ISOBAR Hybrid Compression-I/O Interleaving for Large-scale Parallel I/O Optimization

Eric R. Schendel [1,2,+], Saurabh V. Pendse [1,2,+], John Jenkins [1,2], David A. Boyuka II [1,2], Zhenhuan Gong [1,2], Sriram Lakshminarasimhan [1,2], Qing Liu [2], Hemanth Kolla [3], Jackie Chen [3], Scott Klasky [2], Robert Ross [4], Nagiza F. Samatova [1,2,*]

[1] North Carolina State University, Raleigh, NC 27695, USA

[2] Oak Ridge National Laboratory, Oak Ridge, TN 37830, USA

[3] Sandia National Laboratory, Livermore, CA 94551, USA

[4] Argonne National Laboratory, Argonne, IL 60439, USA

[*] Corresponding author: samatova@csc.ncsu.edu
[+] Authors contributed equally

ABSTRACT

Current peta-scale data analytics frameworks suffer from a significant performance bottleneck due to an imbalance between their enormous computational power and limited I/O bandwidth. Using data compression schemes to reduce the amount of I/O activity is a promising approach to addressing this problem. In this paper, we propose a hybrid framework for interleaving I/O with data compression to achieve improved I/O throughput side-by-side with reduced dataset size. We evaluate several interleaving strategies, present theoretical models, and evaluate the efficiency and scalability of our approach through comparative analysis. With our theoretical model, considering 19 real-world scientific datasets both from the public domain and peta-scale simulations, we estimate that the hybrid method can result in a 12 to 46% increase in throughput on hard-to-compress scientific datasets. At the reported peak bandwidth of 60 GB/s of uncompressed data for a current, leadership-class parallel I/O system, this translates into an effective gain of 7 to 28 GB/s in aggregate throughput.

Categories and Subject Descriptors

D.4.2 [**Storage Management**]: Secondary storage—*parallel data compression, file storage*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

HPDC'12, June 18–22, 2012, Delft, The Netherlands.

Keywords

ISOBAR; Hybrid Interleaving; Staging; High Performance Computing; Lossless Compression; I/O

1. INTRODUCTION

As exascale computing comes closer to becoming reality and more powerful High Performance Computing (HPC) systems become available, the complexity of scientific simulations and analyses has grown commensurately. Unfortunately, the level of disk I/O performance offered by these systems has not kept up, leading to a serious bottleneck in read and write performance in these applications. This problem is exacerbated by the increasing frequency of checkpoint operations performed by such computations due to their increasing vulnerability to node failures at this scale, which further adds to the I/O overload.

Ideally, the solution to the I/O bottleneck will involve both data reduction and parallel I/O access pattern optimization. Unfortunately, these two optimization methods have traditionally been in conflict. State-of-the-art I/O middleware solutions, such as ADIOS [22], HDF5 [37], and PnetCDF [20], have no native support for write compression in a parallel context, due to the complexity of handling the resultant non-uniform data, which requires synchronization between all nodes performing shared-file I/O. Further constraining the problem is the fact that scientists cannot sacrifice simulation fidelity, especially at checkpoints, which rules out lossy compression as a viable data reduction method. And yet, typical lossless compression techniques are ineffective on hard-to-compress floating-point data generally produced by such simulations.

However, we argue that these goals can in fact be complementary. Our key insight is that, by dynamically identifying a subset of highly-compressible data to process while asynchronously writing the remainder to storage, we can effectively hide the cost of compression and I/O synchronization behind this transfer, thus rendering parallel write compression viable. This interleaving method is a natural fit for

data staging architectures, where various data transformations can occur while data is "in transit" or in-situ, from compute nodes to disk. Traditionally, staging has been used to compute statistical analyses or perform indexing operations [1, 2]. With interleaved compression and I/O, however, we can augment this functionality by performing compression as an in-situ transfer and storage optimization, as well.

The problem of identifying a highly-compressible subset of the original data is itself quite difficult, as scientific data is usually hard-to-compress with typical compression libraries. We argue that this is because I/O libraries optimized for scientific applications tend to view multi-byte data elements, such as floating-point values, as atomic units of data. Instead, by relaxing this notion and utilizing *byte-level analysis* of the scientific data, better results can be obtained. ISOBAR, or In-Situ Orthogonal Byte Aggregate Reduction, enables exactly this sort of analysis for lossless compression, and has been shown to be effective on such datasets [30]. By modifying ISOBAR's analysis methods to partition the data into compressible and incompressible byte streams, we form an effective basis for interleaving the usage of computing resources. By transmitting the incompressible data over the network immediately, we can hide the cost of compressing the remainder and synchronizing for non-uniform I/O to write it to disk.

Although we demonstrate performance gains when using our methodology in a leadership-class HPC system (the Cray XK6 Jaguar cluster), it is impossible to evaluate our system on every possible cluster configuration. For this reason, it is also desirable to have an analytical performance model, which would allow prediction of performance characteristics on new hardware and software, and would aid application developers in configuration choices.

Therefore, we present a *hybrid compression-I/O* methodology for data reduction and I/O optimization. By employing ISOBAR analysis within a data staging architecture and incorporating the popular ADIOS I/O framework as our I/O backend, we implement effective parallel compression with state-of-the-art I/O performance. Furthermore, we develop a resource interleaving strategy to process the compressible and incompressible components of the data simultaneously, as identified by ISOBAR-analysis. This enables immediate asynchronous transfer and writing of incompressible data while compression is applied concurrently to the remainder. Additionally, we develop a performance model for our methodology, which we demonstate to have a high degree of accuracy through validation against empirical data.

Our system exhibits read and write performance gains proportional to the degree of data reduction, which ranges as high as 46% on scientific datasets. This would translate into an effective increase of 28 GB/s bandwidth over the peak aggregate throughput of 60 GB/s of uncompressed data offered by the leadership-class Lustre parallel filesystem at Oak Ridge National Labs [24]. Even under worst-case conditions, where the dataset is highly entropic and difficult to compress, we show that our system still maintains a gain in throughput over the state-of-the-art.

2. BACKGROUND

2.1 ISOBAR

ISOBAR is a lossless compression method that we built specifically for data that varies in compressibility on a byte-by-byte basis [30]. A ubiquitous example of such data is scientific floating-point data, where the exponent bits can be highly similar while the significand bits are highly entropic. To this end, ISOBAR first performs a *preconditioner* on the linearized input data, selecting data to compress based on its expected degree of compressibility. This is performed by the ISOBAR-analyzer. The analyzer's objective is to identify high-entropic content within a dataset that negatively impacts the compression efficiency and reduces the burden on the compressor from processing the components with low compression potential. This enhances the compression algorithm in terms of the compression throughput as well as the compression ratio.

The input data is considered as a matrix of bytes, where each row is an input value (e.g., a double-precision floating-point) and each column is an individual byte of the input value. The preconditioner counts the frequency of each byte on a column basis and marks that column as compressible if the distribution appears to be non-random. Once these columns are identified, any general purpose compressor may be used, but ISOBAR automatically chooses the best one by user preference (compression ratio vs. speed).

2.2 ADIOS

High-performance computing applications leverage I/O libraries like HDF5 (Hierarchical Data Format), ADIOS (Adaptable I/O System), and PnetCDF (Parallel Network Common Data Form) that allow scientists to easily describe the data that needs to be written out and analyzed. These I/O libraries provide an enhanced I/O performance, by efficiently handling synchronization, and meta-data generation during shared-file writes. We choose to incorporate ADIOS for this paper, since it has been shown to deliver performance improvements of up to 300% at scale [24, 27] on the Cray Jaguar leadership-class computing facilities at ORNL.

