# Segmented Analysis for Reducing Data Movement

Jialin Liu
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
Email: jaln.liu@ttu.edu

Surendra Byna
Computational Research Division
Lawrence Berkeley Laboratory
Berkeley, California, USA
Email: SByna@lbl.gov

Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, Texas, USA
Email: yong.chen@ttu.edu

***Abstract*—Many scientific applications nowadays generate a few terabytes (TB) of data in a single run and the data sizes are expected to reach petabytes (PB) in the near future. Enabling fast extraction of knowledge through analyzing these large datasets holds the key to faster scientific discoveries. However, reading data from traditional storage subsystem is a slow process as the I/O performance lags far behind computational performance. Reducing data movement from the storage subsystem is widely considered a viable option for improving performance of data analysis. In this paper, we propose *Segmented Analysis*, a data movement reduction strategy through reusing results, where multiple similar analysis tasks process the same segments of data. The basic idea is to segment the data accessed in an analysis task, to process the data segments with a given analysis task, and to store the results of segments in a cache for future use. In future, when an analysis task needs to perform the same process on the data segments for which the results are available in the cache, the task can avoid moving data and performing computation for the available results. The Segmented Analysis framework contains modules for computation and I/O access overlap detection, *in situ* segmentation, and segment result caching. We evaluate the Segmented Analysis strategy by varying factors like the overlap rate among analysis tasks, the request size and the granularity of segmentation. We observed 2X to 13X I/O and to 2X to 8X computation speedups when the overlap is above 50%.**

***Keywords*—Big data, Scientific dataset management, Segmented analysis**

## I. Introduction

The volume of data collected from instruments and simulations for scientific discovery and innovations keeps increasing rapidly. For example, the Global Cloud Resolving Model (GCRM) project [3], part of DOE's Scientific Discovery through Advanced Computing (SciDAC) program, is built on a geodesic grid that consists of more than 100 million hexagonal columns with 128 levels per column. These 128 levels cover a layer of 50 kilometers of atmosphere up from the surface of the earth. For each of these grid cells, scientists need to store, analyze, and predict parameters such as the wind speed, temperature, pressure, etc. Most of these global atmospheric models process data at a 100-kilometer scale (the distance on the ground); however, scientists desire higher resolution and finer granularity, which can lead to significantly larger datasets.

With the increasing data volume in this Big Data era, poor I/O performance becomes a critical bottleneck. The main cause of I/O bottlenecks is slow disk-read speeds compared to both the CPU speed and memory bandwidths. As the volume of data collected from instruments and scientific simulations keep increasing rapidly, the I/O cost can dominate an application's execution time. Recent studies show that reducing data movement is a straightforward, yet powerful method to address the I/O problem. Some of the techniques reduce data movement include *in situ* processing of data concurrently with data production by simulations and experiments [6, 12, 21], and compression of data before writing data to storage [9, 11, 13]. Reducing data is also considered a strategy for reducing energy consumption of large scale systems [7, 19].

Although there are efforts to reduce data movement during output phase of I/O, many analysis applications cannot take advantage of that strategy. Simulations and experiments without a *priori* knowledge of all possible analysis tasks cannot exploit *in situ* processing and need to store most of the data for exploratory analysis. This is true in many climate simulations, where scientists perform their analysis based on multiple iterations of diverse analysis tasks. They typically prefer to store most of the data from simulation output. Analyzing data from these applications typically reads the data from storage into memory to process. Active storage [15, 17, 20] attempts reducing data movement during analysis phase by moving computation near storage devices, which assumes storage devices are powerful to perform the processing required. However, many analysis tasks require more computational power than storage devices offer, especially when working on Big Data. Accessing compressed data reduces data movement, yet requires an extra decompression step before processing. Reducing data movement during analysis phase requires more attention.

