Using Balanced Data Placement to Address I/O Contention in Production Environments

Sarah Neuwirth*, Feiyi Wang†, Sarp Oral†, Sudharshan Vazhkudai†, James H. Rogers† 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, vazhkudaiss, jrogers}@ornl.gov

***Abstract*—Designed for capacity and capability, HPC I/O systems are inherently complex and shared among multiple, concurrent jobs competing for resources. 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.**

This paper proposes to unify two key approaches to tackle the imbalanced use of I/O resources and to achieve an end-to-end I/O performance improvement in the most transparent way. First, it utilizes a topology-aware, Balanced Placement I/O method (BPIO) for mitigating resource contention. Second, it takes advantage of the platform-neutral ADIOS middleware, which provides a flexible I/O mechanism for scientific applications. By integrating BPIO with ADIOS, referred to as Aequilibro, we obtain an end-to-end and per job I/O performance improvement for ADIOS-enabled HPC applications without requiring any code changes. Aequilibro can be applied to almost any HPC platform and is mostly suitable for systems that lack a centralized file system resource manager. We demonstrate the effectiveness of our integration on the Titan system at the Oak Ridge National Laboratory. Our experiments with a synthetic benchmark and real-world HPC workload show that, even in a noisy production environment, Aequilibro can improve large-scale application performance significantly.

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

I. Introduction

Large-scale scientific applications stress the capability of file and storage systems by producing large amounts of data in a bursty pattern. With the advent of big data, it is expected that future large-scale applications will generate even more data. 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 large-scale high performance computing (HPC) deployments, file systems are often shared among applications concurrently running on a single system and sometimes among applications running on multiple systems (e.g., a center-wide file system). 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. Bursty writes, e.g. checkpointing data, can cause resource contention leading to hotspots which are detrimental to parallel application performance. Hotspots lead to variations in completion times across processes, and therefore, to a blocking behavior and wasted computational capacity.

The balanced placement I/O (BPIO) library [1] addresses the resource contention problem by providing a topology-aware, balanced placement strategy that is based on a tunable, weighted cost function of available system components. It balances the workload by keeping track of the usage frequency of all available file system components to improve the overall I/O performance. BPIO is available as an easy-to-use, user-space library. In order to increase the application adaptability and transition, this paper proposes Aequilibro, the transparent merging of BPIO and ADIOS [2], [3]. ADIOS is a flexible middleware that provides an easy mechanism to describe data elements, types, and the I/O method. With this unification, ADIOS-enabled applications can effortlessly take advantage of BPIO's performance benefits without any further modifications. The implementation and evaluation are carried out on the Titan system [4] and Spider II [5]. By repeating small-, medium- and large-scale runs over an extended period of time in a production environment, we demonstrate that the integration is effective in improving application I/O independently from any node allocation and I/O interferences. Aequilibro combines the optimization done at the interconnect network level and the load balancing done at the file system level by BPIO with the benefits of an I/O framework such as ADIOS to provide near optimal I/O methods for the user. Aequilibro can be used with a variety of available I/O solutions, including POSIX I/O, MPI-IO [6], HDF5 [7], netCDF, and SILO [8].

The contributions of this paper are two-fold. First, we present Aequilibro, an unification of the topology-aware load balancing library BPIO and the ADIOS middleware library for two popular I/O methods, POSIX I/O and MPI-IO. Second, we present an evaluation of Aequilibro based on an analysis methodology that utilizes the Interleaved Or Random (IOR) synthetic benchmark [9] and Genarray, an HPC workload simulator. The performance is evaluated at different scales to show how Aequilibro mitigates resource contention and improves scientific application performance

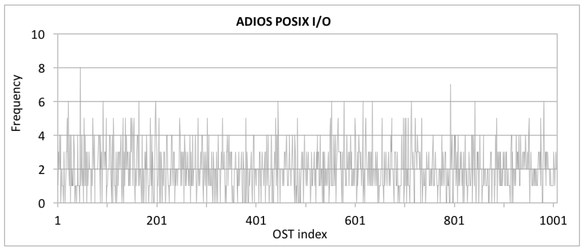

Figure 1. ADIOS POSIX I/O with default Lustre data placement.

significantly in busy environments. The paper is organized as follows. Section II provides the background and motivation. Section III describes Aequilibro. In Section IV, the analysis methodology and performance evaluation are discussed. Section V presents related work. The paper concludes in section VI.

II. Motivation and Background

Resource contention adversely impacts the performance and scalability in high-performance systems. As described by Wang et al. [1], there exists a significant variation in usage across system resource components. The same behavior can be observed for applications using ADIOS for I/O on Titan. Titan is a Cray XK7 system with 18,688 compute nodes. Its parallel file system is Spider II [5], which is based on the Lustre technology [10] and one of the world's fastest and largest POSIX-compliant parallel file systems. It is configured and deployed as two independent and non-overlapping file systems, each with 144 Lustre Object Storage Servers (OSSs) and 1,008 Lustre Object Storage Targets (OSTs). Figure 1 displays a representative example OST usage distribution out of 30 scaled runs on Titan for ADIOS with the default Lustre data placement strategy for a 2,016 node allocation. It can be observed that there is a significant variation in usage across storage targets. Parallel file systems lack advanced routing and workload balancing mechanism. Consequently, imbalanced resource utilization increases contention.

There are different possible approaches to address the resource contention problem. One is the improvement of the Lustre OST allocation scheme. This is a Lustre-specific solution, and therefore, disregarded. Another approach is the deployment of a centralized, system-wide I/O coordination and control mechanism. E.g., Fastpass [11] is a data center network framework that aims for high utilization with zero queuing. For large-scale HPC systems, this approach is not feasible. Multiple applications are running currently, with a variety of different I/O patterns and workloads. A system-wide I/O mechanism would need to stall applications to coordinate the I/O requests in a balanced manner, leading to a tremendous communication overhead. Instead of improving the performance, it likely would lead to a sub-optimal utilization of the available computational resources.