ADIOS essentially provides an efficient componentization of the HPC I/O stack. Through an XML file, it provides the option to describe the data, and to choose the optimal transport methods like POSIX and MPI-IO without the need to recompile the application codes. The data written using ADIOS is in the form of a native BP (Binary Packed) file, comprising of "process groups," which are variables described in XML configuration, usually tagged according to their functionality. For example, checkpoint and restart data is written under a single process group, as is analysis data.

3. HYBRID COMPRESSION-I/O FOR DATA STAGING ARCHITECTURES

The prevailing I/O strategy in current peta-scale systems is to offload the burden of I/O to dedicated *staging nodes*, as shown in Figure 1. This allows minimal idle time on the compute nodes allocated for the simulation, as the I/O nodes handle the rate-limiting disk writes and reads collectively and network bandwidth is an order of magnitude faster than disk bandwidth. However, given the trend of ever-increasing simulation data sizes, combined with the need to checkpoint simulation state to minimize data loss in the face of node failure, the I/O offloading approach alone cannot keep up with the computational throughput available.

3.1 Method

A promising approach to aid in mitigating this problem is

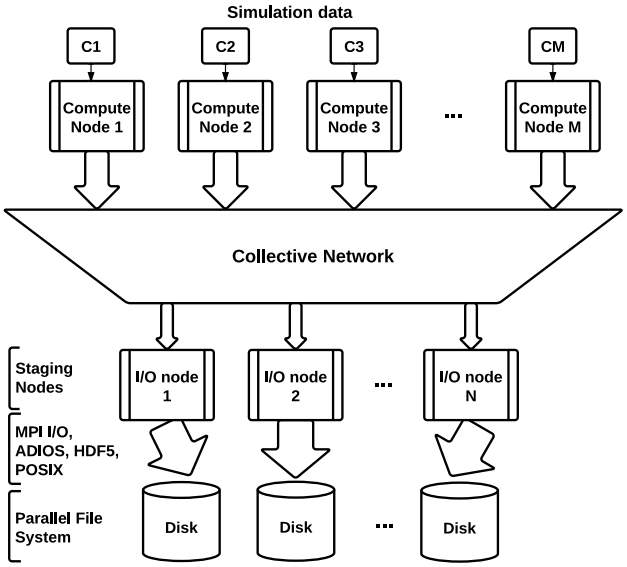


Figure 1: A peta-scale computing system with staging environment.

to write compressed data to the disk [36]. This reduces the aggregate amount of data to be written, which can alleviate the I/O bottleneck to some extent. However, there are a number of technical challenges in utilizing compression in the scientific computing environment. Most state-of-the-art compression algorithms do not provide enough compression throughput to justify the data reduction, and those that do sacrifice the compression ratio. Moreover, the target simulation data is notoriously "hard-to-compress;" traditional compression algorithms provide meager compression ratios [30]. In addition, compression brings up the non-trivial issue of managing the writes of variable sized chunks of data. This mandates a strategy to efficiently handle disk I/O, taking into account appropriate data organization and writer node synchronization while also keeping the overhead imposed by the associated metadata in check.

Current state-of-the art I/O frameworks such as ADIOS and HDF5 have no capability to compress and store simulation data when performing parallel writes to a parallel filesystem. In this work, we utilize the data staging model, based on the ISOBAR technology, to write and simultaneously compress simulation data. In addition to using the data staging paradigm, interleaving compression and I/O can also be directly integrated into I/O frameworks. ISOBAR was introduced as a high-throughput compressor built specifically for hard-to-compress datasets [30]. Especially important is the fact that we can use ISOBAR to produce multiple streams of data that, once defined by the analysis portion of ISOBAR, can be operated on independently. This presents the perfect opportunity to hide the compression costs by asynchronously writing streams (the incompressible byte streams) while operating on the remaining streams (compressing the compressible byte streams). We call this the *hybrid compression-I/O* approach. The interleaving of compression and I/O helps to hide the compression costs, while the reduction in data size reduces disk costs.

Figure 2 illustrates our generic hybrid approach. ISOBAR partitions the data into two streams: compressible and incompressible bytes, and incorporates a small, constant-sized metadata block to each (containing the buffer size and the analysis array, a bitfield of marking which bytes are compressible). The compressible stream is compressed and then written to disk, while the incompressible stream can be immediately written to disk. There is no dependency between the two streams, so we may choose to order the operations how we wish. Thus, we issue an asynchronous write of the incompressible stream, then begin compressing the compressible stream. Finally, we write the compressible stream. This strategy has numerous benefits: we maximize resource utilization by performing network and I/O operations while compressing, and since compression throughput tends to be much higher than I/O bandwidth, it is possible to eliminate the compression costs entirely.

There are roughly two possible performance scenarios of interleaving, based on the individual performance of compression and file writing, shown in Figure 3. The worst case occurs when the uncompressed data is written before the remainder of the data is compressed. This is depicted in Figure 3a. In this scenario, the compression time is the bottleneck, and so is only partially hidden by the writing of the incompressible data to disk. However, this case generally occurs when most of the data is deemed compressible, in which case the increased compression cost directly translates into substantial data reduction; thus, overall time-to-disk is still reduced due to a higher compression ratio.

The second scenario shown in Figure 3b occurs if data compression finishes before the incompressible data has been fully written, and must wait to write the compressed data stream. In this scenario, compression time is completely hidden, and can be considered a "free" operation with respect to time. Also, the idle time can be used for other activities, such as running ISOBAR-analysis on another chunk if available. A related scenario to this is when data compression and incompressible data writing finish at approximately the same time resulting in full utilization of all resources. This case also completely hides the compression costs.

3.2 Data Layout in ADIOS

The layout of our compressed and uncompressed data on disk is managed by ADIOS's self-describing file format (.bp), which is specifically designed to attain scalable high performance I/O, to support delayed consistency and data characterization, and to maintain compatibility with standard file formats. Also, since this file format is log-based [27], new data can be appended without incurring any additional overhead, irrespective of the number of processes or timesteps. As a result, the performance improvements we report using our hybrid framework over a single timestep can be maintained over multiple timesteps.

For our system, we define two ADIOS groups (or "data containers"): one for the compressed data, and one for the uncompressed data. Every writing process submits data to each of the two groups. Depending on the transport method chosen in the configuration file, ADIOS can store data from all the writing processes into a single shared file using collective MPI-IO or multiple files using POSIX I/O with a separate file handle for each writing process. Fast access to data for a specific process or timestep is supported via footer indexes that avoid the known limitation of header-based formats [29], where any change to the length of the file data requires moving the index.

