Load-aware Elastic Data Reduction and Re-computation for Adaptive Mesh Refinement

Mengxiao Wang[1], Huizhang Luo[2], Qing Liu[2], and Hong Jiang[1]

[1] Department of Computer Science and Engineering, University of Texas at Arlington
[2] Department of Electrical and Computer Engineering, New Jersey Institute of Technology

***Abstract*—The increasing performance gap between computation and I/O creates huge data management challenges for simulation-based scientific discovery. Data reduction, among others, is deemed to be a promising technique to bridge the gap through reducing the amount of data migrated to persistent storage. However, the reduction performance is still far from what is being demanded from production applications. To this end, we propose a new methodology that aggressively reduces data despite the substantial loss of information, and re-computes the original accuracy on-demand. As a result, our scheme creates an illusion of a fast and large storage medium with the availability of high-accuracy data. We further design a load-aware data reduction strategy that monitors the I/O overhead at runtime, and dynamically adjusts the reduction ratio. We verify the efficacy of our methodology through adaptive mesh refinement, a popular numerical technique for solving partial differential equations. We evaluate data reduction and selective data re-computation on Titan, using a real application in FLASH and mini-applications in Chombo. To clearly demonstrate the benefits of re-computation, we compare it with other state-of-the-art data reduction methods including SZ, ZFP, FPC and deduplication, and it is shown to be superior in both write and read speeds, particularly when a small amount of data (e.g., 1%) need to be retrieved, as well as reduction ratio. Our results confirm that data reduction and selective data re-computation can 1) reduce the performance gap between I/O and compute via aggressively reducing AMR levels, and more importantly 2) can recover the target accuracy efficiently for AMR through re-computation.**

I. Introduction

In recent years, the advances in high-performance computing (HPC) are at an accelerated rate. However, the performance of storage systems is still lagging far behind, despite the emergence of new storage technologies, such as solid-state drive (SSD), and non-volatile memory. This trend has continued on the next-generation leadership class systems. For example, for Summit [1] at Oak Ridge Leadership Computing Facility (OLCF), it is designed to have 5-10X more flops with, however, a modest 2.5X higher I/O throughput than Titan [2], the current production system at OLCF. Suggested by the Amdahl's Law, the improved flops without the commensurate improvement in I/O can limit the overall speedup, and therefore must be addressed in order to fully exploit the compute capabilities of these systems. The HPC communities have been striving to explore solutions across the software and hardware stacks to mitigate the growing disparity between compute and I/O. This disparity is further exacerbated by the fact that I/O interference from applications running concurrently.

Very recently, there has been renewed interest in data reduction, and its importance in exascale science production has been well recognized. The goal of data reduction is to reduce the volume and velocity of data from applications before data are migrated to the persistent storage. Prior work in this area include both lossless and lossy compression. Lossless data compression, e.g., Huffman encoding [3], LZ77 [4], FPZIP [5], GZIP [6], data deduplication [7], can guarantee no information loss, however, it only achieves a modest reduction ratio due to the typical high entropy of scientific data [8]. In particular, data deduplication, commonly used for non-HPC data, retains unique data and discard redundant data by calculating and checking a cryptographically secure hash-based fingerprint for each data chunk. A recent study [9] shows that data deduplication can only reduce 20% to 30% data of scientific applications on HPC systems, which is far from being useful in production. In contrast, lossy data compression, e.g., ISABELA [10], ZFP [11] and SZ [12], improves the compression ratio at a cost of information loss. While these lossy compressors achieve higher compression performance, the reduction ratio is still far from the 100X reduction that is demanded from large HPC applications. Another key concern is the negative impact of reduced accuracy on knowledge discovery, which is not well-studied and may be unacceptable in certain cases.

This work aims to address the aforementioned challenges by more aggressively reducing data coming out of HPC applications according to the storage overhead, but with the ability to maintain the original data accuracy. By and large, our approach is motivated by three key observations. First, HPC simulations are governed by well-established laws of physics and mathematics, which can always be re-computed in contrast to perishable data (e.g., observational, and social data). For example, solving the Navier-Stokes equation for modeling viscous fluid under the same parameter space, the same results should be re-produced. Second, given the general trend that compute is becoming increasingly cheaper as compared to I/O, it presents an opportunity of trading compute for I/O. Ideally what will be sent to the storage system should be the minimum to reduce the storage footprint, and the full data can be computed on demand. Third, HPC storage is shown to have significant I/O variability [13], and therefore data reduction needs to be load-aware so that the precious resources on HPC systems can be well utilized. Especially for

978-1-7281-4409-2/19/$31.00

I/O-intensive applications, I/O performance degrading caused by interference becomes an issue to be addressed on demand as the gap between compute and I/O increasing. The data re-computation is selective in nature, since scientific discovery commonly examines a subset of data in physical problem, e.g., extracting an isosurface or a 2D plane. In re-computation, a subset of a spatial region or a subset of time steps is re-computed on-the-fly to construct the accuracy demanded. Analogous to the idea of using cache to bridge the gap between on-chip registers and DRAM, this approach provides an illusion of a fast and large storage medium to applications, and high-accuracy data can be accommodated with fast access speed. In essence, this methodology is designed to lower the I/O burden of those data that users are less likely to examine. Finally, the subset of data to be re-computed is determined and driven by users, and therefore the execution of data analytics is more targeted and the efficiency can be greatly improved.

We implement this idea on four adaptive mesh refinement (AMR) applications from Chombo [14], an open source AMR framework, and FLASH [15], a production astrophysics code. Herein, the data reduction is achieved by discarding a certain number of levels of AMR data product, and the reduction ratio can be dynamically adjusted online based upon I/O overhad and the performance goal of applications. During post-processing, the original accuracy can be recovered by further refining meshes over selected spatiotemporal slices.

The rest of the paper is organized as follows. We introduce the background and motivation in Section II. The design and the detailed implementation are described in Section III. The performance results are presented in Section IV. In Section V, related work is demonstrated, along with conclusions in Section VI.

