Canopus: Enabling Extreme-Scale Data Analytics on Big HPC Storage via Progressive Refactoring

Tao Lu[1], Eric Suchyta[2], Jong Choi[2], Norbert Podhorszki[2], Scott Klasky[2], Qing Liu[1], Dave Pugmire[2], Matthew Wolf[2], and Mark Ainsworth[3,2]

[1]Department of Electrical and Computer Engineering, New Jersey Institute of Technology
[2]Computer Science and Mathematics Division, Oak Ridge National Laboratory
[3]Division of Applied Mathematics, Brown University

Abstract

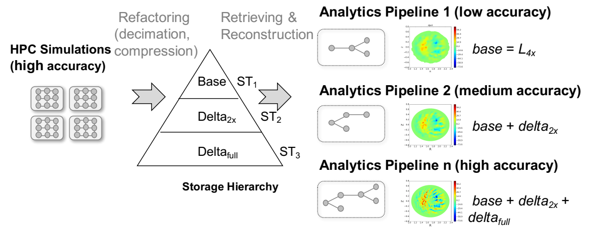
High accuracy scientific simulations on high performance computing (HPC) platforms generate large amounts of data. To allow data to be efficiently analyzed, simulation outputs need to be refactored, compressed, and properly mapped onto storage tiers. This paper presents *Canopus*, a progressive data management framework for storing and analyzing big scientific data. *Canopus* allows simulation results to be refactored into a much smaller dataset along with a series of deltas with fairly low overhead. Then, the refactored data are compressed, mapped, and written onto storage tiers. For data analytics, refactored data are selectively retrieved to restore data at a specific level of accuracy that satisfies analysis requirements. *Canopus* enables end users to make trade-offs between analysis speed and accuracy on-the-fly. *Canopus* is demonstrated and thoroughly evaluated using blob detection on fusion simulation data.

1 Introduction

In 2015, the authors worked with a team of computational scientists from Princeton Plasma Physics Laboratory to conduct a hero fusion simulation on Titan [1], utilizing 300,000 cores to model the largest fusion reactor in the world. The fidelity and resolution of the simulation demanded a solution that could efficiently store and analyze 1 PB of new data per day. On Titan, we estimated that the I/O would take at least 5 hours to finish, which was deemed too expensive. On the other hand, the total capacity of one file system partition on Titan was 14 PB. With the data rate of 1 PB per day, the file system capacity would be quickly saturated in less than 14 days. Such a large amount of data posed a multitude of data management challenges. Neither the throughput nor the capacity could be sustained by the parallel file system on Titan at the time. This forced the fusion scientists to sacrifice the fidelity of the run in order to accomplish the science campaign in an acceptable time.

What fundamentally caused this analytics difficulty was that the numerical calculations in a large scientific simulation often used the highest possible accuracy in both spatial and temporal dimensions to minimize errors after a long iteration. This nevertheless leads to a higher data volume and velocity of simulation data that the downstream analytics may not be able to digest. Working with the fusion scientists and other Titan users, we have identified the following opportunities to tackle this challenge. First, scientists do not always require analytics be performed at the the same level of accuracy as the simulation, and a lower accuracy may suffice on a case-by-case basis. Second, the storage hierarchy is becoming increasingly deep with the emergence of new storage/memory technologies. Different storage tiers possess distinct I/O characteristics and constraints, and these need to be fully exploited to meet the performance and capacity requirements from data analytics.

To facilitate data analysis in an efficient and flexible manner, we design and implement *Canopus*, a data management middleware that can refactor the simulation results (via decimation) into a base dataset along with a series of deltas. The refactored data are further compressed and mapped to storage tiers. For post-run data analytics that may examine the data with some certain accuracy and performance requirements in mind, *Canopus* allows the base level and a selected subset of deltas to be retrieved to restore the target accuracy, which may or may not be the same as the full accuracy. Figure 1 illustrates the *Canopus* workflow. For example, a simulation with a decimation ratio of 4 generates a base dataset L_{4x} that is 25% of the full accuracy size, along with $delta_{2x}$ and $delta_{full}$. In particular, $delta_{2x}$ is the difference between L_{4x} and L_{2x} (i.e., data generated with the decimation ratio of 2), and $delta_{full}$ is the difference between L_{2x} and L_{full} (i.e., the full accuracy data). These datasets can be further compressed to L^c_{4x}, $delta^c_{2x}$, $delta^c_{full}$ and mapped onto the following storage tiers, ST_1 (the fastest but the smallest in capacity, e.g., non-volatile memory (NVRAM)), ST_2 (slower