Figure 4 shows the ADIOS data layout specific to our ap-

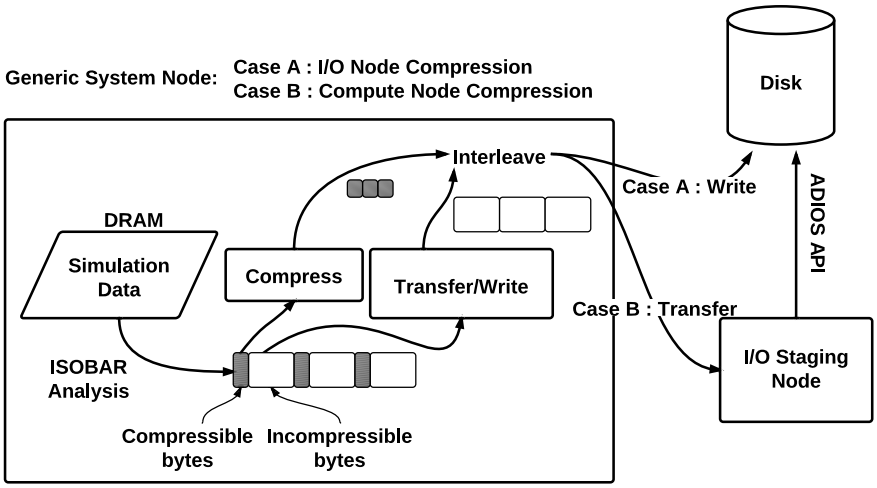
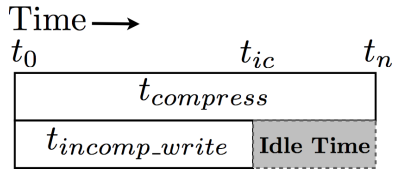
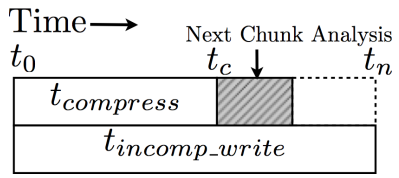


Figure 2: The hybrid compression-I/O method; interleaved compression may occur at the compute nodes or the I/O nodes.



(a) Compression is the rate limiting factor.



(b) Compression interleaving results in free time usable for the analysis of the next chunk.

Figure 3: Possible scenarios for compression and incompressible data write times.

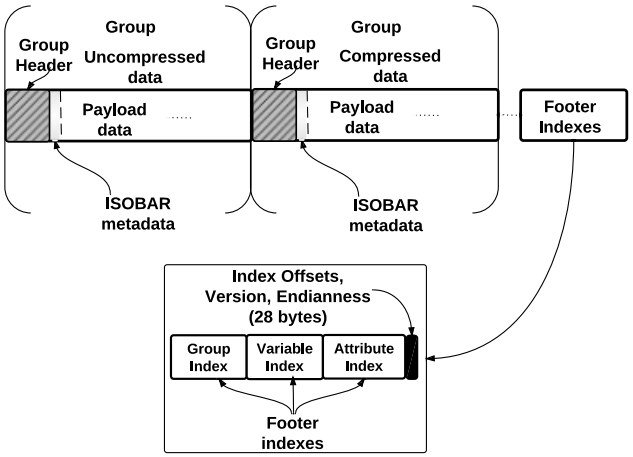


Figure 4: Data layout (with associated metadata) for a single timestep.

plication for a single timestep. ADIOS handles the data organization among groups via local group headers and indexes. The ISOBAR metadata is stored first within each group, followed by the payload data. The metadata size is dependent on the ADIOS system configuration, the chunk size, and the compute-I/O node ratio. The relative ordering of groups is arbitrary, dependent on the order in which the processes submit data and on synchronization among them (via a coordination token) [23]. The global footer indexes shown are used for query-driven data retrieval (accessing a specific timestep or process output). In this work, we confine our focus to data layout for the write-all/read-all paradigm, which is ubiquitous in HPC computing, especially in checkpoint and restart operations. From the standpoint of future work, however, our framework can be adapted for the WORM paradigm by leveraging previous work [13] to optimize the data layout on disk. In addition, support for compression schemes [19, 18, 17] which offer up to 7 times reduction in datasizes can be extended. Together, these possibilitie provide avenues for increasing throughput gains whilst maintaining read performance.

Table 1: Average metadata overhead for different interleaving strategies.

Test Case	**Average Metadata Overhead** (%)			
	ISOBAR		ADIOS	
	POSIX	MPI	POSIX	MPI
Base	0.00002	0.00002	0.00321	0.00148
ISOBAR at I/O nodes	0.00004	0.00004	0.00902	0.00417
ISOBAR at compute nodes	0.00007	0.00007	0.00978	0.00471
Serial ISOBAR	0.00004	0.00004	0.00806	0.00358

The metadata required to support this format is very small, requiring less than 0.01% overhead in the worst case, as summarized in Table 1. If applied to 1 GB of data, this translates into less than 100 KB of overhead. As shown in the table, the metadata can be split into two categories: that maintained specifically for ISOBAR, and that required by ADIOS. The exact amount of metadata is dependent on the transport method used (POSIX or MPI-IO) as well as

the location of the compressor (compute node or I/O node). The overhead is minimal in the base case (no compression), as expected, and maximal when compressing at the compute nodes, which generates the most individual streams of data. The metadata cost is also higher for the POSIX transport method, since this allocates one file per writing process, as opposed to the single shared file maintained by MPI-IO.

4. PERFORMANCE MODELING

While we demonstrate that our hybrid compression-I/O methodology provides improved I/O performance in one testing environment, there are many supercomputing systems, each with widely-varying performance characteristics. Since our optimization algorithms exhibit a strong dependence on hardware parameters, it is important to devise an accurate performance model, so that we can generalize the results collected in Section 5 to other systems. This will enable application designers to estimate the benefit of compression given their particular hardware configuration, problem characteristics, etc. We therefore develop such a performance model, which we then validate against empirical data, as reported in Section 5.4.

4.1 Model Preliminaries

Given our target cluster architecture, we make some underlying assumptions in our model. We assume a fixed compute node to I/O node ratio, ρ, consistent with the majority of I/O staging frameworks currently in use. Furthermore, on each compute node, we assume fixed-size input *chunks* of size (C), which are all written following the bulk-synchronous parallel I/O model. This is a common mode of operation when writing checkpoint and restart data, which synchronously flushes the simulation state to file, then continues with the simulation. We also assume that the I/O staging framework (e.g., ADIOS) and the network architecture provide a relatively consistent I/O and transfer rate, respectively. As shown in numerous experiments with ADIOS [22, 23], this is a reasonable assumption to make. Finally, we require some *a priori* information about the compression performance in order to accurately predict overall system performance. Fortunately, this can be gathered easily by running the ISOBAR analysis stage on a small, representative set of data to predict overall performance [30].

To provide a complete model, there are three cases of writing from compute nodes to disk that we wish to consider for comparative purposes. The first is when no compression is performed, and data is written directly to disk (through the I/O nodes). This forms our base case to compare the other compression methods against, and allows us to check the sanity of our model before looking at the more complex compression models. The remaining cases use ISOBAR interleaved compression, but in different locations. The second case compresses at the compute-node level. If time-to-disk is our sole optimization metric, then we expect the second method of compute-node compression to perform best, exhibiting the greatest aggregate compression throughput due to the large number of compute cores utilized (by constrast, compressing at the I/O nodes yields less aggregate compression throughput by a factor of ρ, which is 8 in our experiments, but can be much higher). Our third and final case is compressing at the I/O-node level. This case is important as future staging architectures shift toward dedicating compute nodes strictly to simulation work [21], relying on asynchronous RDMA to offload data to I/O nodes and prevent simulation stalls.

Table 2: Input symbols for the performance models.

Input Symbol	**Description**
C	The chunk size
ρ	Compute to I/O node ratio
θ	Throughput of the collective network between the compute and I/O nodes
δ	Size of the metadata
μ_r	Throughput of the disk reads
μ_w	Throughput of the disk writes
α	Fraction of the chunk that is compressible
σ	Compression ratio (compressed vs original)
T_{prec}	Throughput of the ISOBAR preconditioner
T_{comp}	Compression throughput
T_{decomp}	Decompression throughput

Table 3: Output symbols for the performance models.