The third approach is to balance the I/O workload on an end-to-end and per job basis as done by the balanced placement I/O library (BPIO) [1]. BPIO tackles the problem by intelligently allocating I/O paths for a parallel file system. The library 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 I/O contention. It utilizes the Fine-Grained Routing (FGR) congestion avoidance method [12]. FGR organizes I/O paths to minimize end-to-end hop counts and congestion. This is done by pairing clients with their closest possible, and in the case of Titan, optimal LNET router. The algorithm behind the BPIO library 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. The most general case is defined by

$$C = w_1 R_1 + w_2 R_2 + ... + w_n R_n \qquad (1)$$

where C is the cost of an I/O path, R_i is a resource component, and w_i is the weight factor with $\sum_{i=1}^{n} w_i = 1$. For Lustre, possible resource components are logical I/O routers (i.e., LNET), or actual file system and networking resources (i.e., Lustre I/O routers, OSSs, and OSTs). The algorithm loops over all reachable storage targets to choose one with the lowest placement cost per compute node. This is repeated for all I/O clients allotted to the application once before the application enters the I/O write phase. An initial performance evaluation of the library was performed on the Titan supercomputer [4].

ADIOS [2], [3] is a flexible middleware that provides a simple I/O application programming interface (API) with portable, fast, scalable, metadata-rich output. One of the salient features is that I/O methods can be changed by modifying the configuration file without the need to modify or recompile the application code. ADIOS has demonstrated impressive I/O performance results on leadership class machines and clusters. There are two key ideas. First, users do not need to be aware of the low-level layout and organization of data. Second, application developers should not be burdened with optimizations for each platform they use. It is capable of I/O aggregation on behalf of the application to increase the I/O performance and scalability. However, it does not provide an I/O balancing mechanism. Aequilibro provides ADIOS-users the ability to take full advantage of the balanced data placement strategy, and therefore, to tackle the resource contention problem on file system level.

III. Aequilibro Software Architecture and Implementation

Aequilibro is implemented in the context of ADIOS. Specifically, it is integrated as on optional feature in the file creation and write phase for selected ADIOS transport methods.

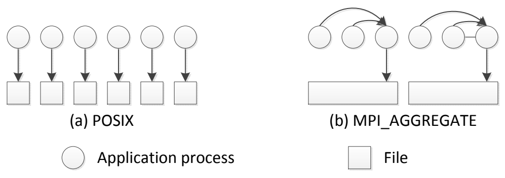

Figure 2. ADIOS transport methods.

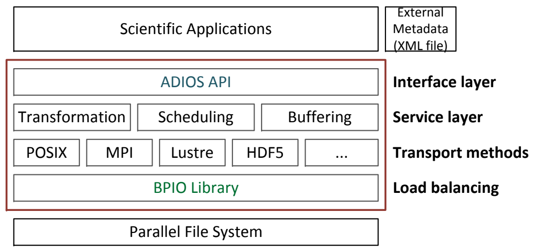

Figure 3. Aequilibro software stack with example XML-file.

A. ADIOS Transport Methods

ADIOS provides a mechanism to externally describe an application's I/O requirements using an XML-based configuration file. This results in a runtime selectable technique for performing basic ADIOS operations, e.g. *adios_open()* and *adios_write()*. Depending on the selected transport method, a different number of files is produced. *POSIX* and *MPI_AGGREGATE* are two examples for transport methods.

POSIX takes advantage of the concurrency of parallel file systems. Each writing process is responsible for writing data to its own output file, called file-per-process strategy. This is illustrated by Figure 2 (a). One process is responsible for an index file. The transport method results in an index file along with a subdirectory containing all of the files written individually by the application processes. *MPI_AGGREGATE* is a sophisticated, MPI-IO-based technique derived from the *MPI* and *MPI_LUSTRE* transport methods. Instead of using the default data aggregation, it provides an optimized method that aggregates data from multiple MPI processes into large chunks before writing them out to the file system, as shown in Figure 2 (b). A subset of application processes acts as an aggregator for a subset of peers. The number of aggregators (writers) and the number of OSTs can be specified by the user. The method creates a collection of seperate files, with one file corresponding to each of the aggregators. The user is able to define Lustre-specific striping information.

B. Aequilibro

Aequilibro, our proposed integration solution, resolves the resource contention problem. By integrating BPIO into ADIOS, we are translating BPIO's end-to-end and per job I/O performance improvements directly and transparently to the user applications. ADIOS' software architecture consists of several layers. The best way to integrate BPIO into ADIOS is to select the transport methods that are suited best to benefit from a balanced data placement and implement the integration directly into those methods. The BPIO library acts as a shim layer between the transport methods and the parallel file system, as depicted by Figure 3.

The load balancing library is integrated in the *POSIX* and *MPI_AGGREGATE* transport methods offered by the ADIOS framework. There are two reasons for this decision. We believe that those two I/O techniques are widely used on current HPC systems. As pointed out by a recent study of HPC systems [13], between 50% and 95% of jobs use the POSIX I/O library. The remaining jobs use MPI-IO directly or libraries built atop MPI-IO. The second reason for selecting *POSIX* and *MPI_AGGREGATE* is related to our test system. The current implementation of the BPIO library is deployed on Titan's Spider II file system. Spider II is based on Lustre 2.4 and therefore, inherits its limitations. Currently, BPIO only supports the file-per-process I/O strategy. Lustre 2.4 lacks the ability to provide fine-grained control of object placement. Lustre 2.7 and beyond offer a new feature [14] that provides the user the ability to specify a list of OSTs to be used for a single shared file. Future Aequilibro versions will take advantage of this feature.