In this paper, we propose a new method for reducing the data movement, called *Segmented Analysis*. The idea behind our strategy is to detect the computation and I/O overlap among multiple analysis tasks and maintain the intermediate results of the tasks for future reuse and reduce repeated I/O and computation. Compared to traditional *caching*, which stores frequently or recently used "data", we cache frequently or recently computed analysis result segments for future reuse. In our strategy, when a future analysis task needs to access the same segments of data for which we have cached results for, our proposed framework detects the overlapping segments and accesses only the data for which analysis results are not available. The framework then processes the results of existing segments and new segments. There is a possibility of partial overlap of existing segments, where the new analysis segments can only use part of results from individual segments. We design an efficient fine-grained segmentation strategy to tackle the issue.

The rest of this paper is organized as follows. Section II

presents a motivating example for overlapping analysis tasks and performance benefits. Section III presents the design of the segmented analysis framework. We present our system setup and performance evaluation results in Section IV. Section V discusses related work and compares them with our proposed strategy. Section VI concludes the discussion and briefly discusses future work.

## II. Motivation

Big Data computing is clearly critical, as the data volume of many applications keeps increasing. Previously, numerous research efforts have focused on optimizing the I/O cost. Among these studies, reducing data movement has been generally agreed as an effective approach. In the real-time systems (e.g., Hive and Pig), users can issue different tasks constantly. As a result, the possibility of overlaps among consecutive tasks, either computation overlap, or I/O overlap, or both, is increased. In climate sciences, a typical case is the CDO (Climate Data Operator) [2]. The CDO has provided more than 200 operators to manipulate the NetCDF dataset Each time the users can choose one operator to access the data and conduct computations, but as the data volume keeps increasing, the overlap of consecutive operations results in redundant data movement.

```
To compute ensemble mean:
Task1:cdo ensmean in1 in2 in3 ofile1
Task2:cdo ensmean in3 in4 in5 ofile2
Task3:cdo ensmean in1 in2 in5 ofile3
```

As shown in the above commands, there are three tasks to calculate the ensemble mean of specific subsets. Without any optimization, each task will access all data independently. However, we can find that there is I/O overlap (e.g., both Task1 and Task2 access 'in3') among different tasks. Caching each tasks' analysis results reduces the data movement.

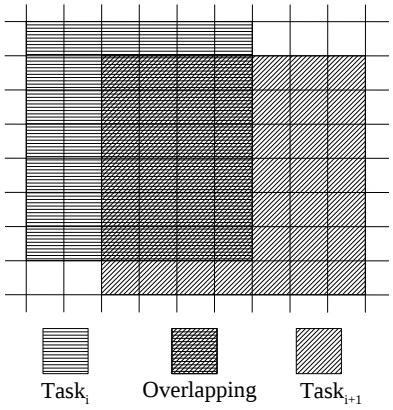


Fig. 1: Task Overlapping

In Figure 1, we describe a more general case and show how our segmented analysis can reduce the data movement. We demonstrate two tasks in this figure. The two tasks execute consecutively and perform the same type of computation. The $task_i$ computes the 'mean' of the whole data. The $task_{i+1}$ accesses and computes 'mean' for the data shown in 'orange' in Figure 1. It can be found that between the two tasks there is an overlapping part. By a fixed-size grid segmentation, the overlapping data reveals 24 segments. These segments are essentially not needed to be accessed and processed repeatedly. If there is a way to detect (the computation is supposed to be conducted on the whole data previously) and reuse the results of the overlapping part before new tasks come, the data movement could be significantly reduced.

We propose the segmented analysis idea with roughly three steps. First, during the analysis of $task_i$, partition the data into segments (e.g., 42 segments in $task_i$). Second, along with the $task_i$'s computation, perform the same computation within each segment and store the results as sub-results (the computation is supposed to be conducted on the whole data previously). Third, reuse the sub-result in the $task_{i+1}$ if possible. Compared to accessing all the 49 segments for $task_{i+1}$ in traditional way, the performance speedup is about $49/(49-25) = 1.96$X.