II. Background and Motivation

Scientific applications running on HPC platforms are all applying discrete computing and modeling due to the nature features. Since the complexity in practical environment, scientific computing and modeling need to solve partial differential equations instead of simply applying physical and mathematical equations. Finite element method (FEM) subdivides a problem domain in to small part and then assemble them together while minimizing the associated error via computation to obtain the target function, which has been developed to be the primary method in scientific applications to model and simulate the problems. Therefore, as a transformation of FEM, AMR has the advantages in flexibility and scalability through tracking features with adaptive mesh and providing adequate higher spacial and temporal resolution, which is widely used in computational fluid dynamics (CFD), astrophysics, climate modeling, turbulence, mantle convection modeling, combustion, biophysics and many more areas.

A. AMR Basics

AMR provides adaptive numerical methods to refine the spatiotemporal resolution of grids in regions where there are interesting physics. Fig. 1 shows an example of AMR data

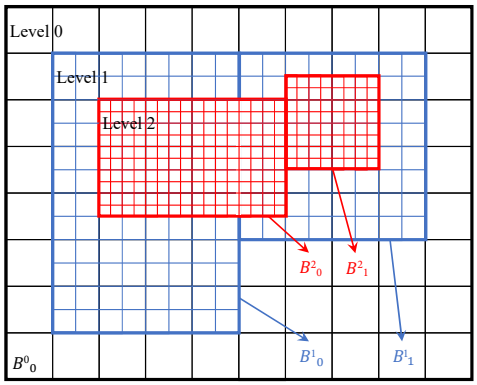

Fig. 1. AMR meshes in three levels. The refinement ratio is 2 in both dimensions. The area with the coarsest mesh is the original problem domain. The area with finer mesh (in blue) is the region refined at level 1, and the area the finest mesh (in red) is the region further refined at level 2. $B^1{}_0$ and $B^1{}_1$ are the grids at level 1. $B^2{}_0$ and $B^2{}_1$ are the grids at level 2, are computed based upon $B^1{}_0$ and $B^1{}_1$.

output consisting of three levels of fidelity. In a time-dependent AMR, a new level of grid can be computed in the following steps:

Step 1: Estimate the point-wise local error in the current grid, e.g., using the Richardson extrapolation [16].

Step 2: Identify those grid points that yield an error that is higher than the prescribed threshold.

Step 3: Generate a new set of finer grids that can cover the flagged grid points efficiently. Finer grids are identified recursively until either the maximum level of refinement is reached, or the local error bound is satisfied. In particular, for time evolution, the error estimation and regridding are applied to both the temporal and spatial dimensions.

Data reduction can be executed with regard to resolution, precision, and fidelity, respectively. To reduce the resolution, a mesh can be decimated by a given factor into a smaller mesh. To reduce the precision, the number of bits used to represent a floating-point number is reduced by cutting down the mantissa. To reduce the fidelity of data, the degree of freedom involved in solving a model is reduced, e.g., by reducing the number of variables. Recognizing that the reduction ratio of precision reduction is limited which is usually used in lossy compressors, and fidelity reduction is in general domain and application dependent, we choose the resolution reduction, which in general works for all mesh-based applications. Therefore, we choose AMR applications in this work which have general usage in scientific computing and the level-structured output, where reduction and re-computation can be done without sophisticated physics or mathematics related changes. Namely, 1) the tree-like block-structured data can be easily reduced by cutting blocks, and 2) a child grid (e.g., level 2) can be computed based upon its immediate parent grid (e.g., level 1). Thus, AMR makes data reduction and re-computation easy to be implemented.

In this paper, we select four AMR applications: *PoissonNode* is a node-centered Poisson solver; *Sedov* is an

astrophysical fluid dynamics model to simulate blast wave; *AMRGodunov* is a CFD solver for compressible flow; *AMRINS* is a simulator for incompressible flow.

B. Motivation

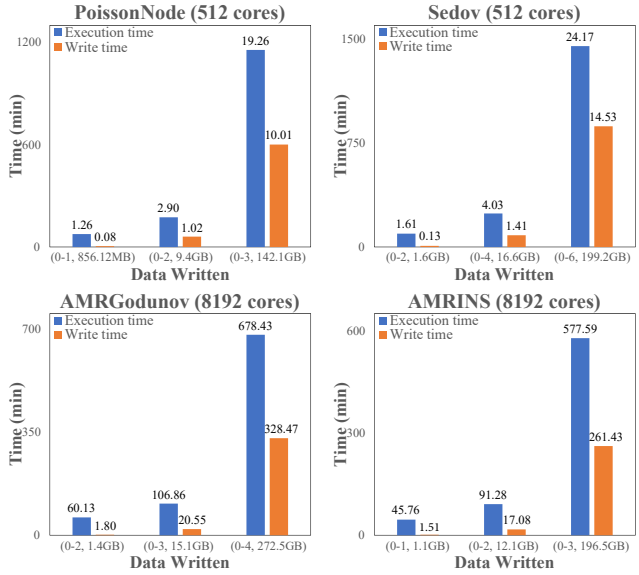

Fig. 2. Application performance vs. the number of AMR levels. The notation of (X-Y, Z) means that the data from level X to level Y are calculated and written to the storage device with size of Z. For example, in the figure of *PoissonNode*, (0-1, 856.12MB) represents the data of level 0 to level 1 are computed and written, and the data size is 856.12MB.

To quantitatively understand the I/O overhead in applications and the potential of using data reduction to improve application performance, we run four AMR applications, i.e., *PoissonNode*, *Sedov*, *AMRGodunov*, and *AMRINS*, which involve both time-dependent and time-independent calculations. We measured the execution and I/O time under various number of refinement levels, as shown in Fig. 2. From the results, we can find two observations: 1) As the number of levels increases, the four applications become quickly I/O-bottlenecked. For *PoissonNode*, at 512-core with a single storage device, a ratio reflecting the full Titan [2] configuration, the time spent on writing four levels accounts for 52% of the total execution time, while that of writing two levels only takes 6% of the execution time; 2) The I/O overhead is sensitive to data fidelity. For example, in the case of *PoissonNode*, writing data without the two highest levels, i.e., (0-1, 856.12MB), can reduce the I/O overhead to 6%. With the two additional AMR levels, *PoissonNode* quickly transits from compute-intensive to I/O-intensive, and this clearly demonstrates the sensitivity of application performance to data fidelity.

III. Design and Implementation

The heart of our work is a strategy that utilizes data reduction in an aggressive and adaptive manner, by adjusting the amount of data that are migrated to storage, with the awareness of storage load. The caveat of degraded fidelity is resolved by re-computing the original fidelity, if needed, during post-processing. As such, our approach takes advantage of the strengths from both lossy compression (i.e., high reduction ratio) and lossless compression (i.e., no information loss). This creates an illusion to end users that accessing high-fidelity data from storage systems is dramatically faster, without increasing the physical bandwidth of storage hardware.

