An I/O Load Balancing Framework for Large-scale Applications (BPIO 2.0)

Sarah Neuwirth*, Feiyi Wang†, Sarp Oral† and Ulrich Bruening*
*Institute of Computer Engineering, University of Heidelberg, Germany
{sarah.neuwirth, ulrich.bruening}@ziti.uni-heidelberg.de
†National Center for Computational Sciences, Oak Ridge National Laboratory, USA
{fwang2, oralhs}@ornl.gov

***Abstract*—Designed for capacity and capability, HPC storage systems are inherently complex and shared among multiple, concurrent jobs competing for resources. The lack of centralized coordination and control often render the end-to-end I/O paths vulnerable to load imbalance and contention. With the emergence of data-intensive HPC applications, storage systems are further contended for performance and scalability. BPIO is a topology-aware, load balancing library for mitigating resource contention. This work introduces BPIO 2.0, a dynamic, shared load balancing framework based on BPIO. It transparently intercepts file creation calls during runtime to balance the workload over all available storage targets. The utilization of BPIO 2.0 requires no source code modification and is independent from any I/O middleware. We demonstrate the effectiveness of our framework on the Titan system with a synthetic benchmark in a noisy production environment.**

***Keywords*-Parallel File System, High Performance Computing, Load Balancing, Performance Evaluation, Performance Model**

I. Introduction

Large-scale scientific simulations can be both compute and I/O intensive as they stress the capability of file and storage systems by producing large amounts of data in a bursty pattern. Parallel file systems distribute the workload over multiple I/O paths and components to satisfy the I/O requirements in terms of performance, capacity, and scalability. In high performance computing (HPC) storage deployments, file systems are often shared among multiple, concurrently running applications with different workloads. This often results in file system and network contention. The observed I/O bandwidth at the application-level can be much lower than the theoretical peak bandwidth of the underlying storage system.

We propose an I/O balancing framework, called BPIO 2.0, that takes advantage of the balanced placement I/O (BPIO) [1] library for large-scale scientific applications. The library addresses the resource contention problem by implementing a topology-aware, balanced placement strategy. BPIO 2.0 requires no modification or recompilation of the application. We conclude with an initial performance evaluation on the Titan [2] system.

II. Motivation and Background

We first present on overview of Titan and Spider II to provide an understanding of our evaluation platform. Afterwards, Aequilibro, an I/O middleware that indirectly motivates BPIO 2.0, is introduced.

Titan is a Cray XK7 system with 18,688 compute nodes. Its parallel file system is Spider II [3], which is based on the Lustre technology [4] and one of the world's fastest and largest POSIX-compliant parallel file systems. It is configured and deployed as two independent, non-overlapping file systems, each with 144 Lustre Object Storage Servers (OSSs) and 1,008 Lustre Object Storage Targets (OSTs).

Resource contention has a disadvantageous impact on the performance and scalability of HPC storage systems such as Spider II. Aequilibro [5], [6], an ADIOS-based middleware, attempts to resolve the load imbalance by utilizing the BPIO library. BPIO employs a placement strategy that provides a binding between an I/O client (compute node or MPI process) and a storage target that aims to resolve application-level contention. The BPIO algorithm uses a placement cost function that takes a weighted average of how frequently different file system resources have been used by previous I/O requests issued by the same application. ADIOS [7] is a flexible middleware that provides a simple I/O application programming interface (API) with portable, fast, scalable, metadata-rich output. An initial performance evaluation has been carried out on the Titan system. ADIOS-enabled applications can transparently take advantage of BPIO with similar performance results as presented by Wang et al. [1]. Drawbacks are that the usage is limited to specific ADIOS transport methods, namely POSIX and MPI_AGGREGATE, and the overhead added by ADIOS limits the performance improvement compared to a direct BPIO integration into an application. By designing a preloadable BPIO-based framework, we translate the benefits of BPIO transparently into an application without changes to the application source code or recompilation.

III. Balanced Placement I/O Framework

BPIO 2.0 is designed to work with parallel file systems and does not require any modifications to application or I/O library source code. It utilizes the BPIO library for intelligently allocating I/O paths for a parallel file system. Figure 1 illustrates the runtime environment. BPIO 2.0 uses function interposition, similar to Darshan [8] and Recorder [9], to prioritize itself over standard functions. Once BPIO 2.0 is specified as the preloading library, it intercepts POSIX I/O and MPI-IO file

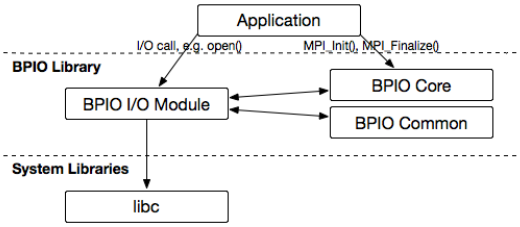

Fig. 1. BPIO 2.0 runtime environment.

creation calls and reroutes them to the BPIO library. The original POSIX or MPI-IO function is called after invoking the load balancing. This balancing technique is transparent to the user because alterations are made without modifications to the application source code. It offers per job and end-to-end I/O performance improvement in the most transparent way.