This performance benefit comes from trading off the computation for I/O. The fundamental idea is to apply the same computation into finer segments, such that the future analysis can be reconstructed using the sub-results when there is overlapping. It should be noted that there are two conditions, i.e., the computation has overlap and the I/O has overlap, which are necessary for a successful segmented analysis. The challenges include how to detect the overlap and how to design a flexible segmentation strategy, etc. We discuss and address them in next section.

## III. Design of Segmented Analysis

The segmented analysis framework has two major functions, i.e., overlap detection and *in situ* segmentation. The overlap detection detects the I/O and computation overlap among existing tasks and new task. The *in situ* segmentation provides a fine-grained segmentation strategy.

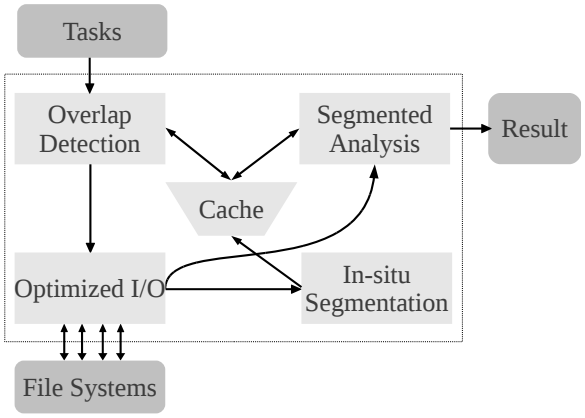


Fig. 2: Segmented Analysis Framework

Figure 2 shows an overview of the framwork. When a new tasks comes in, the computation and I/O overlap will be detected by looking up the previous analysis results from the cache. After detecting the overlap and using the previous analysis results, the I/O and computation are optimized. The data is returned to the 'Segmented Analysis' to complete the current task and at the same time, the data will also be processed by the *in situ* segmentation. After segmenting, the

sub-results of finer-grained segments are stored in the cache. We discuss the functions in detail in the following subsections.

### A. Overlap Detection

The 'task' in this paper refers to a combination of computation and I/O. For a 3D scientific dataset, the I/O in netCDF layer is represented as a start/length n-D array. The 'start' array stores the beginning positions on every dimension, and the 'length' array stores the offsets from the beginning position on every dimension. In HDF5, this I/O representation is called a 'hyperslab'. The 'task' therefore can be formalized as:

$$\begin{aligned} Task_i &= < Computation, I/O > \\ I/O &= DataRange[start, length] \end{aligned} \quad (1)$$

We store all the existing tasks in a tree, where each 'computation' node has multiple start/length pairs, as shown in Figure 3.

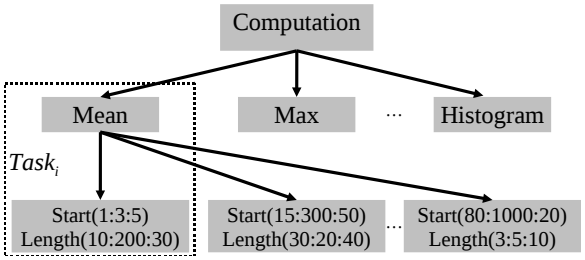


Fig. 3: Task Trees in Segmented Analysis System

To detect the overlap of the current task with previous tasks, we perform a lookup operation on the task tree. First, the computation type (e.g., mean) is searched and matched. If the same computation type is found in the task tree, then the system will start searching the data range. Matching returns 'YES' whenever there is intersection between two data ranges. After matching, there may be multiple overlapping data ranges that can be further reused. The current tasks will use those sub-results to reduce the data movement.

### B. In situ Segmentation

We record every task with its analysis result in the task tree. The new task's data movement can just focus on the non-overlapping part. For the overlapping part.

As shown in Figure 4, when the data are loaded from the storage nodes, the data will be segmented using a High-level Segmenting function, which is to perform in-memory segmenting for the data. The 'High-level Segmenting' in our system is different with the chunking algorithm in existing I/O libraries. The existing library's chunking is called Low-level Chunking in our system. For example, in HDF5, users can specify the chunking granularity, e.g., 10 on each dimension. The process can read/write by a $10 \times 10 \times 10$ chunk unit (for a 3-D dataset). This strategy has been proven to improve the I/O performance comparing to accessing byte by byte.