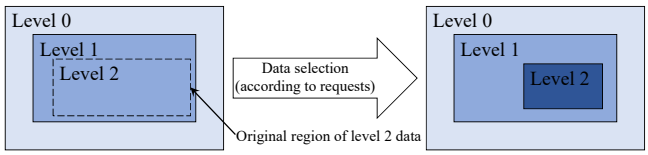

Fig. 3. An example of spatially data selection.

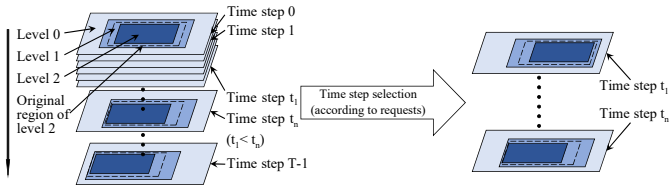

Fig. 4. An example of temporally data selection.

A. Load-aware Data Reduction

In a production HPC environment, I/O interference can occur when multiple applications utilize the same storage devices concurrently. As a result, the I/O overhead can be fairly random and may grow to a degree that is beyond the performance goal. Therefore, at each timestep we need to make runtime adjustments of data reduction so that the performance goal can be achieved, while retaining most data product. To achieve this goal, we implement two different strategies.

1) Semi-dynamic Data Reduction: Semi-dynamic data reduction consists of two data reduction processes, static data reduction and dynamic data reduction. To reduce data, we first select a set of high levels of AMR grids to be eliminated from further refinement, e.g., in Fig. 1, data in $B^2{}_1$, based upon the performance goal of an application and the system load, which is considered as the process of static data reduction. If the current I/O overhead is deemed as high (e.g., 15%) as a result of a transient peak of storage load, we further discard a set of levels to reduce the data volume.

For unknown I/O performance of the storage system, at each timestep, both the per-step and accumulated I/O overhead are calculated to monitor the dynamics of the storage system load, by recording the execution and write time of the current step, and the accumulated execution and write time. In particular, the purpose of controlling the per-step overhead is to avoid overloading the storage system during the congested period, while that of controlling the accumulated overhead is to attain the overall performance goal of an application. If either of the two metrics exceeds a given threshold, we further reduce the

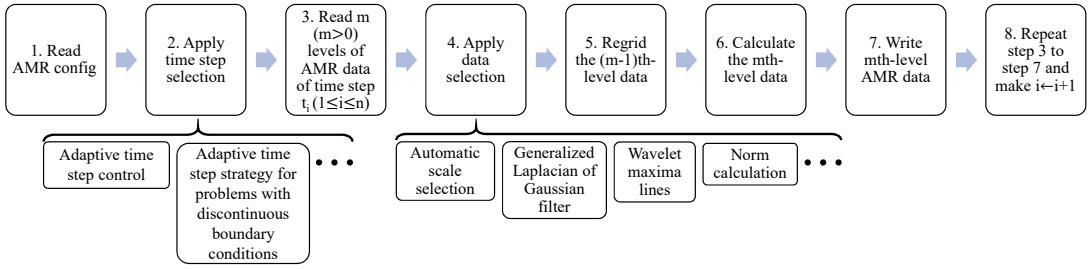

Fig. 5. AMR data re-computation.

AMR levels, e.g., data in $B^2{}_0$ are not computed and written to storage in the subsequent timestep.

For I/O performance of the storage system we can predict, according to the compute phase and write phase of each timestep of applications, the write time can be estimated to decide whether the timestep needs further reducing data or not. If I/O bursts occur within the write phase of each timestep, data generated in the timestep will be further reduced, which means both computation and write are reduced, e.g., data in $B^2{}_0$ are not computed and written to storage in the current timestep. For HPC platforms, applications typically run repeatedly in a period of time, so similar I/O patterns can be seen no matter in a long-term or short-term observation. In other words, the past I/O behavior of the storage systems have high likelihood to reflect the future I/O behavior for prediction use on HPC platforms.

Semi-dynamic data reduction is designed and implemented to make the applications' I/O consumption controlled while the system is relatively busy. It is aimed to increase the efficiency of the system instead of I/O taking a significant part of total machine time and increasing runtime. By reducing the amount of data output to the storage devices, semi-dynamic data reduction can restrain the total I/O time of applications and bring impressive total time saving. Meanwhile, faster I/O also benefits other applications with the same limited I/O resources.

2) Full-dynamic Data Reduction: Full-dynamic data reduction is a strategy to adaptively control the amount of data generated and written to the storage according to the I/O performance of the storage system. When the storage system is idle or slightly used with minor I/O operations, it is much under-utilized which is a waste in a concept of energy and resource. Therefore, during the period of time, the system should be fully utilized by running computation timesteps and migrating data generated. If the time is long enough, all data produced by the timesteps will be written without any data reduction and any needs for re-computation. However, if the time is not enough to migrate all data, we will dynamically reduce data to avoid over-utilization of the system, which the amount of data reduced without computation is decided by the time length of idle status of the storage system and this part of data can be recovered through re-computation. In this manner for data reduction, the utilization of the storage system can be maximized to output data generated by applications. Data with high accuracy can be produced as much as possible for users' demand and the re-computation for other details of the data can be minimized in post data processing, where the overhead of re-computation is mitigated. However, for applications may generating large amount of redundant data, the reduction ratio is much smaller than semi-dynamic and simple data reduction, which results in large burden to the capacity of the storage system and high I/O consumption of the HPC platform with low efficiency.