Refactoring (decimation, compression)
Retrieving & Reconstruction
HPC Simulations (high accuracy)
Base
Delta$_{2x}$
Delta$_{full}$
ST$_1$
ST$_2$
ST$_3$
Storage Hierarchy
Analytics Pipeline 1 (low accuracy)
base = L_{4x}
Analytics Pipeline 2 (medium accuracy)
base + delta$_{2x}$
Analytics Pipeline n (high accuracy)
base + delta$_{2x}$ + delta$_{full}$

Figure 1: Canopus workflow.

but larger in capacity, e.g., solid-state drive (SSD)), ST_3 (the slowest but the largest in capacity, e.g., disks) respectively. Next, data analytics has three options to examine the data: (1) it requests the lowest accuracy by quickly retrieving L^c_{4x} from ST_1, and decompressing it to obtain L_{4x}; (2) it restores a higher accuracy by additionally retrieving and decompressing $delta^c_{2x}$ from ST_2, and then performing L_{4x} + $delta_{2x} = L_{2x}$; and (3) it restores the highest accuracy by further retrieving/decompressing $delta^c_{full}$ from ST_3, and then calculating L_{4x} + $delta_{2x}$ + $delta_{full} = L_{full}$. These options allow users to progressively augment the data accuracy by fetching the deltas level by level on-the-fly.

In this paper, we describe our efforts in implementing the progressive factoring and data exploration. The functionalities are demonstrated in the blob detection use case in fusion XGC1 [2] diagnostic data, and we are particularly interested in examining the *dpot* variable, which is a 2D scalar field organized in planes in a fusion reactor. The experiments were conducted on Titan where we utilized DRAM-backed *tmpfs* and the Lustre parallel file system to build a 2-tier storage hierarchy for write and read. Overall this paper makes the following contributions:

- We design and implement *Canopus*, a data management framework that allows simulation data to be refactored so that users can perform exploratory data analysis progressively. *Canopus* utilizes the mesh decimation and floating-point compression algorithms to mitigate storage footprints and expedite data analytics on simulation results. It enables scientists to define and control data analysis according to their data collection practices and analysis needs.

- *Canopus* refactors and maps data to proper storage tiers, taking advantage of the capacity and performance characteristics of each tier. The co-design of analytics and storage provides a new data management paradigm in HPC environments.

2 Data Refactoring

In order to provide a range of accuracies to users, *Canopus* refactors simulation data into a base low accuracy dataset along with a series of deltas. The resulting data are then placed onto storage tiers, with the base dataset

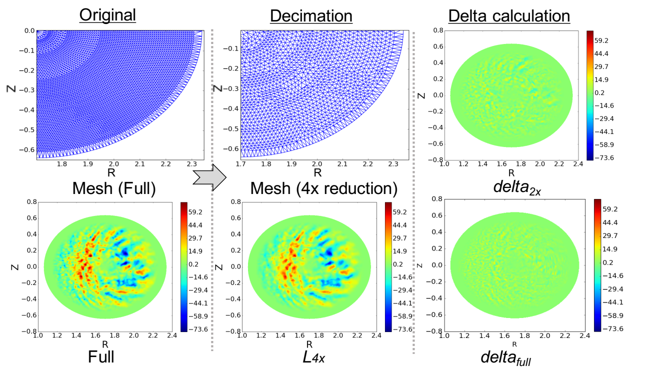


Figure 2: Data refactoring. Note x- and y-axis are location coordinates within a plane.

being placed onto a high storage tier (that can accommodate the base data), and the deltas being placed onto low storage tiers. Since the refactoring is performed during the execution of a simulation over a large number of computing cores, refactoring including compression itself needs to be relatively cheap, compared to the total simulation time. *Canopus* specifically targets the applications in which the simulation results need to be written once but will be analyzed a number of times, which is typical for scientific data analysis (e.g., for parameter sensitivity study). Therefore, the new functions and/or performance advantages in data retrieval are our major focus here.