During the initialization of the ADIOS framework with *adios_init()*, the BPIO library is configured with system-specific mapping files generated by the fine-grained routing congestion avoidance method. In addition, a list of allotted I/O clients is created. Algorithm 1 describes the details of initializing Aequilibro. At the entry of an I/O write phase, the BPIO library is invoked with the list of allotted I/O clients. Depending on the transport method, this takes place in the *adios_open()* or *adios_groupsize()* call. The BPIO library returns a list with an I/O client to OST assignment and uses the information to create the files, as described in Algorithm 2. For POSIX I/O, a file is created with Lustre's *llapi* library. The stripe size, stripe count, and start OST of a file are set via the logical object volume (LOV) layer and stored in *lov_user_md*. LOV handles client access to OSTs. With IOCTL, the striping information is applied. For MPI-IO, the file creation is divided in two steps. A file is pre-created as described for the POSIX case and afterwards, it is opened with *MPI_File_open()*.

Algorithm 1 Initialization

```
1: /* Global parameters */
2: $nids* \leftarrow$ NULL, $osts* \leftarrow$ NULL, $my\_nid \leftarrow 0$
3: Initialize ADIOS transport method
4: Initialize BPIO library with FGR mapping files
5: $size \leftarrow$ Number of participating processes
6: $my\_nid \leftarrow$ MPI_Get_processor_name(...)
7: Allocate arrays $nids$ and $osts$ for NID to OST binding
8: $nids \leftarrow$ List of unique node identifiers for each process
```

Algorithm 2 NID/OST Binding and File Creation

```
1: Invoke BPIO library with $nids$ for NID/OST binding
2: $osts \leftarrow$ List of I/O client (NID) to OST assignments
3: struct lov_user_md $opts$ = { 0 };
4: /* OST_OFFSET describes the number of OSTs */
5: if ($osts[my\_rank] \geq$ OST_OFFSET) then
6:     $my\_ost = osts[my\_rank] -$ OST_OFFSET
7: else
8:     $my\_ost = osts[my\_rank]$
9: end if
10: $opts.lmm\_stripe\_offset \leftarrow my\_ost$
11: Open file with O_LOV_DELAY_CREATE flag
12: Set striping information for file with IOCTL
```

The BPIO library is designed in a way that adds minimal overhead. The placement algorithm is invoked only once for every I/O write phase. The internal data structures keep the storage overhead small. As described by Wang et al. [1], the algorithm that is used by BPIO is sensitive to the size of the possible resources and routing paths. When the number of I/O requests is small and tightly packed in close proximity, a set of less optimal OSTs might be used.

IV. Data Collection and Analysis

In this section, we first describe the evaluation methodology. Then, we present the experimental setup, and analyze our evaluation results both from a synthetic benchmark tool and a real-world HPC workload.

A. Methodology

To evaluate the I/O performance, we use the Interleaved Or Random (IOR) benchmark and Genarray, a test code that emulates I/O patterns similar to S3D [15].

1) IOR Benchmark: We adopt the well-known IOR [9] synthetic benchmark tool to assess the strength and weakness of Aequilibro. IOR provides a flexible way of measuring I/O performance under different read/write sizes, concurrencies, file formats, and file layout strategies. It supports different I/O interfaces ranging from traditional POSIX to advanced parallel I/O interfaces like MPI-IO and differentiates parallel I/O strategies between file-per-process and single-shared-file approaches. Shan et al. [16] demonstrated that IOR can be used to characterize and predict the I/O performance on HPC systems at scale. We utilize IOR to evaluate the direct use of the BPIO library in comparison to Aequilibro handling I/O for an application. The performance evaluation is divided into two steps. First, the BPIO library is directly integrated into the IOR benchmark. Before creating a file, the BPIO library is used to determine the compute node to OST assignment. The second step is to use ADIOS for the I/O handling in the IOR benchmark tool. All I/O interface calls are replaced by ADIOS API calls. Essentially, we use IOR as a workload generator to drive the ADIOS framework. Table I displays the IOR benchmark variants that are used for the evaluation.

Table I
IOR benchmark variants.

Index	Variant	Description
(I)	Default	The original IOR benchmark without any modification.
(II)	BPIO	A modified version of the IOR tool that utilizes the BPIO library for balanced data placement.
(III)	ADIOS	An IOR benchmark where all I/O calls are replaced with the ADIOS API for I/O handling.
(IV)	Aequilibro	Same code base as IOR ADIOS, but utilizes the BPIO library for balanced data placement.

We compare the performance of *IOR Default* without any modifications with the IOR tool that utilizes the BPIO library for balanced data placement. Furthermore, an evaluation of IOR using the ADIOS API for I/O handling in comparison to IOR that uses Aequilibro is performed. We believe that integrating BPIO with IOR provides a good way to verify the results previously obtained with the placement I/O benchmarking tool (PIO) [1]. Another advantage is that IOR can be used to get initial performance results for MPI-IO. Using ADIOS with IOR provides an easy way to stress the file system while handling file I/O with the ADIOS API. A side benefit is that Aequilibro can be tested without any additional code modification. IOR just needs to be recompiled against the Aequilibro framework. In addition, the results obtained with IOR Default and IOR BPIO can be used as a reference to evaluate the performance of ADIOS. The metric of interest is the end-to-end I/O *performance improvement* gained by using BPIO. It is provided in percentage and calculated as follows:

$$\text{Performance Improvement} = 100 * \left(\frac{\text{BW}_{BPIO}}{\text{BW}_{default}} - 1 \right) \quad (2)$$

2) I/O Interference: Large-scale HPC systems waste a significant amount of computing capacity because of I/O interferences caused by multiple running applications concurrently accessing a shared parallel file system and its storage resources. Lofstead et al. [17] introduced internal and external interference to characterize the variability in I/O performance. We use their definition to evaluate the performance of Aequilibro in terms of I/O interference.