B. Data Selection in Post-Processing

As aforementioned, to deal with information loss as a result of data reduction, a higher accuracy is required to be re-computed during post-processing. However, if the entire dataset needs to be re-computed to the original accuracy, there is no performance gain using our methodology. Therefore, understanding the impact of data selection to the overall performance is important. Prior work has documented that retrieving and analyzing only a subset of a problem domain is common [17], if not always. Overall data selection aims to select data of interest both spatially and temporally, and we choose a few of them to evaluate data re-computation in this work. The classic spatially selection methods include, e.g., *bounding box* (BB), *automatic scale selection* (ATSS) [18], and *norm calculation* (NC). Bounding box is a simple selection method that retrieves a rectangular or a cuboid region of data chosen by users. ATSS applies the size of local scales to extract features. NC calculates the spatial distance between two points to cluster the data points with similar characteristics. Fig. 3 illustrates a bounding box selection on top of a level 3 grid in an AMR output. Similarly, Fig. 4 shows the temporal selection in a time evolution alongside spatial selection, in which n timesteps are selected out of T simulation outputs. The classic temporal selections include *time step selection* (TSS), which selects timesteps at a certain interval, *adaptive time step control* [19], termed as ATSC1, and a variant of adaptive time evolution strategy [20], termed as ATSC2. ATSC1 uses an explicit Runge Kutta method to control timesteps within a prescribed tolerance. In contrast, ATSC2 applies the finite-difference adaptive moving grid to control timesteps with discontinuous boundary conditions.

C. Re-computation in Post-Processing

After data selection, the selected spatiotemporal regions will be recovered to a higher fidelity according to the users'

requirement, and the process of selection and re-computation will iterate until the resulting fidelity is satisfactory. For time-dependent applications, the steps of re-computation are listed in Fig. 5. A typical post-processing data analysis is described as follows: The re-computation starts with reading AMR configurations, including refinement ratio, error threshold, and etc., and the reduced data from the persistent storage, e.g., level 0 to level $(m-1)$. Subsequently, the temporal selection is applied to filter timesteps, followed by spatial selection to identify sub-regions for further re-computation. Then a new set of finer grids at level m is computed, in a way that is similar to how a regular AMR grid is computed (Section II-A). The regenerated level m is analyzed by end users, who then decide whether a further refinement is needed for level $m+1$. For example in Fig. 1, the end users may perform analysis on level 0 and 1. Although level 1 data have a lower fidelity than level 2, the analysis can provide insights and guidance to determine whether or where the subsequent re-computation in boxes ${B^2}_0$ and ${B^2}_1$ at level 2 is needed. If the accuracy of level 1 suffices or it does not contain important physics as determined by the users, the data analysis terminates.

IV. Performance Evaluation

This section aims to quantitatively understand the efficacy of elastic data reduction and re-computation, and verify whether our design goal of providing a fast data store with high-accuracy data is possible on the current leadership class systems. Our approach is implemented in Chombo, a widely used adaptive mesh refinement package, and we test three AMR applications, *PoissonNode*, *AMRGodunov*, *AMRINS*, and a production application *FLASH* (*Sedov*). Among these four applications, *PoinssonNode* and *Sedov* are time-independent applications, while the rest are time-dependent. We use Cray-Pat, a performance analysis tool, to measure the compute and I/O timings.

A. Testbed

We ran all experiments on Titan, the current leadership class system at OLCF that provides computational and storage resources for the largest computational applications. It features a hybrid architecture that consists of 18,688 compute nodes, with each containing two 16-core AMD Opteron processors and a NVIDIA Tesla K20 GPU accelerator, with the total memory capacity of 710 TB. Titan utilizes a center-wide Lustre file system to sustain the I/O outputs from large simulations as well as to accommodate data analysis and visualization needs. In total, Titan can achieve a peak performance of 27 PF with 1TB/s I/O bandwidth. For all the results in this work, one Lustre storage target is used to store AMR data.

B. Load-aware Elastic Data Reduction

We evaluate the effect of load-aware data reduction. In order to accurately examine the impact of the I/O interference for a running application, we measure the I/O trace of the Lustre storage device around two month. Then, we apply long short-term memory (LSTM) algorithm to learn the I/O performance of HPC storage system on Titan and try to predict the I/O performance. LSTM is a popular recurrent neural network technique, which can keep the impact of system status for a relatively long time to influence the current system status. Therefore, we can predict the occurrence of the I/O interference in the future hours. Fig. 6 shows that the predicted I/O trace and the real I/O trace we monitored and that the

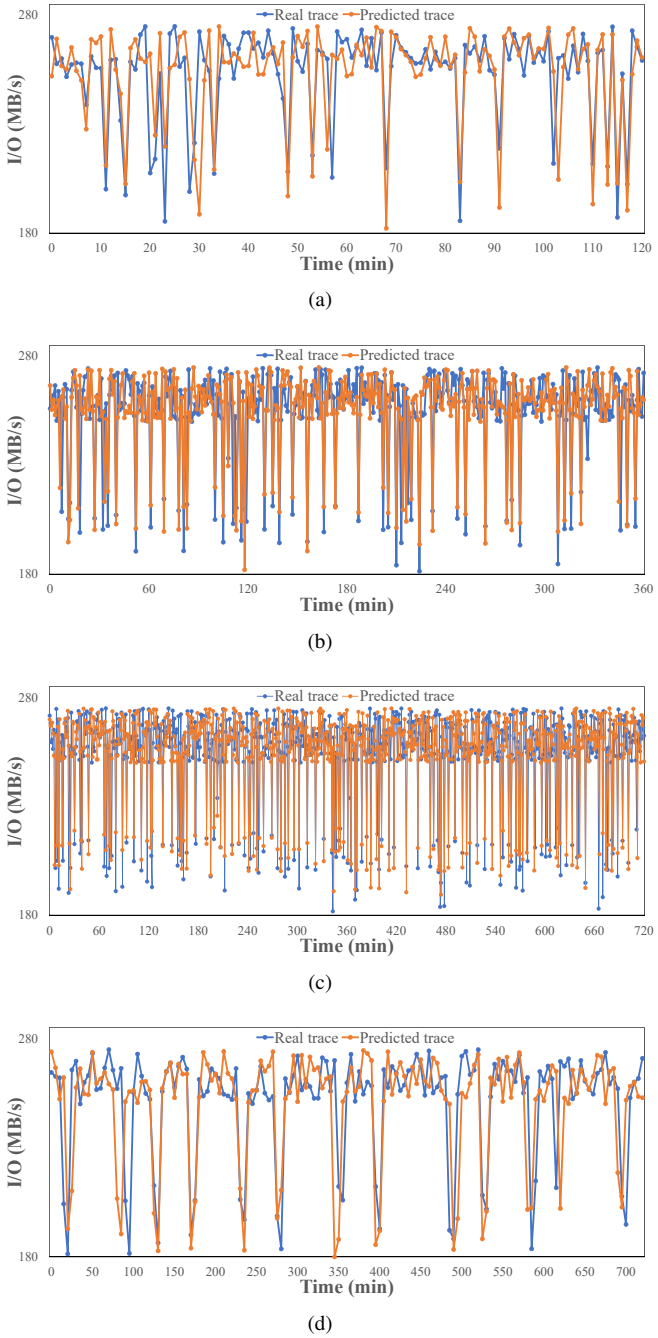

Fig. 6. I/O prediction. (a) - (c) shows per-minute I/O prediction for 2 hours, 6 hours and 12 hours, respectively. (d) shows per-5-minute I/O prediction for 12 hours. The orange line represents the predicted I/O trace of the storage device we monitored meanwhile the blue lines shows the real I/O trace of that.