Refactoring. In general, *Canopus* supports various approaches to refactoring data, including byte splitting, block splitting, mesh decimation. In this paper, we choose the mesh decimation to demonstrate the *Canopus* framework, because 1) the majority of the scientific simulations use mesh based data models; 2) mesh decimation enables a wide range of accuracies, as compared to other methods; and 3) it can generate a lower accuracy that is complete in geometry, and can be directly acted upon by analytics. Overall the data refactoring is done iteratively and each iteration consists of two key steps: mesh decimation and delta calculation (Figure 2).

1) Mesh decimation. *Canopus* can decimate data efficiently for both structured and unstructured meshes. It is less involved to decimate a structured mesh since its geometry (nodes and their connectivities) is rather simple. For unstructured meshes, *Canopus* adopts edge collapsing [3, 4, 5] to decimate unstructured meshes from level l to level $l+1$. To this end, we first insert all edges of a mesh into a priority queue. The priority of an edge is set as the distance between its two vertices. Data decimation is conducted by successively cutting the edge with the lowest priority from the mesh until the decimation ratio is reached. Namely the shortest edge, $E^l_{i,j}$ between vertex V^l_i and V^l_j at level l, will be removed from the mesh. Next, a new vertex, $V^{l+1}_i = NewVertex(V^l_i, V^l_j)$,

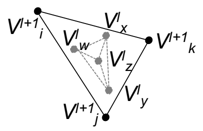
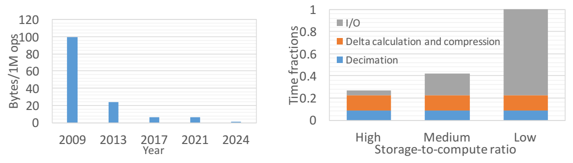


Figure 3: Demonstration of delta calculation.



(a) Storage-to-compute trend for large HPC systems. (b) The trend of storage I/O overhead.

Figure 4: Storage-to-compute trend and its impact on data refactoring.

will be constructed, and new edges between the adjacent vertices of V_i^l, V_j^l, and V_i^{l+1} are constructed and inserted into the priority queue. In general V_i^{l+1} is a linear combination (represented by $NewVertex(\cdot)$) of V_i^l and V_j^l. For simplicity, $V_i^{l+1} = (V_i^l + V_j^l)/2$. The time complexity of decimation is dominated by the cost of the insert operation in a priority queue, which is typically $O(logN)$. Note that the size of the decimated data (without compression) equals the size of the original data divided by the decimation ratio (greater than 1), and the mesh decimation can be done in embarrassingly parallel, since it can be done locally without requiring communications with others.

2) Delta calculation. After decimation, instead of placing the original data on storage, *Canopus* generates the delta, which is the difference between the pre- and post-decimation data. The reason is that we notice in the fusion datasets, the delta is much smoother than the original data (Figure 2), and can be fairly well compressed by ZFP [6], which exploits the local smoothness to achieve a high compression ratio. For data analytics that needs to perform at the original accuracy, the original data can be restored from the post-decimation data and the deltas. In the storage hierarchy, the post-decimation data is typically stored in a high tier, and the delta is stored in a low tier (due to its larger size). Figure 3 illustrates the delta calculation in a triangular mesh. $delta_n^l = F(V_n^l) - \alpha \cdot F(V_i^{l+1}) - \beta \cdot F(V_j^{l+1}) - \gamma \cdot F(V_k^{l+1})$, where $\alpha + \beta + \gamma = 1, n \in \{w, x, y, z\}$. And $F(\cdot)$ is a function that maps V_i^l to its field value (e.g., *dpot*). For simplicity, we set $\alpha = \beta = \gamma = 1/3$.