Internal interference: When too many write processes from one specific application try to write to a single storage target at the same time, internal interference can occur. Write caches are exceeded which leads to a blocking behavior of the application until the buffers are cleared. We utilize the IOR benchmark to evaluate the internal interference by writing data of different sizes to OSTs while scaling up the number of nodes/writers. The IOR benchmark is configured to use 1,008 OSTs and POSIX I/O.

External interference: Even if an application tries to evenly use all available storage resources, external interference can still occur. This is caused by the fact that a parallel file system is a shared resource where access is shared between all compute nodes and running applications. To demonstrate the I/O performance variability, we perform hourly IOR tests with an 1,008 node allocation, one process per node (one writer per node), and POSIX I/O with the file-per-process I/O strategy. For better characterization, we use the *imbalance factor* of an I/O action as defined by Lofstead et al. [17]. It describes the ratio of the slowest ($wtime_{max}$) vs. the fastest write ($wtime_{min}$) times across all writers:

$$\text{Imbalance factor} = \frac{wtime_{max}}{wtime_{min}} \tag{3}$$

The imbalance factor reflects the impact of the slowest writer on the overall performance. Therefore, the factor can be utilized to characterize the imbalance of an application.

3) HPC Workload: S3D is a combustion code simulation that is widely used on HPC systems. It generates a large amount of I/O requests. Verifying the I/O performance improvement of S3D with Aequilibro provides us with a good indicator of the impact on other large-scale applications. Genarray is an S3D workload simulator provided by ADIOS. In Genarray, three dimensions of a global array are partitioned among MPI processes along X-Y-Z dimensions in the same block-block-block fashion. Each process writes an N^3 partition. The size of each data element is 4 bytes, leading to the total data size of $N^3 * P * 4$ bytes, where P is the number of processes. Therefore, I/O methods can be exchanged by simply modifying the corresponding XML-file. One key difference between the IOR benchmark tool and Genarray is that by default Genarray utilizes all cores present on a compute node. This improves the computational efficiency of the simulation. On the other hand, Genarray generates pressure on single storage targets, because each compute node hosts its own operating system with a single mount point per file system.

4) Selecting IOR and ADIOS Parameters: In order to accurately model an HPC workload behavior, the benchmark parameters must be aligned with the desired workload: *API*, *FilePerProc*, *WriteFile*, *NumTasks*, *BlockSize*, and *TransferSize*. On Titan, the memory size is 32 GB/node, 2 GB/processor. The IOR benchmark is run with a *BlockSize* of 4 GB to eliminate cache effects, a *TransferSize* of 1 MB and the *FilePerProc* mode. For POSIX I/O, the *fsync* and *useO_DIRECT* are used. O_DIRECT bypasses I/O and file system buffers. For MPI-IO, the same effect can be achieved by enabling the *direct_io* MPI-IO optimization hint. For each run, the node allocation and OST distribution is tracked. For Lustre-specific settings, each file is created with a *StripeSize* of 1 MB and a *StripeCount* of 1. The *StripeSize* needs to be aligned with the *TransferSize* in order to get the best performance. *StripeCount* specifies the number of OSTs where the data is striped across while *StripeSize* defines the size of one stripe. The default Lustre stripe count is

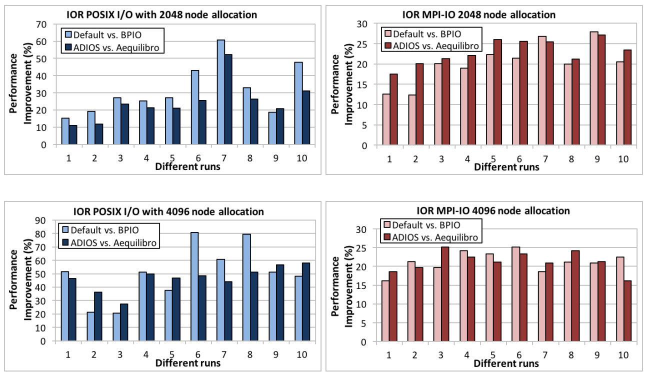

Figure 4. Performance improvements for IOR large-scale runs.

4. ADIOS is a meta-data rich middleware. To get a better insight on the raw I/O performance, the meta-data file is disabled with *have_metadata_file*=0. The number of OSTs has to be specified for *MPI_AGGREGATE*.

B. Experimental Setup

All tests were performed on the Titan supercomputer. In order to get representative results, two major issues are addressed. First, 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. Second, a broad set of compute nodes are used instead of just a certain subset of nodes. This demonstrates that independently from any specific node set on Titan, an application can readily benefit from the presented balancing framework. The application level placement scheduler (ALPS) on Titan returns a node allocation list where nodes tend to be logically close to each other. There are two attempts to get a higher node coverage. The first one is to submit scaling tests one after another independently, in the hope that a different set of compute nodes is covered with every run. The second attempt is to submit scaling runs in parallel to occupy a larger set of nodes. Both approaches are used to get a broader coverage. More than 30 scaled runs were obtained for IOR variants (I) to (IV) (see Table I), with each run ranging from 8 to 4,096 nodes. For each node allocation, three iterations are performed to obtain the average badwidth results. All of our experiments are conducted in a noisy, active production environment. Therefore performance numbers may not always be conclusive. To cope with this issue to draw consistent observation, multiple tests are performed with at least three repetitions per run. Iterations within the same run get the same node allocation. Each iteration executes IOR variants (I) to (IV). This is essential in order to average out the variance across the same set of allocated nodes. In addition, a large set of Titan compute nodes is covered throughout these tests. For the internal interference, we run