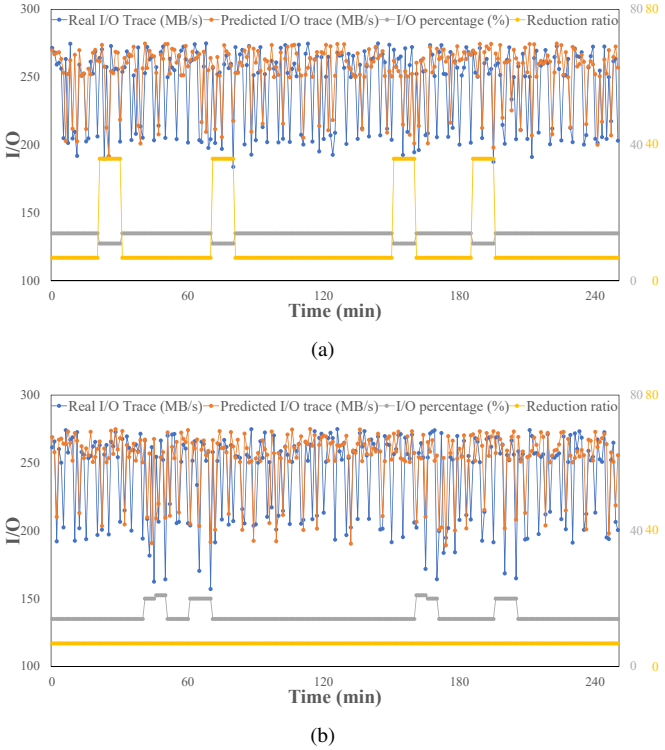

Fig. 7. Performance comparison of *AMRGodunov* applying (a) semi-dynamic data reduction and that (b) without dynamical data reduction according to the I/O prediction. The orange line represents the predicted I/O trace of the storage device we monitored meanwhile the blue lines shows the real I/O trace of that. The gray line indicates the I/O percentage changes for all timesteps and the reduction ratio of each timestep is shown by the yellow line.

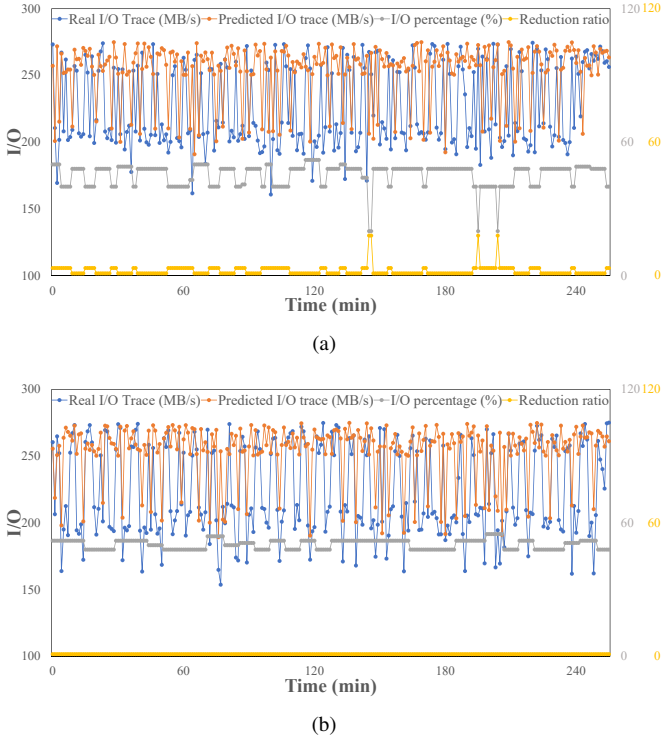

Fig. 8. Performance comparison of *AMRGodunov* (a) with applying full-dynamic data reduction and (b) without applying full-dynamic data reduction according to the I/O prediction.

overall I/O prediction accuracy of LSTM is around 80%.

1) Semi-dynamic Data Reduction: To maintain the I/O percentage of an application execution under its I/O budget, e.g., 15%, high levels of AMR data are eliminated directly and low levels of AMR data are determined to be further reduced when I/O interference occurs. Otherwise, the I/O will be an issue to degrade the performance of the application and decrease the efficiency of the storage system.

Fig. 7(a) shows *AMRGodunov* I/O performance with semi-dynamic data reduction according to the I/O prediction and Fig. 7(b) shows the I/O performance only with high levels of AMR data reduction. While applying semi-dynamic data reduction strategy, we can dynamically further reduce data according to the I/O prediction with statically data reduction at the beginning of the execution of the application. As shown in Fig. 7, data of 16 of 100 timesteps are further reduced based on the I/O prediction, which the reduction ratio of the 16 timesteps changes dramatically. However, without dynamic data reduction of low levels according to the I/O prediction, the reduction ratio of each timestep are constant and only 66 timesteps are executed in 250 minutes of which the I/O percentage of 16 timesteps increases due to the I/O interference in the storage system.

2) Full-dynamic Data Reduction: When the system is relatively idle or all the generated data are very important, full-dynamic data reduction plays an important role by outputting the data with dynamically reducing data while I/O interference occurs to avoid system over-utilization.

Fig. 8 shows the performance difference between *AMRGodunov* with and without applying full-dynamic data reduction. With full-dynamic data reduction, the reduction ratio and the I/O percentage of each timestep of the application dynamically change according to the I/O prediction. When the storage system has few I/O operations, the application writes data with high accuracy to the storage system in order to reduce the cost of re-computation for the recovery use. When the storage system is busy or I/O bursts occur, we compute and write data with relatively high accuracy, which would increase data recovery of re-computation for future analysis use. In this data reduction strategy, we can obtain data of 64 timesteps of *AMRGodunov* with around 250-minute execution time. However, data of only 37 timesteps can be generated by *AMRGodunov* without full-dynamic data reduction in around 250-minute execution time, which none of data are reduced and I/O percentage of each timestep is around 50%.