In our system, the High-level Segmenting does not have to align with the existing library's chunking. When the data are ready on the compute nodes, the existing chunking, which

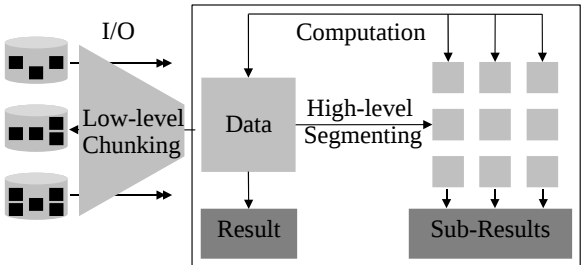


Fig. 4: *In situ* Segmentation Process

is the fixed-size chunking, could not well serve the segmented analysis. The reason is that the detected overlapping data could be of any size. If the detected overlap has dimensionality less than the fixed-size chunk, then the segmented analysis can not reuse the sub-results. For example, if the fixed-size chunking specified priorly is a 3-D cube, but the overlapping is only detected on some 2-D planes, then such mismatching will make the intermediate results hard to use.

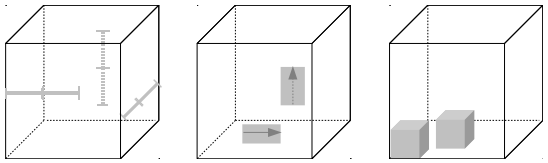

Fig. 5: Dimension-driven Segmentation for 3-D Data

We design the *dimension-driven segmentation* to provide a flexible segmentation strategy. As shown in Figure 5, the left-most subfigure shows a 1-D segmenting along differnet dimension. The reason we distinguish the x-y-z dimension is to consider the different access patterns. In real applications, the access pattern usually varies from x-y to y-z and x-z dimension (for 3D dataset). The middle subfigure in Figure 5 is an example of how to segment on 2-D planes. Comparing to 1-D segmenting, where only the length is variance, the size of the 2-D planes is determined by length and width. For 3-D segmenting, the segment grows on all the three dimensions. This segmentation strategy also applys to higher dimensional data. When the segmentation type and segment size are specified, the sub-results on each segment will be calculated and maintained.

## IV. Evaluation

### A. Dataset and Experimental Setup

We have conducted all our experiments on a 640-node cluster. The dataset we used is a synthetic 4-D NetCDF dataset, which has size of 105GB. The file system on this cluster is Lustre. We striped the data across 40 Object Storage Targets (OSTs), with the stripe size of 1 MB. The segmented analysis system is currently implemented on Parallel netCDF(v1.3.1).

### B. Performance Results and Analysis

We plot an initial evaluation result to demonstrate the performance improvement in Figures 6 and 7. In this test, we initialize our system with 100 processes on 9 compute

nodes (each node has 12 processes). The system accepts one task each time. The 'task' for this test is to calculate the sum of a subset. We have set a fixed request size, of 76 MB data, for each process. 100 tasks are issued to how the performance varies when overlapping is detected. Traditionally, each task will get its own result independently. Using the segmented analysis strategy, each task's result will be utilized and reused for future task. We labeled the traditional non-optimized method as 'SegOff' in the figure, and our system as 'SegOn'. As the overlapping percentage increases from 10% to 90%, we observe a clear I/O reduction in Figure 6. The I/O performance speedup increased from 1.2X at 10% overlap to 13.5X at 90% overlap. With 100% overlap, the speedup is 11000X. In real application, 100% overlap may not occur often, yest we can expect an average 3X speedup for all other cases. Such performance gain mainly comes from the reduction of data movement, which confirms that our system is promising in the big data processing.

Figure 7 shows performance improvement due to computation overlap. The computation cost of the non-optimized analysis (labeled as 'SegOff'), remains consistent with the request size. But as shown in Figure 7, we can see the computation cost can also be largely reduced using the segmented analysis. The average computation speedup is from 2X to 8X at overlapping rate from 10% to 90%.