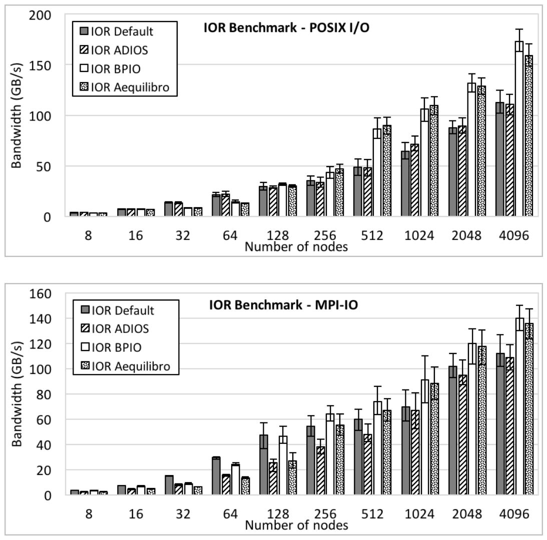

Figure 5. IOR bandwidth performance for IOR (I) to (IV) with error bars.

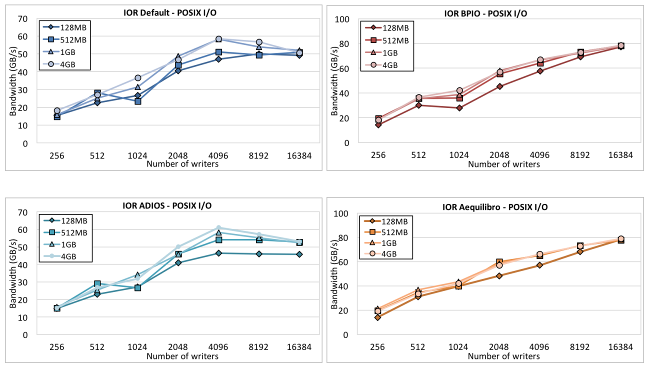

Figure 6. Scaling of aggregate write bandwidth.

more than 30 scaled IOR runs per node allocation. The number of allocated nodes ranges from 16 to 1,024 with 16 MPI processes per node which corresponds to a range from 128 to 16,384 processes in total. Each process writes a separate file. For the external interference, more than 100 samples were obtained.

C. Synthetic Benchmark Results

Figure 4 displays the results for large-scale runs for 2,048 and 4,096 nodes. We compare the bandwidth performance of IOR Default and IOR BPIO (denoted as Default vs. BPIO) and IOR ADIOS and IOR Aequilibro (denoted as ADIOS vs. Aequilibro) with the help of Equation 2. In both cases, the integration provides significant performance improvements. IOR BPIO provides an average improvement of 50% for POSIX I/O with 2,048 and 4,096 nodes. For MPI-IO, a 20–25% improvement can be achieved. Almost identical results can be observed for IOR Aequilibro. POSIX I/O can be improved by 43% on average while MPI-IO shows an improvement of about 25%. Overall, the achieved improvement for MPI-IO is slightly less than for POSIX I/O. This is a known issue on Cray systems, refer to Lofstead [17] for observations made on Jaguar. The reason is to be determined, but could be related to the underlying MPI-IO implementation. While there are variations across different runs, it can be observed that the trend remains the same. There are consistent performance gains across multiple runs and iterations. Optimizing the overall I/O cost leads to a reduced application execution time (especially for large-scale runs) and therefore, to a reduced operational cost per executed application.

Figure 5 displays the average bandwidth results for IOR Default, IOR BPIO, IOR ADIOS, and IOR Aequilibro for POSIX and MPI-IO including error bars, respectively. The error bars depict the maximum and minimum bandwidths achieved. From the performance results, we make the following observations. First, for small-scale runs with less then 128 nodes, the effectiveness of our integration is relatively small. Second, as we scale up in terms of I/O processes and allocated nodes, POSIX and MPI-IO both benefit from the BPIO integration in IOR. For example, large-scale runs with a 4,096 node allocation provide an average throughput of 172.8 GB/s for POSIX using IOR BPIO. That can be translated to an improvement of 50% in comparison to the default data placement. For MPI-IO, IOR BPIO provides an average throughput of 140 GB/s for a 4,096 node allocation. For IOR Aequilibro, a similar performance trend can be observed. For POSIX, large-scale runs provide an average bandwidth of 128.9 GB/s for a 2,048 node allocation and 158.9 GB/s using 4,096 nodes. The performance results obtained from runs using the MPI-IO interface show a similar trend as IOR BPIO. For a run with a 2,048 node allocation, IOR Aequilibro provides an average throughput of 117.4 GB/s which translates to a performance improvement of 24%. For 4,096 nodes, an average throughput of 135.6 GB/s can be obtained, which translates to a 25% performance improvement. The difference between the average bandwidth results of IOR BPIO and IOR Aequilibro can be explained by the additional overhead introduced by the ADIOS framework. The current BPIO implementation assumes that the Lustre stripe count is set to one. This limitation is imposed by the Lustre deployed on Spider II. Even with this restriction, Aequilibro provides a performance improvement and a better throughput per second.