C. Data Selection and Re-computation

Data selection is crucial for identifying the right subset of data to re-compute.If data selection is not effective in narrowing down either the regions or time slices, re-computation will result in equal storage overhead as other approaches. To understand the advantages of data selection, we apply classic data selection algorithms to both time-independent and time-

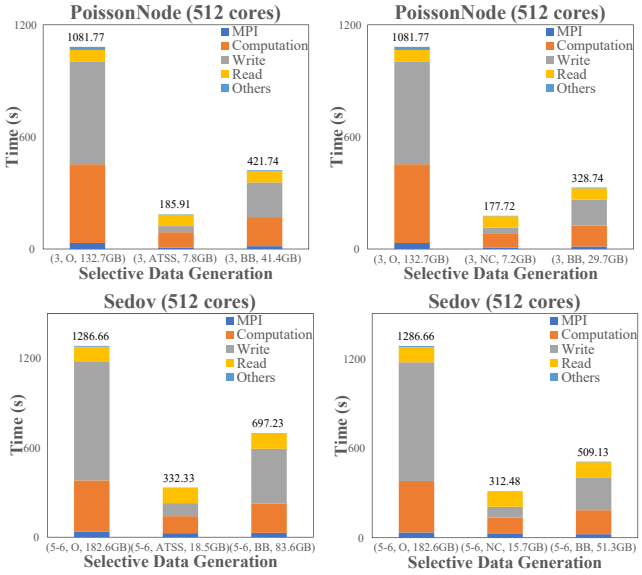

Fig. 9. Comparison of spatial selections for time-independent applications. The notation of (X, Y, Z) means that the data of level X are selectively re-computed with selection strategy Y, and then are written to the storage device with a volume of Z. "O" represents the case that all original data are selected. In each plot, BB is set to a mimimal rectangular area that can cover the regions selected by Y for a fair comparison. For example, in *PoissonNode*, BB generates the smallest box to contain all the data selected by ATSS.

dependent applications and perform re-computation. Fig. 9 shows the time-independent applications, i.e., *PoissonNode* (top two plots) and *Sedov* (bottom two plots), applied with spatial data selection algorithms and re-computation. In each plot, we compare ATSS or NC to BB and the original data without data selection (right most bar). For a fair comparison, the region of a BB is chosen to be the smallest box that can

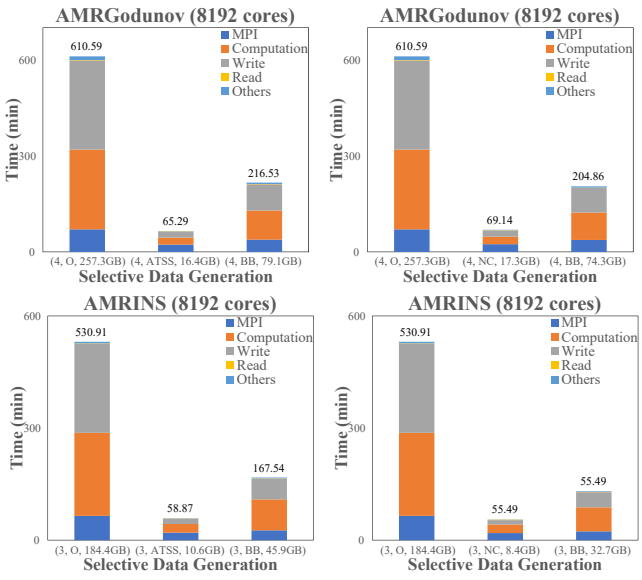

Fig. 10. Comparison of spatial data selections for time-dependent applications.

cover all the points selected by ATSS or NC. It is clear in Fig. 9 that ATSS and NC are more fine-grained selections than BB. In general, the volume of data selected is related to an application itself and the selection method. Fig. 10 shows the impact of spatial data selection criteria used in AMRGodunov and AMRINS, which are time-dependent applications. For time-dependent applications, each timestep can be regarded as a time-independent application, therefore, applying spacial data selection on these two applications indicates the same experiment results. For temporal data selection, the comparison between re-computation with different selections can be found in Fig. 11. The effectiveness of only applying the adaptive time step controls (ATSC1 and ATSC2) are not as obvious as that of using TSS with certain intervals due to the non-trivial correlation between the consecutive time steps in these two applications. However, TSS may lose important physics when capturing the transient states of a scientific process.

Fig. 11. Comparison of temporal selections for time-dependent applications.

D. Data Reduction and Re-computation vs. State-of-the-art

In this part, we aim to understand whether the re-computation brings performance advantages over other state-of-the-art solutions. For easy analysis, we apply static data reduction, which is directly reducing high levels of AMR data of each application without further data reduction while I/O interference occurs. Table 1 compares re-computation with some of the best lossy compressors (SZ, ZFP), and lossless compressors (FPC, DEDUP), for all four applications. We measured the compression time, decompression time as well as the compression ratio. For lossy compressors, SZ and ZFP are selected since they are shown to be best compressors across 13 datasets [12]. For lossless compressors, FPC is shown to be superior in compression ratio for 5 out of 13 datasets, and be comparable to the rest, and achieve the best throughput among all lossless compressors. DEDUP is chosen because deduplication as data reduction technique has been widely used in storage systems, including in some as a standard feature. Through Table 1, it is evident that the compression ratios of FPC and DEDUP are significantly lower than others. This is mainly due to the fact that scientific data are inherently noisy and random with very high information entropy, which makes lossless compressors less effective. More importantly, this observation suggests, even in AMR datasets where there is substantial information redundancy between AMR levels, the