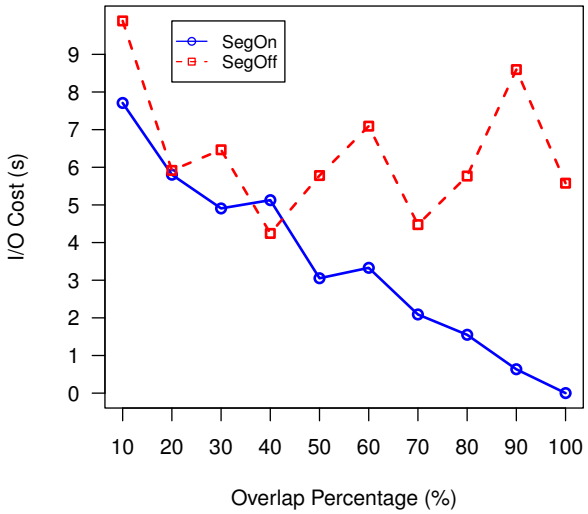


Fig. 6: I/O Cost Reduction with Segmented Analysis

In terms of total execution time (computation time and I/O time) for one task, 99% of the performance gain of the segmented analysis comes from the I/O reduction and 1% from the computation reduction, at the overlapping rate of 50%. This performance gain rate varies as the data size and computation task changes.

With a promising potential in reducing I/O cost, we are also interested in evaluating the system's overhead. As shown in Figure 2, the system has two major components, i.e., overlap detection and *in situ* segmentation. These components consume additional computing resource in the runtime.

We conducted experiments to analyze our system's overhead, and found that the majority of overhead is caused by cache file read, i.e., 72.1% of total overhead on average. The total cost of the system does not impede the performance, with peak overhead only 5% of total execution time.

