R. Pearce, B. V. Essen, M. Gokhale, and S. Matsuoka, "Towards a distributed large-scale dynamic graph data store," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 892–901.

[21] R. Pearce, M. Gokhale, and N. M. Amato, "Scaling Techniques for Massive Scale-Free Graphs in Distributed (External) Memory," in *2013 IEEE International Parallel & Distributed Processing Symposium (IPDPS2013)*, 2013, pp. 825–836.

[22] ——, "Faster parallel traversal of scale free graphs at extreme scale with vertex delegates," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 549–559.

[23] D. Ediger, R. McColl, J. Riedy, and D. Bader, "Stinger: High performance data structure for streaming graphs," in *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, Sept 2012, pp. 1–5.

[24] S. Sallinen, K. Iwabuchi, S. Poudel, M. Gokhale, M. Ripeanu, and R. Pearce, "Graph coloring as a challenge problem for dynamic graph processing on distributed systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2016.

[25] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 17–30.

[26] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Software: Practice and Experience*, vol. 34, no. 8, pp. 711–726, 2004.