TABLE I
RE-COMPUTATION VS. STATE-OF-THE-ART COMPRESSORS

Application	Method	Total time (execution+compression+write)	Data retrieval (read+decompression)				Compression ratio
PoissonNode (142.1GB)	SZ	554.9s+69.7s+78.4s	114.1s+10.6s				7.8
	ZFP	552.6s+110.9s+245.4s	345.5s+109.6s				2.5
	Re-computation	**113.2s+0s+60.9s**	**1% Decompression**	**5% Decompression**	**10% Decompression**	**20% Decompression**	**15.1**
			59.9s+11.4s	**60.3s+49.5s**	**60.8s+70.1s**	**59.6s+127.4s**	
	FPC	553.3s+59.9s+500.9s	707.9s+16.8s				1.2
	DEDUP	550.8s+25.8s+468.2s	634.5s+77.3s				1.3
Sedov (199.2GB)	SZ	578.8s+78.6s+123.1s	171.22s+12.5s				7.2
	ZFP	580.4s+104.6s+226.5s	311.0s+114.6s				3.9
	Re-computation	**156.9s+0s+84.8s**	**1% Decompression**	**5% Decompression**	**10% Decompression**	**20% Decompression**	**12**
			101.9s+20.5s	**101.6s+87.3**	**100.4s+126.4s**	**102.3s+221.1s**	
	FPC	573.3s+24.9s+673.5s	900.1s+22.8s				1.3
	DEDUP	575.6s+36.5s+668.7s	896.1s+102.4s				1.3
AMRGodunov (272.5GB)	SZ	350.1min+141.3s+93.2min	5.2min+30.3s				3.4
	ZFP	348.3min+195.2s+82.4min	4.7min+61.3s				3.8
	Re-computation	**85.3min+0s+18.6min**	**1% Decompression**	**5% Decompression**	**10% Decompression**	**20% Decompression**	**18**
			63s+10.1min	**67.2s+43.5min**	**73.8s+72.4min**	**77.4s+132.4min**	
	FPC	348.9min+38.1s+283.6min	16.2min+31.5s				1.1
	DEDUP	351.8min+48.7s+237.7min	13.6min+138.2s				1.3
AMRINS (196.5GB)	SZ	316.4min+81.3s+26.6min	1.4min+15.2s				9.2
	ZFP	318.2min+122.1s+54.3min	2.8min+110.1s				4.5
	Re-computation	**72.2min+0s+16.1min**	**1% Decompression**	**5% Decompression**	**10% Decompression**	**20% Decompression**	**16.3**
			53.7s+9.2min	**52.2s+41.3min**	**55.1s+67.1min**	**54.3s+110.4min**	
	FPC	315.7min+26.8s+230.9min	11.4min+23.5s				1.1
	DEDUP	313.6min+35.1s+192.1min	9.6min+95.9s				1.3

best lossless compressors are still sensitive to the minuscule difference between AMR levels, resulting in subpar compression ratios. Compared to lossy compressors, re-computation achieves a compression ratio that is 1.5-5X and 3-6X higher than that of SZ and ZFP, respectively. With the philosophy that aggressively decimates AMR levels, it by design leads to a higher reduction ratio.

For domain scientists, they have need of frequent data access for post-analysis and in other data compression techniques data compression is also an essential process before storing data for post-processing to reduce I/O burden. That is why compression time is presented as a part of total time as well as execution time and write time of applications. In addition, the total of decompression time and read time of accessing the essential data is regarded as data retrieval time. In the column of data retrieval, our approach of re-computation shows 1%, 5%, 10% and 20% decompression, which represent that 1%, 5%, 10% and 20% of high-level data users interested in were re-computed by our approach, respectively, while other methods cannot do. Through the results in this column, it can be observed that re-computation has much less data accessing time than other data compression techniques due to the higher compression ratio achieved by our approach. However, for decompression time, re-computation is increasingly lower than others with the percentage of high-level data increasing, which results from that our approach selectively re-computed the data to retrieve the information of users' interest, especially for time-dependent applications. In the column of total time, it can be seen that our approach has zero overhead for compressing data since our approach eliminated high levels of data to retain the essential low levels of data for subsequent analysis during applications run, which is much better than other compression techniques. Meanwhile, with high levels of data reduced, re-computation also reduced time costs for execution of the applications comparing to other methods needing to compute the original datasets. Moreover, due to the high compression ratio of our approach, the write time of re-computation is also much less than other techniques. Therefore, through the results in the column of total time, re-computation saves around 6X of total time for the four applications. Since domain scientists care the time cost from end to end more than the time cost of data compression and decompression, our approach compares total time plus data retrieval as a whole part to other methods. In this term, re-computation still has a better performance than other compression techniques due to without calculating and storing data that users are not to examine.

V. RELATED WORK

A multitude of I/O optimizations have been studied to address I/O contention, such as collective I/O and MPI-IO, these optimizations did not address the interferences generated by the concurrent usage of a shared parallel file system from multiple applications. The root cause is that there are no QoS mechanisms on HPC storage systems, and that HPC systems deploy "first-in-first out (FIFO) I/O job scheduling at the file system level with and adjustment based on priority. Recently, several scheduling policies were proposed to address inter-job I/O congestion on HPC systems. These strategies range from non-work-conserving scheduling (anticipatory scheduling [21] and the CFQ scheduler [22]) which keeps the disk head idle after serving a request until the next request arrives, to resource-aware job scheduling [23] and I/O-aware job

scheduling [24], [25] which obtain per-application resource or I/O requirements through tracing/profiling.

One recent effort of note is the in-memory computing. In-memory computing is a novel paradigm to process data as much as possible in RAM , instead of in storage devices, to reduce the cost of data movement. Analyzing data online and reducing the amount of data migrated to storage make in-memory computing an attractive and effective approach for data-intensive applications. Overall, there are two in-memory computing strategies, in situ and in transit, which have been developed for various applications scenarios for HPC data analytics. In situ data analytics can be either embedded in the application code, or run on the same node using helper cores, e.g., Darmaris [26], Functional Partitioning [27], GePSeA [28]. However, in situ typically requires modifications to applications, and may interfere with simulations. Meanwhile, for in transit, GLEAN [29] and DataSpaces [30], are performed on auxiliary staging nodes. In transit decouples analytics from simulations, and hence the impact on simulations can be minimized through asynchronous data movement. In addition, a few frameworks involve both in situ and in transit analysis. PreData [31] uses simple stateless analysis codes in situ, and then further extract data from applications to staging nodes. Bennett et al. [32] combine in situ and in transit visualization and analysis using DataSpaces and ADIOS [33]. However, despite of the advantages of in situ and in transit techniques, a fact of required data post-analyzing and necessary data post-processing for most scientific applications must be confronted, where data-backed analysis is still significant for many researchers as the standard for data analysis. Key challenges of in-memory computing are that it cannot support exploratory data analytics where the quantities to examine are unknown a priori, and for both in situ and in transit, they incur higher resource overhead, and the cost-performance trade-off needs to be carefully considered.