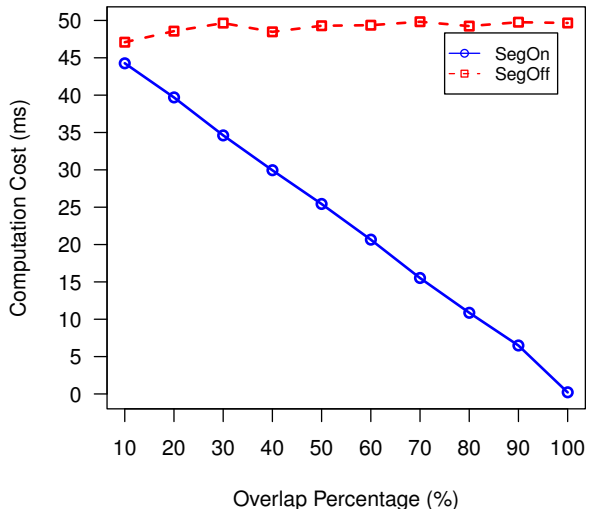


Fig. 7: Computation Cost Reduction with Segmented Analysis

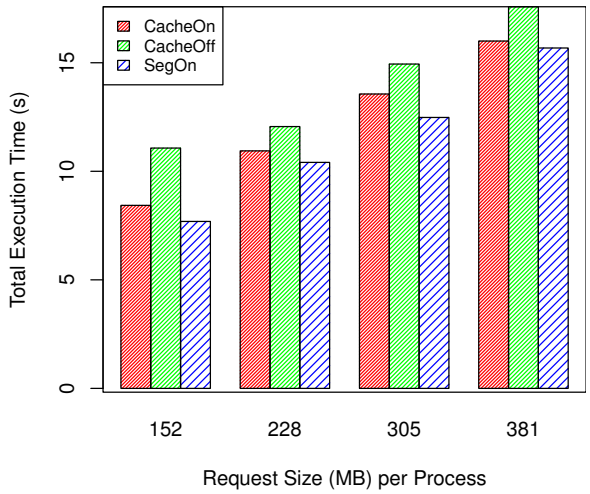


Fig. 8: Cache Data vs Cache Results

The main idea of the segmented analysis framework is to cache the analysis results. We compare its performance with the traditional data caching, in which the overlapping data are cached on the local client side. In Figure 8, we vary the request size per process and manually set a overlap rate at 10% for all three cases. The totally accessed data in each group of test is $152 \times 100 = 14GB, 22GB, 29GB$ and $37GB$ separately. The first test case is 'CacheOn', in which the local cache is turned on to maintain the overlapping data. The 'CacheOff' is a traditional case that with no cache for overlapping data. The third case 'SegOn' is our segmented analysis, where only analysis results are cached on the local. We can see the 'SegOn' outperforms both of the other two cases. Besides, 'CacheOn' achieves a comparable performance with our framework. But, it should be noted that as the request size increases, the size of cached data on the client side also increases. This increasing storage overhead on the client side can not be accepted. The segmented analysis shows promising

potential in the big data era.

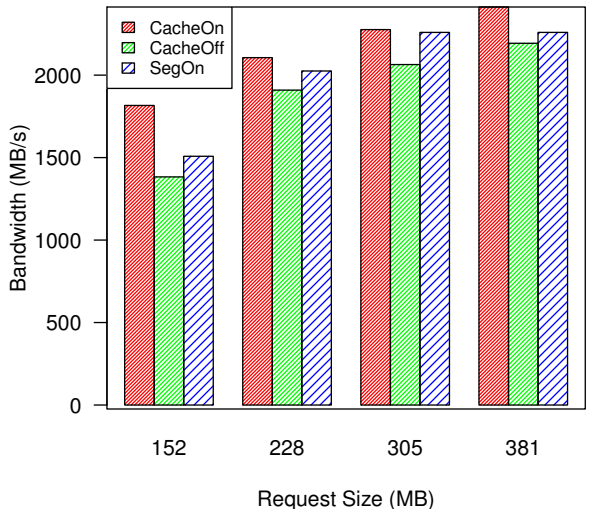


Fig. 9: Bandwidth Analysis

We also plot the aggregated bandwidth of the three cases in Figure 9. The 'CacheOn' has the best performance in terms of I/O bandwidth, with 8% more comparing to 'SegOn', and 14.7% larger than 'CacheOff'. The reason that 'CacheOn' has higher bandwidth is that part of its data are accessed from local, and another part of the data are from remote storage nodes. The local data takes less time than remote storage nodes via network. 'CacheOff' only accesses data from remote storage nodes, which takes more time and results in poor I/O bandwidth. The 'SegOn' accesses data from storage nodes, but only for the non-overlap data, which has smaller size that the initial request size. This test shows that our segmented analysis system can achieve a better Bandwidth than the no-cache analysis, and a closer performance with the all-cache analysis. Since all-cache (cache all the overlapping data) is not practical as the data volume keeps increasing, we are more confident in the potential of this segmented analysis.