Output Symbol	**Description**
t_{prec}	Time to run the ISOBAR preconditioner on the data
$t_{compress}$	Time to compress the data (algorithm dependent)
$t_{transfer}$	Total transfer time [1]
$t_{comp\_transfer}$	Transfer time for compressible data [1]
$t_{incomp\_transfer}$	Transfer time for incompressible data [1]
t_{write}	Time to write data to the disk
t_{comp_write}	Time to write compressible data to the disk
t_{incomp_write}	Time to write incompressible data to the disk
t_{write_depend}	Time for all the dependencies to complete before writing the compressible byte stream
$t_{decompress}$	Time to decompress the data (algorithm dependent)
t_{incomp_read}	Time to read the incompressible data from the disk
t_{comp_read}	Time to read the compressible data from the disk
$t_{combine}$	Time to reconstruct data from compressible and incompressible portions
$t_{combine\_depend}$	Time until all data is ready to be recombined into the original chunk
t_{comp_ion}	Intermediate processing time for handling compressible data at I/O nodes
t_{incomp_cn}	Itermediate processing time for handling incompressible data at compute nodes
t_{total}	Total end-to-end data transfer time
τ	Aggregate throughput

[1]Interpretation of transfer direction based on context of usage i.e., compute to I/O for writes and I/O to compute for reads.

In this work, we refer to the I/O node as a staging node on the system that receives the data from the compute nodes and sends it to the OSS (Object Storage Servers), which

manage writing of the data to the Lustre File System. We do not operate at these file system nodes.

We will build the models in increasing order of complexity: the base case of no compression, compression at the I/O nodes, and compression at the compute nodes. Tables 2 and 3 summarize the symbols for parameters and output variables used in the model. In all scenarios, the aggregate throughput τ is given by

$$\tau = \frac{\rho \cdot C}{t_{total}}, \tag{1}$$

where ρ is the number of compute nodes and C is the chunk size.

4.2 Base Case: No Compression

In this scenario, we simply transfer the simulation data from the compute nodes to the I/O nodes ($t_{transfer}$), followed by writing the data to file (t_{write}), in a synchronous manner. The end-to-end transfer time for a chunk of data from the compute nodes to the disk ($t_{total} = t_{transfer} + t_{write}$) is similarly simple, given our assumptions on aggregate network and disk bandwidths:

$$t_{transfer} = \frac{C}{\theta} + \frac{C \cdot \rho}{\theta} \tag{2}$$

$$t_{write} = \frac{C \cdot \rho}{\mu_w} \tag{3}$$

$$t_{total} = \frac{C}{\theta}(1 + \rho) + \frac{C}{\mu_w}\rho \tag{4}$$

To reload the data from disk, the same operations occur in reverse, except with read throughputs instead of write throughputs.

4.3 I/O Node Compression Case

An overview of the compression-I/O workflow is shown in Figure 5. The I/O nodes implement the ISOBAR preconditioner and interleave the disk writes of the incompressible byte-columns with compression.

The compute nodes merely forward the raw data to the I/O nodes. In our design, we issue the I/O operation for the incompressible bytes asynchronously, allowing us to concurrently perform compression. Thus, the writing of the compressed data waits (if necessary) on the incompressible stream writing to complete. This can be captured more intuitively with the task dependency graph shown in Figure 6. Each vertex represents a task and directed edges represent task dependencies. If each edge is weighted to be the completion time of the originating task, then the longest path from the head vertex to the tail vertex, also known as the *critical path*, gives the overall run time of the interleaved process. Using this diagram, we can capture the overall runtime in a single equation. First, we define the cost of each individual task.

The total preconditioning time in this case is ρ times the preconditioning time of a single chunk. Moreover, the partitioning has to be handled by the I/O node. The partitioning throughput is approximately equal to the preconditioner throughput. Thus, the total preconditioning time is given

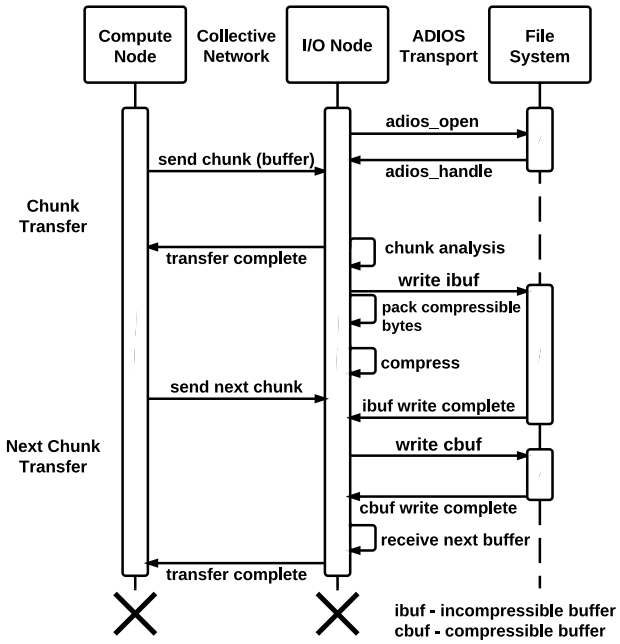


Figure 5: Compute-I/O interleaving strategy with compression at the I/O nodes.

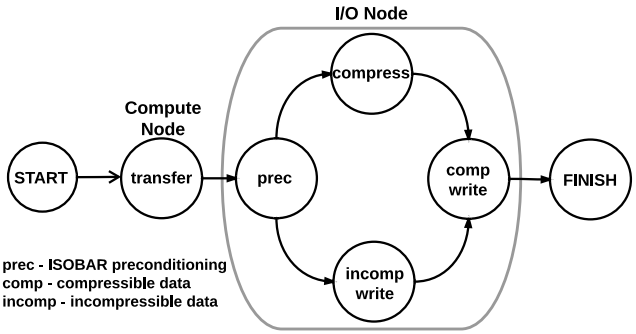


Figure 6: Task dependencies for the I/O node compression case.

by:

$$t_{prec} = 2\left(\frac{\rho C}{T_{prec}}\right) \tag{5}$$

$$t_{compress} = \frac{\rho \alpha C}{T_{comp}} \tag{6}$$

$$t_{incomp\_write} = \frac{\rho(1-\alpha)C}{\mu_w} \tag{7}$$

$$t_{comp\_write} = \frac{(\alpha\sigma C + \delta)\rho}{\mu_w} \tag{8}$$

Looking at Figure 6, we see that incompressible write (t_{incomp_write}) and compression ($t_{compress}$) are interleaved. Thus, the length of the critical path, and therefore the overall writing time, can be calculated as

$$\begin{aligned} t_{total} = {} & t_{transfer} + t_{prec} \\ & + \max(t_{compress}, t_{incomp\_write}) \\ & + t_{comp\_write}. \end{aligned} \tag{9}$$

The reading stage of restoring the chunks into memory from disk requires the dependency graph to be inverted (that is, each directed edge reversed). This maintains the interleaving property of decompression and reading of incompressible byte-columns. The preconditioner task is replaced with reconstruction, which reorders the decompressed byte streams and the incompressible byte stream to their original locations in memory. The overall reading time can be calculated as

$$\begin{aligned} t_{total} &= t_{comp\_read} \\ &\quad + \max(t_{decompress}, t_{incomp\_read}) \\ &\quad + t_{combine} + t_{transfer}. \end{aligned} \tag{10}$$

4.4 Compute Node Compression Case

The integration of ISOBAR into the compute nodes is a more nuanced task. Figure 7 shows the general workflow of the interleaved compression and network-I/O, and Figure 8 shows the corresponding task-dependency graph for this scenario.