Figure 4 evaluates the refactoring cost under various system setups, and in particular Figure 4(a) shows the storage-to-compute trend since 2009 for leadership class HPC systems in the U.S.. Overall, the trend is that compute is becoming cheaper and the gap between compute and storage has widened sharply since 2013 [7]. Figure 4(b) shows the time breakdown of writing data out using *Canopus*, including the time spent on decimation, the time spent on calculating the delta between two adjacent levels along with compression using ZFP, and the time spent on I/O, under various storage-to-compute capabilities, namely, high (i.e., compute-bound), medium, and low (i.e., I/O-bound). For the compute-bound, medium, and I/O-bound scenario, we assign 32, 128, 512 cores per storage target, respectively. In particular, the medium case is chosen to reflect the capabilities of Titian which has 300,000 core with 2,016 (for two partitions) storage targets. As the compute becomes increasingly powerful as compared to the I/O, decimation and delta calculation becomes cheaper, compared to the cost of I/O.

Compression. The refactored data can be further compressed to reduce the storage cost, in terms of both capacity and data movement cost. As of now, *Canopus* has integrated ZFP [6], a state-of-the-art floating-point data compressor, to perform compression. It exploits the local smoothness that typically exists between neighboring data elements, and aggressively reduces data using both lossy and lossless compressions. Other compression libraries such as SZ [8] and FPC [9] can also be integrated into *Canopus* and these are still work in progress. As Figure 2 shows, the delta (e.g., $delta_{2x}$) calculated between adjacent levels exhibits a much higher smoothness than directly compressing the intermediate decimation results (e.g., L_{4x}). Essentially *Canopus* serves as a pre-conditioner that further prepares the data for compression and improves ZFP performance.

Data Placement. The compressed base data and deltas are then placed onto storage tiers. The base data can be decimated to the point that it is small enough to fit into a high tier, e.g., NVRAM, and the additional deltas can be placed onto lower tiers that have larger capacities. Writing data out efficiently is done by utilizing ADIOS [10], which provides a layer of abstraction that hides the storage complexities away from applications. An I/O transport that best utilizes a specific storage tier is selected and configured in an external XML configuration. As aforementioned, we built up a 2-tier storage comprised of DRAM-backed *tmpfs* and the Lustre parallel file system on Titan, and the refactored data is written in parallel from each core, with the high tier being written first followed by the low tier. The I/O time measured here is the total time spent on writing both tiers. This paper assumes once data is placed, there is no further data migration/eviction. However, in reality since the capacity of each tier is shared by users, data migration/eviction is needed to ensure the capacity is best utilized, and this is left for future study.

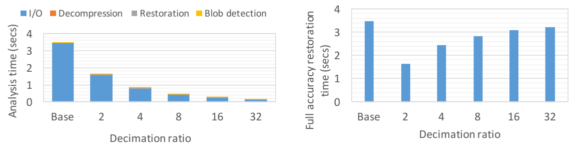
3 Progressive Data Exploration

Canopus supports progressive data explorations on floating-point data for both structured and unstructured meshes. Data analysis starts from the lowest accuracy with the base dataset, which is previously placed on a high storage tier. If the accuracy suffices and the outcome of data analytics is satisfactory, the data exploration ends here. Otherwise, data with the next level of accuracy is constructed by putting together the current accuracy and the associated delta(s) on lower tiers (Figure 1). This step is repeated until the data accuracy satisfies the users. *Canopus* takes advantage of storage characteristics, and maps refactored data to proper storage tiers. The base dataset is small in size and is stored on a high storage tier to enable fast data exploration (e.g., data inspection). Meanwhile, the large capacity of a low tier is utilized to accommodate high accuracy data. Another key benefit of *Canopus* is that the initial analysis on the low accuracy data can provide guidance to the subsequent data explorations, and facilitate retrieving a subset (as opposed to the full set) of the high accuracy data, thus reducing the amount of data retrieved from slow tiers.