VI. Conclusion

In this paper, we demonstrate that under the growing gap between CPUs and disks, I/O is often disrupting scientific application performance. However, it is still entirely possible to achieve a good reduction ratio without sacrificing the accuracy. Therefore, we present load-aware data reduction in conjunction with re-computation, a methodology inspired by the philosophy behind caching that eliminates the amounts of data migrated to storage systems according to the runtime storage system load, and selectively produces data with the desired accuracy, thus creating an illusion that I/O performance is dramatically improved.

Acknowledgment

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] Summit. Https://www.olcf.ornl.gov/summit/.
[2] Titan. Http://www.olcf.ornl.gov/titan/.
[3] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.
[4] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.
[5] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE transactions on visualization and computer graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.
[6] J.-l. Gailly and M. Adler. GNU zip. Http://www.gzip.org.
[7] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.
[8] T. Lu, Q. Liu, X. He, H. Luo, E. Suchyta, J. Choi, N. Podhorszki, S. Klasky, M. Wolf, T. Liu, and Z. Qiao, "Understanding and modeling lossy compression schemes on hpc scientific data," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS 18),*, 2018, pp. 1–10.
[9] D. Meister, J. Kaiser, A. Brinkmann, T. Cortes, M. Kuhn, and J. Kunkel, "A study on data deduplication in hpc storage systems," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–11.
[10] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *European Conference on Parallel Processing*. Springer, 2011, pp. 366–379.
[11] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE transactions on visualization and computer graphics*, vol. 20, no. 12, pp. 2674–2683, 2014.
[12] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *Parallel and Distributed Processing Symposium, 2016 IEEE International*. IEEE, 2016, pp. 730–739.
[13] Q. Liu, N. Podhorszki, J. Logan, and S. Klasky, "Runtime i/o re-routing + throttling on HPC storage," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*. San Jose, CA: USENIX, 2013. [Online]. Available: https://www.usenix.org/conference/hotstorage13/workshop-program/presentation/Liu
[14] M. Adams, P. O. Schwartz, H. Johansen, P. Colella, T. J. Ligocki, D. Martin, N. Keen, D. Graves, D. Modiano, B. Van Straalen *et al.*, "Chombo software package for amr applications-design document," Tech. Rep., 2015.
[15] The Flash Center for Computational Science at the University of Chicago. FLASH 4.4. Http://flash.uchicago.edu/site/flashcode/.
[16] L. F. Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 210, pp. 307–357, 1911.
[17] T. Lu, E. Suchyta, D. Pugmire, J. Choi, S. Klasky, Q. Liu, N. Podhorszki, M. Ainsworth, and M. Wolf, "Canopus: A paradigm shift towards elastic extreme-scale data analytics on hpc storage," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 58–69.
[18] T. Lindeberg, "Feature detection with automatic scale selection," *International journal of computer vision*, vol. 30, no. 2, pp. 79–116, 1998.
[19] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, "Implicit-explicit runge-kutta methods for time-dependent partial differential equations," *Applied Numerical Mathematics*, vol. 25, no. 2-3, pp. 151–167, 1997.
[20] L. K. Bieniasz, "Use of dynamically adaptive grid techniques for the solution of electrochemical kinetic equations: Part 3. an adaptive moving grid–adaptive time step strategy for problems with discontinuous boundary conditions at the electrodes," *Journal of Electroanalytical Chemistry*, vol. 374, no. 1-2, pp. 23–35, 1994.
[21] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous i/o," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 117–130.
[22] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. R. Ganger, "Argon: Performance insulation for shared storage servers." in *FAST*, vol. 7, 2007, pp. 5–5.
[23] G. P. Rodrigo, E. Elmroth, P.-O. Östberg, and L. Ramakrishnan, "Enabling workflow-aware scheduling on hpc systems," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2017, pp. 3–14.
[24] S. Herbein, D. H. Ahn, D. Lipari, T. R. Scogland, M. Stearman, M. Grondona, J. Garlick, B. Springmeyer, and M. Taufer, "Scalable i/o-aware job scheduling for burst buffer enabled hpc clusters," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2016, pp. 69–80.
[25] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai, "Server-side log data analytics for i/o workload characterization and coordination on large shared storage systems," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 819–829.
[26] M. Dorier, G. Antoniu, F. Cappello, M. Snir, and L. Orf, "Damaris: How to efficiently leverage multicore parallelism to achieve scalable, jitter-free i/o," in *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*. IEEE, 2012, pp. 155–163.
[27] M. Li, S. S. Vazhkudai, A. R. Butt, F. Meng, X. Ma, Y. Kim, C. Engelmann, and G. Shipman, "Functional partitioning to optimize end-to-end performance on many-core architectures," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society, 2010, pp. 1–12.
[28] A. Singh, P. Balaji, and W.-c. Feng, "Gepsea: a general-purpose software acceleration framework for lightweight task offloading," in *Parallel Processing, 2009. ICPP'09. International Conference on*. IEEE, 2009, pp. 261–268.
[29] T. Tu, C. A. Rendleman, D. W. Borhani, R. O. Dror, J. Gullingsrud, M. O. Jensen, J. L. Klepeis, P. Maragakis, P. Miller, K. A. Stafford *et al.*, "A scalable parallel framework for analyzing terascale molecular dynamics simulation trajectories," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*. IEEE, 2008, pp. 1–12.
[30] C. Docan, M. Parashar, and S. Klasky, "Dataspaces: an interaction and coordination framework for coupled simulation workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, 2012.
[31] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, Q. Liu, S. Klasky, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "Predata–preparatory data analytics on peta-scale machines," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–12.
[32] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci *et al.*, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–9.
[33] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, M. Parashar, N. Samatova, K. Schwan, A. Shoshani, M. Wolf, K. Wu, and W. Yu, "Hello adios: The challenges and lessons of developing leadership class i/o frameworks," *Concurr. Comput. : Pract. Exper.*, vol. 26, no. 7, pp. 1453–1473, May 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3125