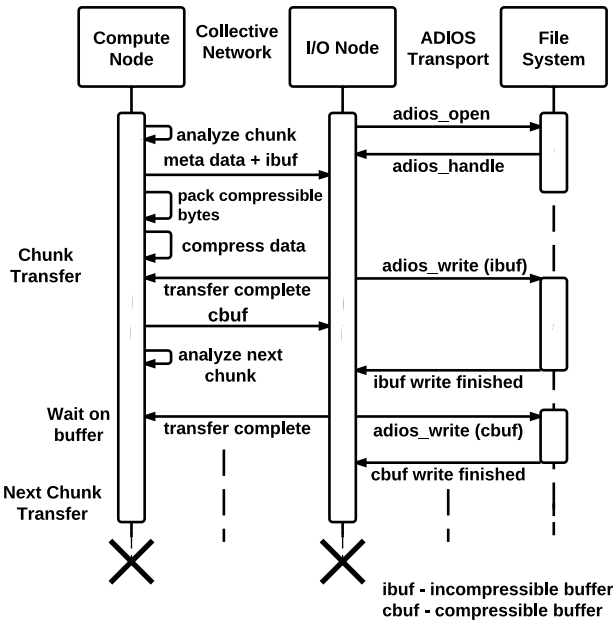


Figure 7: Compute-I/O interleaving strategy with compression at the compute nodes.

After the preconditioner is run, the incompressible byte-columns are sent asynchronously while the remaining byte-columns are compressed. Once the incompressible byte-columns are sent, the I/O nodes may immediately issue its asynchronous writing operation. Once the compression is complete, the compressed stream must wait for the incompressible network transfer to complete (if necessary) to transfer its results. Finally, once the incompressible bytes are written to disk, the compressed stream may be written.

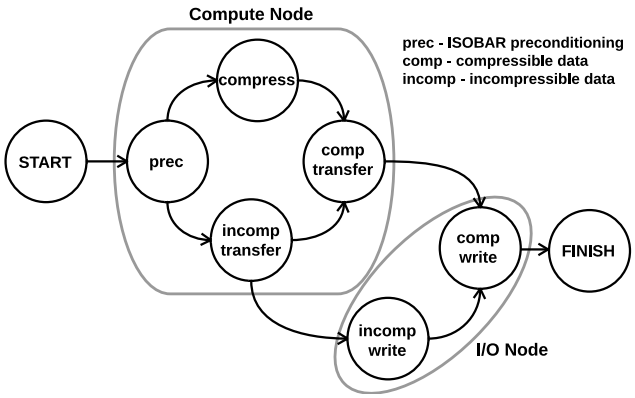


Figure 8: Task dependencies for the compute node compression case.

Individually, these operations are modeled as:

$$t_{prec} = \frac{C}{T_{prec}} \tag{11}$$

$$t_{incomp\_transfer} = \frac{(1-\alpha)C}{\theta}(1+\rho) \tag{12}$$

$$t_{incomp\_write} = \frac{(1-\alpha)C}{\mu_w}\rho \tag{13}$$

$$t_{compress} = \frac{\alpha C}{T_{comp}} \tag{14}$$

$$t_{comp\_transfer} = \frac{(\alpha\sigma C + \delta)}{\theta}(1+\rho) \tag{15}$$

$$t_{comp\_write} = \frac{(\alpha\sigma C + \delta)}{\mu_w}\rho \tag{16}$$

As discussed in Section 3 and seen in the dependency graph in Figure 8, interleaving is achieved through the compression and transfer/writing of network data. Additionally, there may be interleaving of the transfer of compressed byte-columns and the incompressible byte writing. Thankfully, given the structure of the graph and the residence of the tasks on different nodes, we may define the critical path using the following two quantities:

$$\begin{aligned} t_{comp\_ion} &= \max(t_{compress}, t_{incomp\_transfer}) \\ &\quad + t_{comp\_transfer} \end{aligned} \tag{17}$$

$$\begin{aligned} t_{write\_depend} &= \max(t_{comp\_ion}, \\ &\quad t_{incomp\_transfer} + t_{incomp\_write}) \end{aligned} \tag{18}$$

t_{comp_ion} represents the time taken to compress and send the compressible byte-columns to the I/O nodes, accounting for stalls caused by a longer incompressible byte-column transfer. t_{write_depend} represents the time taken for all dependencies to clear before writing the compressible byte stream, including possible stalls at the second network-I/O interleaving level. Adding in the preconditioner and the compressible byte-column writing, the total time-to-disk can be defined as follows:

$$t_{total} = t_{prec} + t_{write\_depend} + t_{comp\_write} \tag{19}$$

Once again, the reading of hybrid-compressed data chunks causes an inversion of the dependency graph, except that write operations are replaced with read operations, the transfers are reversed, the compressed byte-columns are decompressed, and the preconditioner task is replaced with the reconstruction task. In fact, in this particular instance, the

inverted task graph is isomorphic to the original task graph, thus simplifying building the read model. The compressible byte-columns are read and then asynchronously sent to the compute nodes while the incompressible byte-columns are read. Afterwards, the incompressible byte-columns are transferred asynchronously while the decompression process begins. Finally, the compressible and incompressible columns are recombined. This is captured in the following model:

$$t_{comp\_read} = \frac{(\alpha\sigma C + \delta)\rho}{\mu_r} \tag{20}$$

$$t_{comp\_transfer} = \frac{\alpha\sigma C + \delta}{\theta} \tag{21}$$

$$t_{decompress} = \frac{\alpha\sigma C}{T_{decomp}} \tag{22}$$

$$t_{incomp\_read} = \frac{(1-\alpha)C\rho}{\mu_r} \tag{23}$$

$$t_{incomp\_transfer} = \frac{(1-\alpha)C}{\theta} \tag{24}$$

The reconstruction time is assumed to be constant for fixed sized data chunks. Similar to the write case, the critical paths are defined as follows:

$$t_{incomp\_cn} = \max(t_{incomp\_read}, t_{comp\_transfer}) + t_{incomp\_transfer} \tag{25}$$

$$t_{combine\_depend} = \max(t_{incomp\_cn}, t_{comp\_transfer} + t_{decompress}), \tag{26}$$

where t_{incomp_cn} represents the time taken for the incompressible byte stream to reach the compute nodes, taking into account stalls as a result of the compressible byte stream being sent first, and $t_{combine\_depend}$ represents the time until all data is ready to be recombined into the original chunk of data. Thus, the total time to restore the data to its original state is

$$t_{total} = t_{comp\_read} + t_{combine\_depend} + t_{combine}. \tag{27}$$

5. EXPERIMENTS AND RESULTS

In this section, we present the empirical evaluations of our framework via a set of microbenchmarks to evaluate the throughput performance for the writes as well as the reads. We report the percentage improvement in performance obtained using the hybrid compression-I/O framework (at the compute as well as the I/O nodes) over the base case without compression. We also report theoretical evaluations for the interleaving strategies discussed in Section 4 via performance model simulations. Lastly, we specify the parameters used for the simulations and present a comparison between the predicted and actual system performance.