Figure 6 presents the results of the scaled internal interference tests, as introduced in Section IV-A2. The results represent the average write bandwidth on Titan for IOR with POSIX I/O. As we scale up the number of nodes and writers, it can be observed that in the case of IOR Default and IOR ADIOS the effects of internal interference consistently decrease the average bandwidth with an increasing number of writers. For IOR BPIO and IOR Aequilibro, the internal

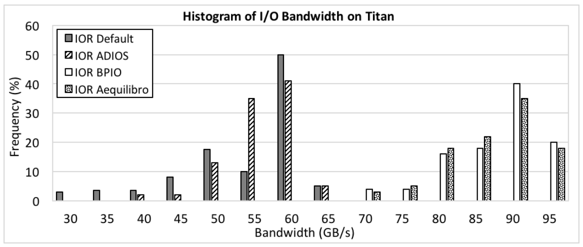

Figure 7. I/O performance variability due to external interference.

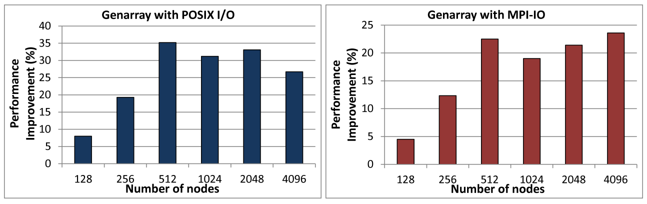

Figure 8. Average I/O bandwidth improvement for S3D workload.

interference effects can still be observed, but they have less impact on the overall bandwidth performance due to a balanced data distribution over all available storage targets.

Figure 7 displays the histograms of I/O bandwidth based on the external interference tests data collected in over 100 runs. It can be seen that in busy production environments like Titan, there is a substantial I/O variability between different runs. Note, we utilize BPIO and Aequilibro solely for our test runs. There are multiple other applications scheduled at the same time. In addition, the imbalance factor is calculated based on Equation 3. For IOR Default, the imbalance factor is about 6.9 in average, while for IOR BPIO it ranges in between 1.3 and 1.9 leading to an improvement by a factor of 3.5. For both IOR ADIOS and IOR Aequilibro, the imbalance factor ranges in between 1.1 and 1.2. ADIOS offers synchronous write methods. The imbalance factor does not provide any information about the overall performance.

The experimental results of our series of tests demonstrate that a framework like ADIOS highly benefits from a balancing library like BPIO. Even though the tests were carried out in a busy environment, Aequilibro is able to mitigate effects of I/O interferences. Similar performance trends can be observed for MPI-IO (not reported due to brevity), but with inferior bandwidth performance. We conclude that combining tools tackling different sides of the same coin resolve I/O and resource contention even when just utilized for specific applications.

D. HPC Application Results

We perform scaled runs with 128, 256, 512, 1,024, 2,048, and 4,096 nodes which correspond to 2,048, 4,096, 8,192, 16,384, 32,768, and 65,536 MPI processes, respectively. We use weak scaling of the problem size grid such that each process generates an 8 MB output/checkpoint file periodically (10 checkpoints in each run). The I/O bandwidth measurement is performed for default (ADIOS) and balanced data placement (Aequilibro) by running three Genarray simulations within the same allocation. Figure 8 displays the summary of the I/O bandwidth improvements observed for Genarray. The improvements are averaged over ten runs for each configuration. It can be observed that smaller node count (i.e., 128 and 256) runs result in a lower performance improvement. For large-scale runs, we observe that Aequilibro significantly improves the I/O bandwidth. This is consistent with the IOR synthetic benchmark performance results. For large node/processor counts, ADIOS-based applications can directly benefit from the BPIO library without any additional code changes.

V. Related Work

I/O load imbalance [18] and the increasing gap between compute capabilities of HPC systems and their I/O capabilities are known problems in the HPC domain. A large number of research efforts have been proposed to address these problems and to provide better I/O techniques [19].

There are two distinct but related research areas for scientific HPC. The first one focuses around the interconnect network which is the backbone for message exchange and I/O traffic. For example, Luo et al. [20] introduce a preemptive, core stateless optimization approach based on open loop end-point throttling. Ezell et al. [12] present a Fine-Grained Routing (FGR) congestion avoidance method. Li et al. [21] present ASCAR, a storage traffic management system for improving the bandwidth utilization and fairness of resource allocation. All I/O patterns of all enumerations of applications have to be analyzed to generate a traffic rule set. This is not feasible for large-scale HPC systems like Titan. There are too many applications and the I/O patterns change drastically depending upon the run.

The second area takes a file and storage-system centric view. Zhu et al. [22] present CEFT-PVFS, a modification to the PVFS file system [23] to achieve a better I/O load balancing. Singh et al. [24] address the problem of load imbalance in the setting of cloud data centers. Congestion and load imbalance can still occur at large scale as demonstrated by Dillow et al. [25]. Luu et al. [13] analyze the problem of low I/O performance on leading HPC systems. They use Darshan [26] logs of over a million jobs representing a combined total of six years of I/O. Even though the platforms' file systems have a peak throughput of hundreds of GB/s, only few applications experience high I/O throughput. Lofstead et al. [17] introduce *Adaptive IO*, a set of dynamic and proactive methods for managing I/O interference. The methods are bundled in a new ADIOS transport method that dynamically shifts work from heavily used areas of the storage system to areas that are more lightly loaded. The design is limited on how quickly a coordinator can react to storage load dynamics. Liu et al. [27] introduce an ADIOS transport method that attempts to re-route I/O to less loaded storage areas while applying a throttling technique that limits how much data can be re-routed during writing. Gainaru et al. [28] introduce a global scheduler that minimizes congestion caused by I/O interference by considering the application's past behaviors and system characteristics when scheduling I/O requests.