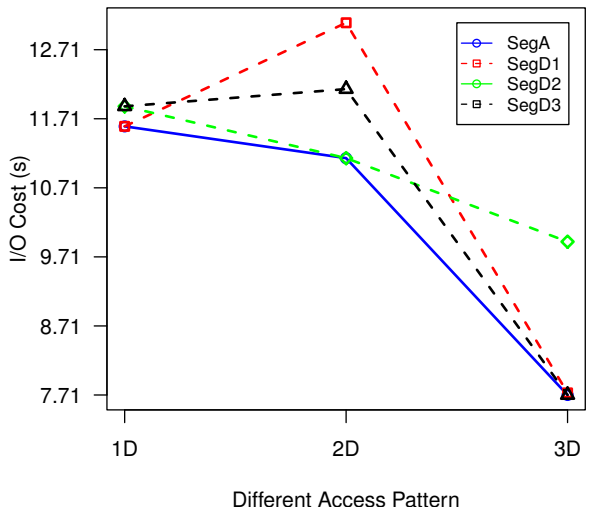


Fig. 10: Fine-grained Segmentation Evaluation

In the segmented analysis framework, we have designed a fine-grained segmenting function. Here we change the access pattern and observe the performance with different segmentation granularity. In Figure 10, 'SegA' is the default configuration in our system (i.e., SegA), which turns on all the segmentation strategy. 'SegD1', 'SegD2' and 'SegD3' are three different segmentations that have granularity of 1D, 2D, and 3D separately. The access pattern are ranges from 1D access, 2D plane access and, 3D subcube access. We can see from Figure 10, without the default segmentation strategy, none of the segmentation can achieve the best performance all the time. For example, 'SegD2' performs badly when the access pattern is on 3D subcube. 'SegD1' is supposed to achieve a good performance when the access pattern is 1D or 2D. However, 'SegD1' does not performs good in 2D case. The reason is that there is additional cost for constructing a 2D plane from the 1D results. This test shows us that a flexible and finer segmentation strategy is necessary to achieve good performance. We will explore further to develop an intelligent segmentation strategy in the future, in which the most frequent access pattern can be better served using flexible segmentation.

## V. Related Work

Reusing analysis results is novel in scientific data analysis. In MapReduce, a recent idea, 'results reusing' has been proved to be a powerful technique that can improve the data processing performance [10, 22]. The fundamental idea is to keep the intermediate results from MapReduce jobs and reuse them for future workflows. Using MapReduce to process large datasets is a totally different story comparing to parallel scientific I/O libraries like PnetCDF [5], ADIOS [1] and HDF5 [4]. Our system focuses on scientific datasets. We figured out reusing the analysis results by detecting the computation and I/O overlapping, and we also designed a fine-grained segmentation strategy utilizing various access pattern and the specific formats of scientific datasets.

By chunking the data, instead of reading/writing byte by byte, an application can get much better I/O performance accessing chunk by chunk [8, 16]. In most scientific I/O libraries, the chunking optimization has been deployed. The term segment in our system is similar to the chunk concept. A major difference is that the segmentation is performed on a higher level, above the I/O layer and the file system layer. Regarding the data organization, there exist studies applying space filling curves in the scientific dataset. The Elastic Data Organization (EDO) was proposed to address the issue of accessing slow dimensions of datasets via providing various data organization schemes [18]. Research efforts have also started investigations on designing an even better file system that considers scientific dataset organization and applications' access characteristics [14]. Our work has a same goal with these works, i.e., improving I/O performance, but focuses on reducing data movement by utilizing existing results. Besides, we designed the fine-grained segments with consideration of the access pattern and data organization. Our effort provides a framework for scientific data analysis for reducing data movement significantly.

## VI. Conclusion and Future Work

Big Data computing brings exciting new opportunities for many scientific discoveries and innovations that can benefit

considerably from both instrument data and simulation data. Big Data computing also challenges the research community to enable fast extraction of knowledge through analyzing these large datasets.

In this research, we present an innovative segmented analysis approach for efficient data movement reduction to enable fast analysis of large datasets. The segmented analysis system consists of computation and I/O access overlap detection, *in situ* segmentation, and segment result caching functions. The fundamental idea is to segment the data accessed in an analysis task, store the analysis results of segments in a cache, and reuse the prior segmented analysis results as much as possible. With such segmented analysis and reuses enabled, the analysis task can avoid moving data and performing computation for the results that are available, which significantly reduces data movement and thus enables fast analysis. We have carried out extensive experimental evaluations and the results have confirmed that the segmented analysis approach reduces data movement and leads to up to 13X I/O performance improvement.

The newly proposed segmented analysis is a generic methodology and can be used in many big data analysis and applications. This study has shown a significant potential of the segmented analysis approach, and we intend to further study the segmented analysis approach for its broad applicability. We intend to study a prefetching-like advanced analysis enabled by segmented analysis as well.

## ACKNOWLEDGMENT

The authors are thankful to Jay Lofstead of Sandia National Laboratory for his constructive and thoughtful suggestions toward this study. This research is sponsored in part by the National Science Foundation under grant CNS-1162488 and the Texas Tech University startup grant. This work is also supported in part by the Director, Office of Laboratory Policy and Infrastructure Management of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

## REFERENCES

[1] ADIOS. http://www.olcf.ornl.gov/center-projects/adios/.