5.1 Experimental Setup

Our experiments were conducted on the Cray XK6 Jaguar cluster at the Oak Ridge Leadership Computing Facility (OCLF). It consists of 18,688 compute systems, each containing a 16-core 2.2 GHz AMD Opteron 6724 processor and 32 GB of RAM. It uses the Lustre [31] file system for parallel I/O and a high performance Gemini interconnect for communication. The compute-I/O node ratio for all experiments is kept fixed at 8 : 1. The definitive choice of a single optimal ratio is non-trivial, since it depends on the size of the data being moved, the degree of inter-node communication, as well as the memory requirement of the staging nodes. The study in this realm is the subject of future work.

We evaluated the system characteristics using a set of micro-benchmarks to measure the network and disk I/O throughputs. The aggregate network throughput for our experiment cases was measured to be 530 MB/s on an average, while the read and write throughputs were measured to be 62.6 MB/s and 15.6 MB/s per node, respectively. It should be noted that for all our experiments, we refer to the term "node" as a processing core on the Jaguar system.

We use the `gts_chkp_zion`, `flash_velx` and `s3d_temp` datasets discussed in [11, 9, 30, 33] for our analyses. The datasets are chosen so as to reflect the entire compressibility spectrum across a range of scientific datasets. The GTS dataset consists of about 2.4 million double precision values of the zion variable's checkpoint and restart data for each 10th timestep of the GTS simulation. It consists of entirely unique values. It has a high degree of apparent randomness and is one of the most "hard-to-compress" scientific dataset. Note that this is only *apparent* randomness; in reality, the dataset contains patterns that are non-trivial to isolate, preventing general-purpose compressors from leveraging them.

The FLASH dataset consists of about 68.1 million double precision values of the velocity variable with entirely unique values. It is also hard-to-compress and exhibits compressibility characteristics similar to most scientific datasets discussed in [30]. Therefore, it is a good representative for a large number of scientific datasets under consideration.

The S3D dataset, on the other hand, consists of about 20.2 million double precision values of the temperature variable with 46% unique values. It is relatively less hard-to-compress in comparision with the other two datasets.

The ISOBAR framework supports the use of any general purpose byte-level compression algorithm. However, for our evaluations, we use `zlib` [12], designed by Jean-loup Gailly and Mark Adler. It is a lossless compression /decompression algorithm that uses an LZ77 algorithm variant to compress the data in a block sequence. In addition to scoring high marks in general-purpose compression rate and compression throughput, the memory usage of `zlib` is independent of any input data.

In addition to the three interleaving strategies discussed in Section 4, we also include a serial compression case for completeness sake, wherein we apply ISOBAR compression serially, i.e., we do not interleave compressible and incompressible data processing. This is essentially an extension of the base case, allowing us to directly evaluate the impact of interleaving.

5.2 Write Performance

Figure 9 shows the results gathered from write microbenchmarks. For each dataset, the results are reported in terms of the percent improvement in the write throughput relative to the base case, measured for each of the four scenarios (i.e. the base case, interleaving at compute nodes, and interleaving at I/O nodes, and the serial compression), versus the number of compute nodes.

We observe that both interleaved approaches (compute node and I/O node compression) yield an improvement in performance over state-of-the-art I/O middleware framework without compression (the base case). As expected, interleaving using compute node compression results in the highest

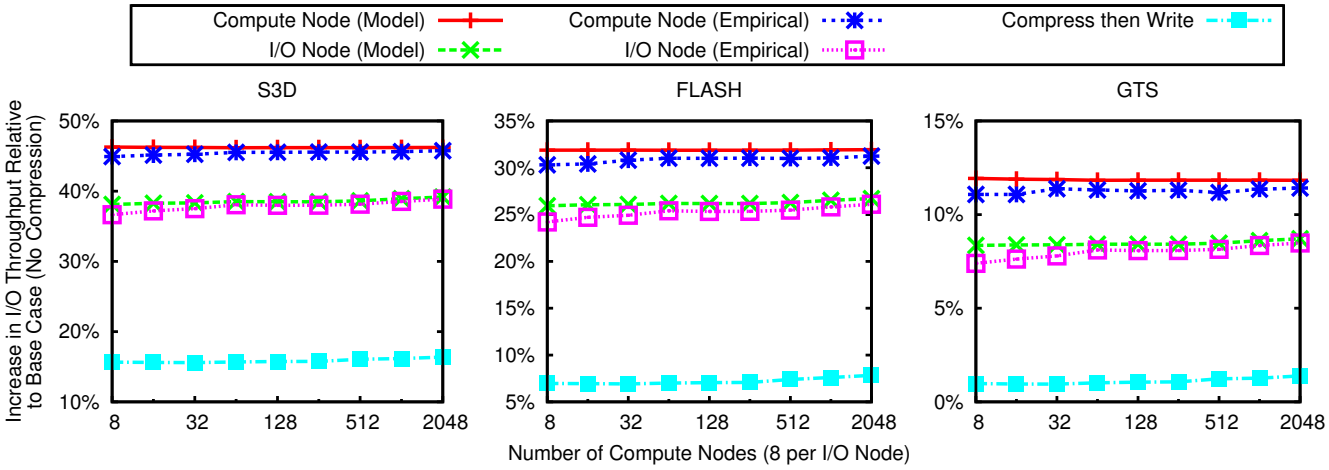


Figure 9: Model and empirical end-to-end write throughput versus number of compute nodes (weak scaling).

performance gain, from around 12% over the base case for the GTS dataset and to as high as 46% over the base case for the S3D dataset. Improvements for the FLASH dataset are 31% over the base case. On the other hand, interleaving using I/O node compression yields improvements in the range of about 8% for the GTS dataset, 37% for the S3D dataset, and 25% for the FLASH dataset. Using ISOBAR serially yields a modest 1% (GTS) to 16% (S3D) gain in throughput performance, suggesting that a significant portion of the performance boost comes from our compression-I/O interleaving strategy, affirming the efficacy of this approach.

The experiments are conducted with weak scaling up to 2048 nodes on the Jaguar system. The stability of the results over a varying number of cores suggests that the framework is, indeed, scalable.

5.3 Read Performance

We carried out equivalent read micro-benchmark tests on the disk data, evaluating the base case (without decompression) and each of the two interleaved decompression strategies (decompression at the compute nodes and at the I/O nodes).

The experimentation results of the read microbenchmarks are shown in Figure 10 in the form of percent improvements over the base case (i.e., direct reads without decompression). Both the interleaved approaches for the reads exhibit performance gains of the same order as reported for the writes for all the datasets. This suggests that the hybrid framework is symmetric with respect to reads as well as writes.

In order to support asynchronous processing of the compressible and incompressible portions of the data for the interleaved scenarios, we used a separate file per ADIOS group. The reason for this is that ADIOS currently reads data from all the groups upfront when using a single file and this operation is inherently blocking, i.e., a request for the read of only the compressed buffer requires the entire data to be read. The two-file-approach does not affect the write performance.

5.4 Performance Modeling

The performance models for evaluation were setup to use the following parameter values. The compute-I/O node ratio was chosen as $\rho = 8 : 1$. Compression efficiency is sensitive to the chunk size for most lossless compression techniques that adapt based on calculated statistics of the subject data [35, 15]. We chose a chunk size $C = 3$ MB taking into account the sensitivity of most lossless compression techniques to the input chunk size [35]. The ISOBAR preconditioner operates at an approximate throughput of $T_{prec} = 500$ MB/s. Other ISOBAR specific parameters were chosen based on the statistical analyses of 24 different scientific datasets [30], 19 of which were "hard-to-compress." The average values of these parameters based on the application type (i.e., α, σ, T_{comp}, T_{decomp} from Table 2) are shown in Table 4.