This paper complements previous work by bridging the gap between the two introduced research ares, interconnect level, and file and storage-system centric view. It provides users with an easy-to-use framework that takes full advantage of the optimizations done at the interconnect level, the load balancing done at the file system level, and the benefits of an I/O middleware library such as ADIOS.

A third area of related work is introduced by commercial data centers where load imbalance and Quality of Service (QoS) are also major concerns for multi-tenant systems. Such methods and techniques are exemplified by *Pulsar* [29], *Baraat* [30], and *Corral* [31]. Pulsar consists of a logically centralized controller with full visibility of the data center topology and a rate enforcer inside the hypervisor at each compute node. It offers tenants their own dedicated virtual data centers (VDC) to ensure end-to-end throughput guarantees. Baraat is a decentralized, task-aware scheduling system. It schedules tasks in a FIFO order, and avoids head-of-line blocking by dynamically changing the level of multiplexing in the network. Based on data from past data center studies, the application behavior is characterized in order to apply the task-aware scheduling heuristic. Corral is based on the assumption that a large fraction of production jobs are recurring with predictable characteristics. It uses characteristics of future workloads to determine an offline schedule which coordinates the placement of data and tasks.

In the scientific HPC context though, we argue that the application requirement and expectation, the highly specialized workload, and the architectural design and integration workflow present some unique challenges on leveraging techniques developed in cloud computing and data center settings [32]. Aequilibro and BPIO were developed for a scientific HPC environment and tested on a system optimized for large-scale simulations, and have not been able to take advantage of these techniques yet. Many of the assumptions made for commercial data center performance optimizations are not applicable to large-scale scientific simulation systems. For example, scientific HPC systems do not have a global view on all available system resources and allocations. Also, storage and I/O systems have been shifting from a machine-exclusive paradigm to a data-centric paradigm where the mixed workload becomes a norm and much less predictable than before [33]. The scientific workloads itself differ from commercial workloads, e.g. search queries, data analytics jobs, and social news-feeds. Commercial and scientific applications have different requirements [32]. While commercial codes can be classified as high-throughput computing, scientific workloads are categorized as latency-sensitive, large-scale, and tightly coupled computations. They assume the presence of a high-bandwidth, low-latency interconnect, a shared parallel file system between compute nodes, and a head node that can submit MPI jobs to all worker nodes.

VI. Conclusions

This paper attempts to resolve I/O contention in busy HPC environments, where multiple, concurrent applications compete for resources. Parallel file systems provide great performance and scalability. However, projecting these to the user applications can be a challenge, mainly due to load imbalances and resource contentions. Most middleware frameworks, such as ADIOS, provide a user friendly way to handle I/O, but they are high-level and not fully adequate to completely translate the underlying raw I/O performance to user applications. Balanced Placement I/O (BPIO), on the other hand, is a user-space library that mitigates resource contention and load imbalance at the lowest level, thereby improving the application level performance. We propose Aequilibro, a transparent unification of BPIO and ADIOS. With our effort, large-scale applications integrated with ADIOS can directly benefit from the BPIO end-to-end and per job I/O performance improvements. Aequilibro is evaluated with IOR, a synthetic benchmark, for two popular APIs, POSIX and MPI-IO. In addition, a real-world HPC workload is used for evaluation. Our results show that ADIOS *POSIX* can be improved by up to 50% on per job basis while ADIOS *MPI_AGGREGATE* shows performance improvements of up to 25%. The simplicity of the integration into a framework like ADIOS shows that BPIO is a viable solution for improving the overall I/O performance.

Future work will cover the investigation of the optimal number of aggregators for Aequilibro. Currently, BPIO cannot handle single shared files. Future Lustre releases support the placement of single data stripes. The BPIO library needs to be enhanced by that feature. BPIO calculates the placement based on a weighted function. The impact of changing the weights needs more research. Although our evaluation is centered around Titan and Spider II, load imbalance and resource contention are a common problem in large-scale HPC systems. We believe that Aequilibro and our proposed techniques can be applied to HPC platforms that lack a centralized resource manager. One possibility is the implementation of a pre-loadable library that transparently adds the load balancing to the application during the link phase of MPI compiler scripts. In addition, the support of HDF5 is planned.

Acknowledgment

We thank the anonymous reviews for their valuable feedback. 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, pp. 656–663, doi:10.1109/PADSW.2014.7097866.

[2] 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 *6th International Workshop on Challenges of Large Applications in Distributed Environments*, 2008, pp. 15–24, doi:10.1145/1383529.1383533.

[3] ORNL, "The Adaptable IO System (ADIOS)," https://www.olcf.ornl.gov/center-projects/adios/, June 2016.

[4] 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 *Cray User Group Conference (CUG)*, 2012.

[5] 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 *Cray User Group Conference (CUG 2013)*, 2013.

[6] R. Thakur, W. Gropp, and E. Lusk, "On Implementing MPI-IO Portably and with High Performance," in *6th Workshop on I/O in Parallel and Distributed Systems*, 1999, pp. 23–32, doi:10.1145/301816.301826.

[7] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An Overview of the HDF5 Technology Suite and its Applications," in *EDBT/ICDT 2011 Workshop on Array Databases (AD '11)*, 2011, pp. 36–47, doi:10.1145/1966895.1966900.

[8] LLNL, "SILO – A Mesh and Field I/O Library and Scientific Database," https://wci.llnl.gov/simulation/computer-codes/silo, 2016.

[9] LLNL, "The Interleaved Or Random (IOR) Benchmark," https://github.com/chaos/ior, June 2016.

[10] P. J. Braam *et al.*, "The Lustre Storage Architecture," 2004.