I/O and Decompression. *Canopus* uses the ADIOS read API to retrieve the refactored data. ADIOS provides a metadata-rich binary-packed data format, and the complexity of retrieving data across storage tiers is considerably reduced. The global metadata in ADIOS maintains the location of the refactored data, and the users can get the attributes of the refactored data via the ADIOS query interface, e.g., using *adios_inq_var(file_handle, "dpot")*. The retrieved data is then decompressed using ZFP.

Restoration. *Canopus* subsequently restores data to the desired level of accuracy by applying a set of deltas to the base data. Namely, $F(V_n^l) = \alpha \cdot F(V_i^{l+1}) + \beta \cdot F(V_j^{l+1}) + \gamma \cdot F(V_k^{l+1}) + delta_n^l$, where $\alpha + \beta + \gamma = 1, n \in \{w, x, y, z\}$. The complexity of restoration is $O(n)$, where n is the number of vertices at level l. One critical step in restoration is to identify the set of vertices that fall into a triangle $< V_i^{l+1}, V_j^{l+1}, V_k^{l+1} >$ at level $l+1$. The brute force approach that checks whether V_n^l falls into this triangle can be expensive due to the potential large number of vertices at level l and triangles at level $l+1$ in the mesh. To this end, *Canopus* stores the mapping between V_n^l and the triangle during the refactoring phase, and the mapping information can be subsequently used to accelerate data restoration.

Figure 5(a) measures the end-to-end time of blob detection pipeline, which consists of I/O, decompression, restoration, and blob detection phases. Without any surprise, I/O is the major bottleneck in this analytics pipeline, with the rest incurring very low overhead. The baseline case in Figure 5 is the time spent on analyzing the data with the highest accuracy on the Lustre paralell file system. Decompression and restoration are not needed in this case. For other test cases, each of them measures the amount of time spent on obtaining the next level of accuracy, and performing blob detection on the



(a) End-to-end time of the analytics pipeline. (b) Restoring full accuracy data from the base dataset and deltas.

Figure 5: The performance gains of Canopus for data analytics.

restored data. For example, at the decimation ratio of 4, the total time spent, approximately 0.82 seconds, is the time to retrieve and decompress L_{4x} (25% of the original data size) and $delta_{2x}$, restore L_{2x} (50% of the original data size) , and perform blob detection on L_{2x}. Figure 5(a) demonstrates that if data of a lower accuracy can satisfy the analysis requirement, the analysis time can be significant reduced. Figure 5(b) evaluates the worst case scenario in *Canopus* where data analysis requires the full accuracy data. Still, Canopus can restore the full accuracy data and reduce the data analysis time by up to 50% due to the I/O savings from fully utilizing the storage hierarchy and data pre-conditioning for ZFP.

4 Impact on Data Analytics

Canopus allows users to select the level of accuracy that is sufficient for one's scientific discovery. The choice of accuracy is mostly driven by two questions: How much faster data analytics can run with the reduced accuracy, and what is the impact of reduced accuracy on data analytics? This section focuses on discussing the second question. The data analytics we demonstrate is blob detection in fusion XGC1 diagnostic data, from which fusion scientists examine the electrostatic potential (*dpot*) in a fusion device and study the trajectory of high energy particles. This is critical to the understanding of instabilities that can build up in fusion reactors. Herein, we use the blob detection function in OpenCV, an open source computer vision library, to identify areas with high field potentials in a 2D plane of *dpot* data, and we test the following parameters of blob detection: Config1: $< minThreshold = 10, maxThreshold = 200, minArea = 100 >$; Config2: $< minThreshold = 150, maxThreshold = 200, minArea = 100 >$; Config3: $< minThreshold = 10, maxThreshold = 200, minArea = 200 >$

Figure 6 illustrates the blob detection results under various accuracy levels: the full accuracy, L_{2x}, and L_{4x}. The circled areas are identified as high energy blobs. As the data accuracy decreases, the number of blobs captured decreases. However, we notice that the blobs tend to expand first and then disappear once the potential falls below a certain threshold. This is caused by the averaging effect of the edge collapsing technique that *Cano-*