[2] Climate data operator. https://code.zmaw.de/projects/cdo.

[3] The global cloud resolving model (gcrm) project. http://kiwi.atmos.colostate.edu/gcrm/.

[4] HDF5. http://www.hdfgroup.org/HDF5/doc/index.html.

[5] Parallel NetCDF. www.mcs.anl.gov/parallel-netcdf.

[6] 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 *HPDC*, pages 27–36, 2011.

[7] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012.

[8] L. T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, and A. Shoshani. Efficient organization and access of multi-dimensional datasets on tertiary storage systems, May 09 1995.

[9] K. M. Curewitz, P. Krishnan, and J. S. Vitter. Practical prefetching via data compression. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 257–266, Washington, D.C., 26–28 May 1993.

[10] I. Elghandour and A. Aboulnaga. Restore: Reusing results of mapreduce jobs. In *Proceedings of the VLDB Endowment (PVLDB), Vol. 5, No. 6, pp. 586-597 (2012)*, volume 5, Feb. 29 2012.

[11] M. Gardner, W. chun Feng, J. Archuleta, H. Lin, and X. Ma. Parallel genomic sequence-searching on an ad-hoc grid: Experiences, lessons learned, and implications. In *SC'2006 Conference*, Tampa, FL, Nov. 2006.

[12] J. Gray, D. T. Liu, M. A. Nieto-Santisteban, A. S. Szalay, G. Heber, and D. DeWitt. Scientific data management in the coming decade. Technical Report MSR-TR-2005-10, Microsoft Research (MSR), Jan. 2005.

[13] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C.-S. Chang, S. Klasky, R. Latham, R. B. Ross, and N. F. Samatova. ISABELA-QA: query-driven analytics with ISABELA-compressed extreme-scale scientific data. In *SC 2011, Seattle, WA, USA, November 12-18, 2011*, page 31, 2011.

[14] J. F. Lofstead, M. Polte, G. A. Gibson, S. Klasky, K. Schwan, R. Oldfield, M. Wolf, and Q. Liu. Six degrees of scientific data: reading patterns for extreme scale science IO. In *Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing, San Jose, CA, USA*, pages 49–60. ACM, 2011.

[15] X. Ma and A. L. N. Reddy. MVSS: An active storage architecture. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, PDS-14(10):993–1005, Oct. 2003.

[16] E. J. Otoo, G. Nimako, and D. Ohene-Kwofie. Using chunked extendible array for physical storage of scientific datasets. In *SC Companion*, pages 1315–1321, 2012.

[17] J. Piernas, J. Nieplocha, and E. J. Felix. Evaluation of active storage strategies for the lustre parallel file system. In *SC'07*. ACM/IEEE, Reno, NV, Nov. 2007.

[18] Y. Tian, S. Klasky, H. Abbasi, J. F. Lofstead, R. W. Grout, N. Podhorszki, Q. Liu, Y. Wang, and W. Yu. EDO: Improving read performance for scientific applications through elastic data organization. In *CLUSTER*, pages 93–102. IEEE, 2011.

[19] L. Wang and S. U. Khan. Review of performance metrics for green data centers: a taxonomy study. *The Journal of Supercomputing*, 63(3):639–656, Mar. 2013.

[20] R. Wickremesinghe, J. S. Chase, and J. S. Vitter. Distributed computing with load-managed active storage. In *Proc. 11th IEEE International Symposium on High Performance Distributed Computing (11th HPDC'02)*, pages 13–23, July 2002.

[21] H. Yu, C. Wang, R. W. Grout, J. H. Chen, and K.-L. Ma. In situ visualization for large-scale combustion simulations. *IEEE Computer Graphics and Applications*, 30(3):45–57, May/June 2010.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Technical Report UCB/EECS-2011-82, EECS Department, University of California, Berkeley, Jul 2011.