IV. Experimental Evaluation

The Titan system with its Spider II file system is used for early evaluation. The IOR benchmark [10] is used to evaluated the performance improvement and bandwidth results for POSIX I/O and MPI-IO. All experiments are conducted in a busy production environment. No tests are run during the quiet maintenance mode. The results show that performance gains can be achieved in an active production environment.

Figure 2 displays the IOR results for runs scaling up to 2,048 nodes for POSIX I/O and MPI-IO. We compare the bandwidth performance of IOR Default and IOR BPIO 2.0. The results are presented as the performance improvement in percentage and calculated as follows: Performance Improvement = $100 * (BW_{BPIO}/BW_{Default} - 1)$. For large-scale runs, BPIO 2.0 provides a consistent improvement greater than 20%. Figure 3 displays the average bandwidth results for IOR Default and IOR BPIO 2.0 for POSIX I/O and MPI-IO. For small-scale runs with less than 128 nodes, the effectiveness of BPIO 2.0 is relatively small. The same observations were made with the original BPIO library. As we scale up in terms of I/O processes and allocated nodes, POSIX and MPI-IO both benefit from BPIO 2.0. For example, large-scale runs with a 2,048 node allocation provide an average throughput of 135.6 GB/s for POSIX using BPIO 2.0. Similar trends can be observed for MPI-IO.

V. Conclusions

This poster presents the first steps towards a full-featured load balancing framework that resolves application-level I/O contention for different I/O interfaces, including POSIX I/O and MPI-IO. BPIO 2.0 is built as a dynamic, shared library so it does not require any modification or recompilation of the source code. Early evaluation has shown that it provides similar performance improvement and bandwidth performance as the direct integration of BPIO into a large-scale application. Future work will include the extensive performance evaluation with different scientific HPC workloads, the support of single shared files and the full support of HDF5.

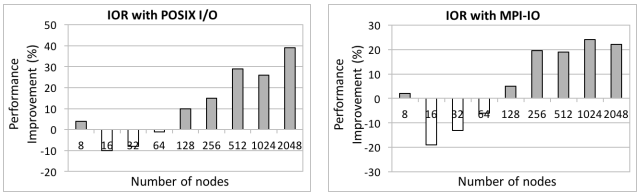

Fig. 2. Performance improvement for IOR with BPIO 2.0.

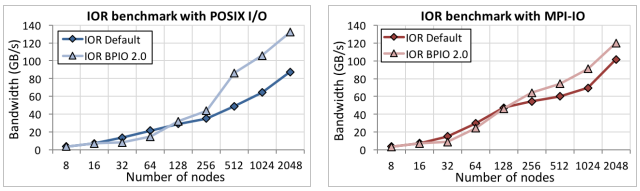

Fig. 3. Bandwidth performance IOR Default and IOR with BPIO 2.0.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

References

[1] F. Wang, S. Oral, S. Gupta, D. Tiwari, and S. Vazhkudai, "Improving Large-scale Storage System Performance via Topology-aware and Balanced Data Placement," in *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014*, December 2014, pp. 656–663.

[2] A. S. Bland, J. C. Wells, O. E. Messer, O. R. Hernandez, and J. H. Rogers, "Titan: Early Experience with the Cray XK6 at Oak Ridge National Laboratory," in *Proceedings of Cray User Group Conference (CUG 2012)*, 2012.

[3] S. Oral, D. A. Dillow, D. Fuller, J. Hill, D. Leverman, S. S. Vazhkudai, F. Wang, Y. Kim, J. Rogers, J. Simmons *et al.*, "OLCF's 1 TB/s, next-generation lustre file system," in *Proceedings of Cray User Group Conference (CUG 2013)*, 2013.

[4] P. J. Braam *et al.*, "The Lustre storage architecture," 2004.

[5] S. Neuwirth, S. Oral, F. Wang, Q. Liu, and S. Vazhkudai, "Improving Large-scale Application Performance with ADIOS and BPIO," Poster at *Smoky Mountains Computational Sciences and Engineering Conference*, 2015.

[6] S. Neuwirth, F. Wang, S. Oral, S. Vazhkudai, J. Rogers, and U. Bruening, "Using Balanced Data Placement to Address I/O Contention in Production Environments," in *28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2016)*, 2016.

[7] J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, and 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*. ACM, 2008, pp. 15–24.

[8] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[9] H. Luu, B. Behzad, R. Aydt, and M. Winslett, "A multi-level approach for understanding I/O activity in HPC applications," in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2013, pp. 1–5.

[10] LLNL, "The Interleaved Or Random (IOR) Benchmark," https://github.com/chaos/ior.