[11] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A Centralized "Zero-queue" Datacenter Network," in *2014 ACM Conference on SIGCOMM (SIGCOMM '14)*, 2014, pp. 307–318, doi:10.1145/2619239.2626309.

[12] M. Ezell, D. Dillow, S. Oral, F. Wang, D. Tiwari, D. E. Maxwell, D. Leverman, and J. Hill, "I/O Router Placement and Fine-Grained Routing on Titan to Support Spider II," in *Cray User Group Conference (CUG 2014)*, 2014.

[13] H. Luu, M. Winslett, W. Gropp, R. Ross, P. Carns, K. Harms, M. Prabhat, S. Byna, and Y. Yao, "A Multiplatform Study of I/O Behavior on Petascale Supercomputers," in *24th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '15)*, 2015, pp. 33–44, doi:10.1145/2749246.2749269.

[14] OpenSFS, "Lustre 2.7.0 Released," http://lustre.org/lustre-2-7-0-released/, March 2015.

[15] J. H. Chen, A. Choudhary, B. De Supinski, M. DeVries, E. Hawkes, S. Klasky, W. Liao, K. Ma, J. Mellor-Crummey, N. Podhorszki *et al.*, "Terascale Direct Numerical Simulations of Turbulent Combustion using S3D," *Computational Science & Discovery*, vol. 2, no. 1, p. 015001, January 2009.

[16] H. Shan, K. Antypas, and J. Shalf, "Characterizing and Predicting the I/O Performance of HPC Applications using a Parameterized Synthetic Benchmark," in *2008 ACM/IEEE Conference on Supercomputing (SC08)*, 2008, pp. 42:1–42:12, doi:10.1109/SC.2008.5222721.

[17] J. Lofstead, F. Zheng, Q. Liu, S. Klasky, R. Oldfield, T. Kordenbrock, K. Schwan, and M. Wolf, "Managing Variability in the IO Performance of Petascale Storage Systems," in *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC10)*, 2010, pp. 1–12, doi:10.1109/SC.2010.32.

[18] J. M. Del Rosario and A. N. Choudhary, "High-performance I/O for Massively Parallel Computers: Problems and Prospects," *IEEE Computer*, vol. 27, no. 3, pp. 59–68, March 1994, doi:10.1109/2.268887.

[19] S. Ahern *et al.*, "Scientific Discovery at the Exascale: Report from the DOE ASCR 2011 Workshop on Exascale Data Management, Analysis and Visualization," http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Exascale-ASCR-Analysis.pdf, 2011.

[20] M. Luo, D. K. Panda, K. Z. Ibrahim, and C. Iancu, "Congestion Avoidance on Manycore High Performance Computing Systems," in *26th ACM International Conference on Supercomputing (ICS '12)*, 2012, pp. 121–132, doi:10.1145/2304576.2304594.

[21] Y. Li, X. Lu, E. L. Miller, and D. D. Long, "ASCAR: Automating contention management for high-performance storage systems," in *31st Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2015, pp. 1–16, doi:10.1109/MSST.2015.7208287.

[22] Y. Zhu, H. Jiang, X. Qin, D. Feng, and D. R. Swanson, "Improved Read Performance in a Cost-effective, Fault-tolerant Parallel Virtual File System (CEFT-PVFS)," in *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid)*, 2003, pp. 730–735, doi:10.1109/CCGRID.2003.1199440.

[23] R. B. Ross, R. Thakur *et al.*, "PVFS: A Parallel File System for Linux Clusters," in *4th Annual Linux Showcase and Conference*, 2000, pp. 391–430.

[24] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage Virtualization: Integration and Load Balancing in Data Centers," in *2008 ACM/IEEE Conference on Supercomputing (SC08)*, 2008, pp. 53:1–53:12, doi:10.1109/SC.2008.5222625.

[25] D. Dillow, G. M. Shipman, S. Oral, Z. Zhang, Y. Kim *et al.*, "Enhancing I/O Throughput via Efficient Routing and Placement for Large-scale Parallel File Systems," in *IEEE 30th International Performance Computing and Communications Conference (IPCCC)*, 2011, pp. 1–9, doi:10.1109/PCCC.2011.6108062.

[26] 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," *ACM Transactions on Storage (TOS)*, vol. 7, no. 3, pp. 8:1–8:26, October 2011, doi:10.1145/2027066.2027068.

[27] Q. Liu, N. Podhorszki, J. Logan, and S. Klasky, "Runtime I/O Re-Routing + Throttling on HPC Storage," in *5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '13)*, 2013.

[28] A. Gainaru, G. Aupy, A. Benoit, F. Cappello, Y. Robert, and M. Snir, "Scheduling the I/O of HPC Applications Under Congestion," in *2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015, pp. 1013–1022, doi:10.1109/IPDPS.2015.116.

[29] S. Angel, H. Ballani, T. Karagiannis, G. OShea, and E. Thereska, "End-to-end Performance Isolation Through Virtual Datacenters," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*, 2014, pp. 233–248.

[30] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized Task-aware Scheduling for Data Center Networks," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, 2014, pp. 431–442, doi:10.1145/2740070.2626322.

[31] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware Scheduling for Data-parallel Jobs: Plan When You Can," in *2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 407–420, doi:10.1145/2829988.2787488.

[32] K. Yelick *et al.*, "The Magellan Report on Cloud Computing for Science," http://science.energy.gov/~/media/ascr/pdf/program-documents/docs/Magellan_Final_Report.pdf, 2011.

[33] G. Shipman, D. Dillow, S. Oral, and F. Wang, "The Spider Center Wide File System: From Concept to Reality," in *Cray User Group Conference (CUG 2009)*, 2009.