The predicted performance for the data writes and reads for all the test scenarios and evaluation datasets are shown in Figure 9 and 10, respectively. It is evident that the theoretical performance improvements are generally consistent with the empirical results. Though some small overestimating bias is visible, it is itself relatively consistent, and can therefore be readily factored out (as has been done in Table 4). Additionally, the fact that the trends exhibited by the model predictions are equivalent to those of the measured results points to a mismeasured system parameter as the likely culprit for the minor discrepency that exists. Thus, the performance model can be used to closely approximate the true behavior of the system.

In addition, we also performed theoretical evaluations of our framework on other scientific datasets from various application domains. These include the MSG [7], OBS [7], and NUM [8, 28] datasets, all of which normally produce a reasonable amount of data per process. The results for the best strategy (interleaving at the compute nodes) for reads as well as writes are shown in the Table 4.

We observe that the expected theoretical performance gains for these more typical datasets are as high as 32%, even after accounting for the observed model bias. This shows that our framework improves significantly on less harder to compress datasets and that the performance gains are directly proportional to the compression ratio of datasets.

6. RELATED WORK

The ISOBAR hybrid compression framework utilizes the the I/O forwarding paradigm [3], which is a common tech-

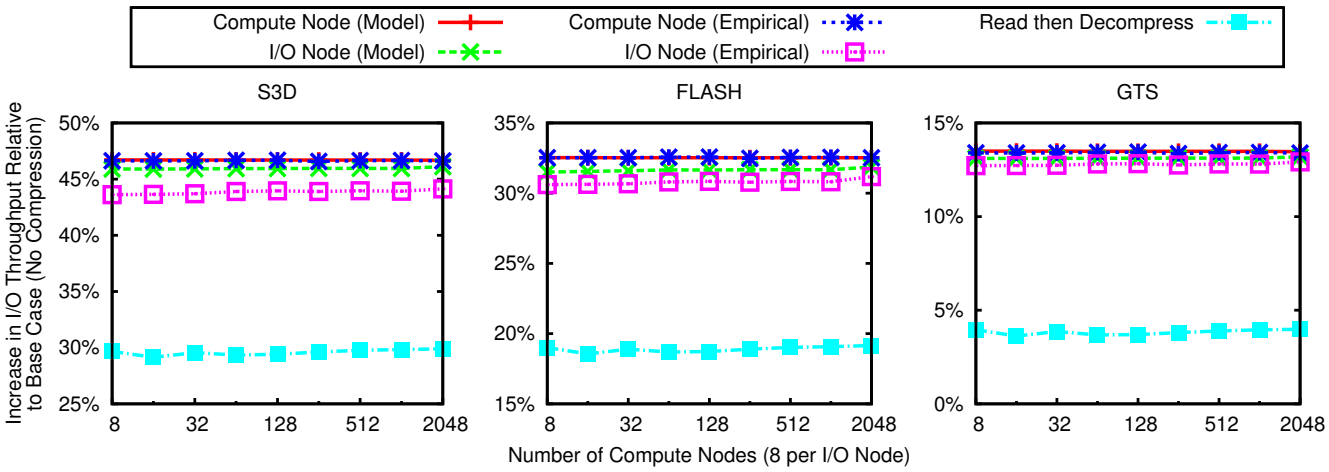


Figure 10: Model and empirical end-to-end read throughput versus number of compute nodes (weak scaling).

nique for alleviating the I/O bottleneck in super-computing environments, and is the subject of active research. The multithreaded ZOID architecture [16], developed under the ZeptoOS [4] project on the IBM BlueGene/P, is a state-of-the-art data staging system. LambdaRAM [32] is an asynchronous data staging system which mitigates WAN latency via dedicated staging nodes. IODC [26] is a portable MPI-IO layer implementing a caching framework wherein certain tasks, such as file caching, consistency control, and collective I/O optimization, are delegated to a small set of I/O Delegate nodes. Recent work in ADIOS includes the DataStager component [2], which focuses on I/O performance through data staging via network rate limiting and I/O phase prediction, and JITStaging [1], which provides a framework for placing data filter, analysis and organization code in the data pipeline to reduce overall time-to-data.

SCR [6] and PLFS [5] are well-known middleware approaches designed specifically for single (N-N) and shared (N-1) checkpointing, respectively. While SCR provides efficient checkpoints and improves system reliability by shifting checkpoint I/O workload to hardware better suited for the job, it is not suitable for applications that need process-global access to checkpoint files. Moreover, hardware and file system support is required to cache checkpoint files. PLFS, on the other hand, transparently rearranges shared checkpoint patterns into single patterns, thereby decreasing the checkpoint time by taking advantage of the increased bandwidth. However, this requires managing the overwhelming pressure resulting from the simultaneous creation of thousands of files within a single directory. Also, since PLFS is specifically a checkpoint file system and not a general purpose file system, certain usage patterns may suffer a significant performance hit [5].

Our approach differs in that we focus on optimizing data staging I/O throughput via compression and resource interleaving. Some recent work has examined compression in a data staging context [36]; however, only traditional compression algorithms are explored, which do not function well on the hard-to-compress scientific data we consider, and resource interleaving is not used to hide the compression and I/O synchronization costs.

Previous work on data deduplication can also be considered a form of compression, detecting and eliminating duplication in data with a goal to improving disk utilization. State-of-the-art deduplication systems include HYDRAstor [10], MAD2 [34], and others [14]. Deduplication can operate at either sub-file or whole-file scale; the relative merits of these aproaches have been explored [25]. Although these systems are scalable, provide a good deduplication efficiency, and attain near-optimal throughput for common filesystem data, they are unfortunately less effective when dealing with peta-scale scientific data. Unlike typical file system data, scientific data exhibits very few duplicate non-contiguous patterns, nullifying much of the effectiveness of the deduplication approach. Furthermore, the possibility of running compression *in-situ* remains desirable for performance reasons, which is not possible with filesystem-bound algorithms such as deduplication.

Table 4: Dataset evaluations for the best strategy of interleaving at the compute nodes.

Application	α	σ	$\mathbf{T_{comp}}$	$\mathbf{T_{decomp}}$	**Avg. write gain (%)***	**Avg. read gain (%)***
GTS †	0.25	0.527	90	414	11.26	13.40
FLASH †	0.25	0.019	149	127	30.87	32.53
S3D †	0.375	0.152	186	457	45.43	46.64
MSG	0.375	0.361	140	560	31.61	31.84
OBS	0.25	0.203	43	172	24.55	24.99
NUM	0.25	0.231	163	652	23.51	23.93

†Validated by experimental results. *Adjusted for model bias.

7. CONCLUSION

The I/O staging paradigm has arisen to cope with the growing gap between computing power and I/O bandwidth in current peta-scale HPC environments. While providing increased and more consistent performance, the sheer scale of the data necessitates lossless compression as a data reduction methodology, leaving the technical challenge of absorbing the costs of compression and overhead in parallel I/O performance on nonuniform chunk sizes.

To meet these challenges, we presented the ISOBAR hybrid compression-I/O framework. The ISOBAR preconditioner allows us to separate the high-entropy components of the data from the low-entropy components, forming independent streams that may be interleaved. The high-entropy components are sent across the network and to disk asynchronously while the low-entropy data is compressed, hiding the compression costs and fully utilizing all compute, network, and I/O resources. Placement of the compression routine itself is an important issue, so we implement a hybrid approach where the compression phase may be placed either on the compute nodes or the I/O nodes, trading off between aggregate compression throughput and leaving the compute nodes free to run the application at hand. Finally, each of the implementations are accurately modeled by a set of performance metrics, allowing a generalization of our methodology's performance past the experimental environment.