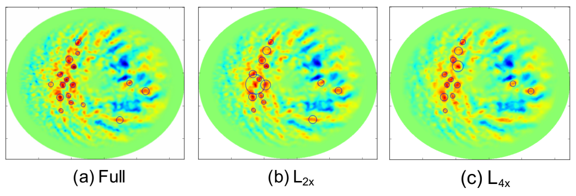
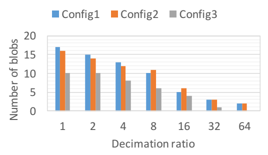
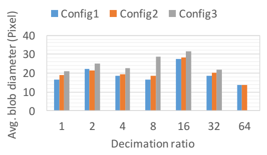


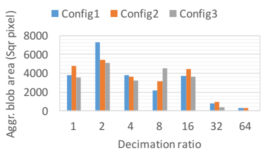
Figure 6: A macroscopic view of Blob detection.



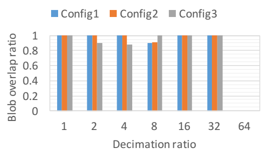
(a) Number of blobs detected.



(b) Avg. blob diameters.



(c) Blob area.



(d) Blob overlapping ratio.

Figure 7: A quantitative evaluation of blob detection.

pus employs to decimate unstructured mesh. This is evidenced by the increased blob diameter and area size in Figure 7(b) and (c), and as such, a cluster of blobs may overlap and merge. Nevertheless, in Figure 6 most blobs in the full accuracy data can still be detected in a moderately reduced accuracy, and blobs detected in a low accuracy data have a high overlap ratio with the blobs detected in the full accuracy data (Figure 7(d)). We define two blobs are overlapped if the distance between two blob centers is less than the sum of the two individual radiuses. The high overlap ratio suggests that the blobs detected in the low accuracy data can still capture the areas with high field potential. If a lower accuracy can not satisfy the scientific requirements, a higher accuracy data can always be further restored and processed.

5 Related Work

Data management is one of the top ten research exascale challenges [11]. Data reduction methods preserving the capability for exploratory analysis are desired. Similar efforts have been made to enable efficient queries directly on compressed database storage [12]. A challenge for HPC storage is to develop a flexible data reduction mechanism that supports scientist-defined analysis.

Due to the relatively low compression ratios, lossless compression algorithms have limited impact on reducing data footprints. Laney et. al. [13] demonstrated that lossy compression mechanisms could achieve a 3 to 5X reduction ratio without causing significant changes to important physical quantities. *ZFP* [6] and *SZ* [8] have been implemented to achieve high reduction ratio on floating-point data. Mesh compression algorithms reduce both geometry data and connectivity data [14]. To enable multi-accuracy analysis, progressive mesh compression has been proposed [15]. In other directions, Migratory Compression [16] recognizes data chunk similarities, and relocates similar chunks together to improve data compressibility. Simulation-time data analysis [17, 18], which tries to perform as much in-memory analysis as possible, has roused research interests.

System level optimizations can also expedite data analysis. SDS [19] reorganizes data to match the read patterns so as to accelerate analytical tasks. BurstFS [20] utilizes high-performance burst buffers to support scalable and efficient aggregation of I/O bandwidth, accelerating read operations of analytical tasks. ADIOS middleware [10] enables us to design and implement *Canopus*, a data refactoring framework which employs mesh compression and floating-point compression algorithms to achieve a high data reduction ratio without sacrificing the analyzability of the data. The approach of *Canopus* also shares some similarities with the concept of lineage by trading compute for I/O [21, 22, 23]. The co-design of data compression, storage, and analysis over mesh-based data differentiates *Canopus* from existing work.

6 Conclusion

This paper describes our efforts in enabling extreme-scale data analytics via progressive factoring. This paper is motivated by a large-scale production run, in which the data was too large to be effectively analyzed. To address this challenge, we design and implement *Canopus*, a data management middle that allows simulation data to be refactored, compressed, and mapped to storage tiers. The key advantage of *Canopus* is that users can perform exploratory data analysis progressively without being forced to work on the highest accuracy data. To achieve this, *Canopus* utilizes mesh decimation to generate a base dataset along with a series of deltas. Its co-design of analysis and storage provides a new data management paradigm for simulation-based science.

7 Acknowledgement

This research was supported by DOE SIRIUS project, the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and Oak Ridge Leadership Computing Facility.

References

[1] "Titan at Oak Ridge Leadership Computing Facility." [Online]. Available: https://www.olcf.ornl.gov/titan/

[2] "XGC1 code." [Online]. Available: http://epsi.pppl.gov/computing/xgc-1

[3] M. Isenburg and J. Snoeyink, “Mesh collapse compression,” in *SCG ’99*, Miami Beach, Florida, USA, 1999.

[4] M. Garland and P. S. Heckbert, “Surface simplification using quadric error metrics,” in *SIGGRAPH ’97*, Los Angeles, CA, 1997.

[5] H. Hoppe, “New quadric metric for simplifying meshes with appearance attributes,” in *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS ’99)*, San Francisco, California, 1999.

[6] P. Lindstrom, “Fixed-rate compressed floating-point arrays,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2674–2683, Dec 2014.

[7] S. Klasky, “CODAR: Center for Online Data Analysis and Reduction,” invited speech in HPCChina’16, Xi’an, China. [Online]. Available: http://www.ncic.ac.cn/codesign/codesign_ppt/CODAR_overview_Oct_27_2016_Klasky.pdf

[8] S. Di and F. Cappello, “Fast Error-Bounded Lossy HPC Data Compression with SZ,” in *IPDPS’16*, 2016.

[9] M. Burtscher and P. Ratanaworabhan, “Fpc: A high-speed compressor for double-precision floating-point data,” *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, Jan 2009.

[10] 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.

[11] ASCAC Subcommittee, “Top ten exascale research challenges,” 2014. [Online]. Available: https://science.energy.gov/ /media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf

[12] R. Agarwal, A. Khandelwal, and I. Stoica, “Succinct: Enabling queries on compressed data,” in *NSDI’15*, Oakland, CA, 2015.

[13] D. Laney, S. Langer, C. Weber, P. Lindstrom, and A. Wegener, “Assessing the effects of data compression in simulations using physically motivated metrics,” *Scientific Programming*, vol. 22, no. 2, pp. 141–155, 2014.

[14] M. Deering, “Geometry compression,” in *SIGGRAPH ’95*, New York, NY, USA, 1995.

[15] D. Cohen-Or, D. Levin, and O. Remez, “Progressive compression of arbitrary triangular meshes,” in *VIS ’99*, San Francisco, California, USA, 1999.

[16] X. Lin, G. Lu, F. Douglis, P. Shilane, and G. Wallace, “Migratory compression: Coarse-grained data reordering to improve compressibility,” in *FAST’14*, Santa Clara, CA, 2014.

[17] V. Vishwanath, M. Hereld, and M. E. Papka, “Toward simulation-time data analysis and I/O acceleration on leadership-class systems,” in *LDAV’11*, 2011.

[18] U. Ayachit, A. Bauer, B. Geveci, P. O’Leary, K. Moreland, N. Fabian, and J. Mauldin, “ParaView Catalyst: Enabling In Situ Data Analysis and Visualization,” in *ISAV’15*, 2015.

[19] B. Dong, S. Byna, and K. Wu, “Expediting scientific data analysis with reorganization of data,” in *Cluster’13*, 2013.

[20] T. Wang, K. Mohror, A. Moody, K. Sato, and W. Yu, “An ephemeral burst-buffer file system for scientific applications,” in *SC’16*, 2016.

[21] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, “Tachyon: Reliable, memory speed storage for cluster computing frameworks,” in *SOCC ’14*, Seattle, WA, USA, 2014.

[22] J. Bent, D. Thain, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny, “Explicit control a batch-aware distributed file system,” in *NSDI’04*, San Francisco, California, 2004.

[23] P. K. Gunda, L. Ravindranath, C. A. Thekkath, Y. Yu, and L. Zhuang, “Nectar: Automatic management of data and computation in datacenters,” in *OSDI’10*, Vancouver, BC, Canada, 2010.