We demonstrated the efficiency of compression-I/O interleaving, improving end-to-end I/O throughput by predicted values ranging from 12 to 46% on datasets that are considered particularly hard-to-compress. We therefore believe that data compression can be used effectively within an HPC environment to help bridge the computational and I/O performance gap.

8. ACKNOWLEDGEMENTS

We would like to thank ORNL's and ANL's leadership class computing facilities, OLCF and ALCF respectively, for the use of their resources. We would also like to acknowledge the use of those scientific data sets at Flash Center for Computational Science. This work was supported in part by the U.S. Department of Energy, Office of Science and the U.S. National Science Foundation (Expeditions in Computing). Oak Ridge National Laboratory is managed by UT-Battelle for the LLC U.S. D.O.E. under contract no. DEAC05-00OR22725

9. REFERENCES

[1] 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. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 27–36. ACM, 2011.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. DataStager: Scalable data staging services for petascale applications. In *Proceedings of the 18th International Symposium on High Performance Distributed Computing*, HPDC '09, pages 39–48. ACM, 2009.

[3] N. Ali, P. Carns, K. Iskra, D. Kimpe, S. Lang, R. Latham, R. Ross, L. Ward, and P. Sadayappan. Scalable I/O forwarding framework for high-performance computing systems. In *International Conference on Cluster Computing and Workshops*, CLUSTER '09, pages 1–10. IEEE, 2009.

[4] P. Beckman, K. Iskra, K. Yoshii, and H. Naik. The ZeptoOS project. http://www.zeptoos.org/.

[5] J. Bent, G. Gibson, G. Grider, B. McClelland, P. Nowoczynski, J. Nunez, M. Polte, and M. Wingate. PLFS: a checkpoint filesystem for parallel applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 21:1–21:12. ACM, 2009.

[6] G. Bronevetsky and A. Moody. Scalable I/O systems via node-local storage: Approaching 1 TB/sec file I/O. Technical report, Lawrence Livermore National Laboratory, 2009.

[7] M. Burtscher and P. Ratanaworabhan. FPC: a high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58:18–31, 2009.

[8] M. Burtscher and I. Szczyrba. Numerical modeling of brain dynamics in traumatic situations - Impulsive Translations. In *Mathematics and Engineering Techniques in Medicine and Biological Scienes*, pages 205–211, 2005.

[9] J. H. Chen, A. Choudhary, B. Supinski, M. DeVries, E. Hawkes, S. Klasky, W. Liao, K. Ma, J. Mellor-Crummey, N. Podhorszki, R. Sankaran, S. Shende, and C. Yoo. Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science and Discovery*, 2(1):015001, 2009.

[10] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. HYDRAstor: a scalable secondary storage. In *Proccedings of the 7th Conference on File and Storage Technologies*, pages 197–210. USENIX Association, 2009.

[11] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *The Astrophysical Journal Supplement Series*, 131:273–334, Nov. 2000.

[12] J. Gailly and M. Adler. Zlib general purpose compression library. http://zlib.net/, Jan. 2012.

[13] Z. Gong, S. Lakshminarasimhan, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova. Multi-level layout optimization for efficient spatio-temporal queries on ISABELA-compressed data. In *Proceedings of the 26th IEEE International Parallel & Distributed Processing Symposium*, IPDPS '12, 2012.

[14] F. Guo and P. Efstathopoulos. Building a high-performance deduplication system. In *Proceedings of the 2011 USENIX Annual Technical Conference*, 2011.

[15] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems. In *Proceedings of the 27th IEEE*

International Conference on Data Engineering, ICDE '11, pages 1199–1208, 2011.

[16] K. Iskra, J. M. 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*, pages 153–162, 2008.

[17] Y. Jin, S. Lakshminarasimhan, N. Shah, Z. Gong, C. Chang, J. Chen, S. Ethier, H. Kolla, S.-H. Ku, S. Klasky, R. Latham, R. Ross, K. Schuchardt, and N. F. Samatova. S-preconditioner for multi-fold data reduction with guaranteed user-controlled accuracy. In *Proceedings of the IEEE 11th International Conference on Data Mining*, ICDM '11, pages 290–299, 2011.

[18] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. ISABELA-QA: Query-driven data analytics over ISABELA-compressed scientific data. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 31:1–31:11. ACM, 2011.

[19] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. Samatova. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Proceedings of the 17th International European Conference on Parallel and Distributed Computing*, Euro-Par '11, pages 366–379, 2011.

[20] J. Li, W.-K. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale. Parallel netCDF: a high-performance scientific I/O interface. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, page 39. ACM, 2003.

[21] J. Liu, J. Wu, and D. Panda. High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming*, 32:167–198, 2004.

[22] J. 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*, CLADE '08, pages 15–24. ACM, 2008.

[23] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan. Adaptable, metadata rich IO methods for portable high performance IO. In *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, IPDPS '09, pages 1–10, 2009.

[24] 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 *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–12, 2010.

[25] D. T. Meyer and W. J. Bolosky. A study of practical deduplication. *ACM Transactions on Storage*, 7(4):1–20, Feb. 2012.

[26] A. Nisar, W.-K. Liao, and A. Choudhary. Scaling parallel I/O performance through I/O delegate and caching system. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '08, pages 1–12, 2008.

[27] M. Polte, J. Lofstead, J. Bent, G. Gibson, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, M. Wingate, and M. Wolf. ...and eat it too: high read performance in write-optimized HPC I/O middleware file formats. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 21–25. ACM, 2009.

[28] J. M. Prusa, P. K. Smolarkiewicz, and A. A. Wyszogrodzki. Simulations of gravity wave induced turbulence using 512 PE CRAY T3E. *International Journal of Applied Mathematics and Computational Science*, 11(4):883–898, 2001.

[29] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82, July 1990.

[30] E. R. Schendel, Y. Jin, N. Shah, J. Chen, C. Chang, S.-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. ISOBAR preconditioner for effective and high-throughput lossless data compression. In *Proceedings of the 28th International Conference on Data Engineering*, ICDE '12. IEEE, 2012.

[31] P. Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the 2003 Linux Symposium*, pages 400–407, July 2003.

[32] V. Vishwanath, R. Burns, J. Leigh, and M. Seablom. Accelerating tropical cyclone analysis using LambdaRAM, a distributed data cache over wide-area ultra-fast networks. *Future Generation Computer Systems*, 25(2):184–191, 2009.

[33] W. X. Wang, Z. Lin, W. M. Tang, W. W. Lee, S. Ethier, J. L. V. Lewandowski, G. Rewoldt, T. S. Hahm, and J. Manickam. Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments. *Physics of Plasmas*, 13:092505, 2006.

[34] J. Wei, H. Jiang, K. Zhou, and D. Feng. MAD2: a scalable high-throughput exact deduplication approach for network backup services. In *Proceedings of the IEEE 26th Symposium on Mass Storage Systems and Technologies*, MSST '10, pages 1–14, 2010.

[35] T. A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.

[36] B. Welton, D. Kimpe, J. Cope, C. Patrick, K. Iskra, and R. Ross. Improving I/O forwarding throughput with data compression. In *International Conference on Cluster Computing*, CLUSTER '11, pages 438–445. IEEE, 2011.

[37] M. Yang, R. E. McGrath, and M. Folk. HDF5 - a high performance data format for earth science. In *21st International Conference on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology*, 2005.