A Compression Framework for Multidimensional Scientific Datasets

Tekin Bicer
Ohio State University
5th Year PhD Student
bicer@cse.ohio-state.edu

Gagan Agrawal
Ohio State University
Student's Advisor
agrawal@cse.ohio-state.edu

I. Introduction

Scientific simulations and instruments can generate tremendous amount of data in short time periods. Since the generated data is used for inferring new knowledge, it is important to efficiently store and provide it to the scientific endeavors. Although parallel and distributed systems can help to ease the management of such data, the transmission and storage are still challenging problems.

For instance, scientific experiments and instruments are collecting data at increasing granularity. The Advanced LIGO Project, funded with a $200 million investment from National Science Foundation, is increasing the sensitivity of LIGO (Laser Interferometer Gravitational-wave Observatories) by a factor of 10, resulting in a three orders of magnitude increase in the number of candidates for gravitational wave signals[1]. The Global Cloud-Resolving Model (GCRM) is a simulation model which can imitate the circulations that are related with clouds. Currently, it uses a grid-cell size of 4 km. GCRM can generate 1 petabyte of data for a 10 day simulation. It is expected to simulate GCRM with higher density in the future which eventually will result in more data generation. For example, 1 km grid-cell size will expand the size of generated data by 64 folds.

Although computational resources can cope up with these rapidly growing datasets, it is difficult to transfer and store them. One popular approach for reducing data transfer overheads is compression. Compression has recently been applied for reading large files in parallel file systems. However, effectively supporting compression for scientific simulation data and integrating compression with data-intensive applications remains a challenge.

There are several libraries which are built to manage scientific datasets, including netCDF and HDF5. These libraries can help scientists to efficiently create and access the scientific datasets. Furthermore, they can easily be integrated into data-intensive applications. These software libraries provide compression capabilities at some level. However, these compression algorithms typically are generic and cannot perform well on highly-entropic scientific datasets.

In our previous work [5], we proposed a compression framework and methodology for scientific datasets. We ported it into a data-intensive computing framework [4] and showed its performance. In this work, we focus on management of multidimensional scientific datasets. We adapt our framework to PNetCDF [1] which is a set of parallel software libraries for scientific datasets. Our framework and methodology exploit the features of PNetCDF, and improve the performance of the overall system.

[1] http://media.caltech.edu/press_releases/13123

II. Related Work

The analysis of large-scale datasets has become a critical challenge because of the management, transport, and access to the data. This is especially true for scientific simulation datasets such as earth's climate system in which the size of the dataset is hundreds of TBs [3] and data is geographically distributed. The efforts and challenges of transferring and accessing climate datasets have been shown in [11], [2].

Compression [13], [9] has been an attractive topic for systems that deal with large datasets. The ultimate goals of compression are to decrease the storage requirements of the data, and maximize the I/O throughput of the systems. Nicolae *et al.* developed BlobSeer, which is a distributed data management service that provides efficient reading, writing and appending functionalities to its users [16]. A recent work by the same authors focuses on a transparent compression system that is built in BlobSeer [15]. Their system provides optimizations such as overlapping decompression with I/O operations, and selecting the compression algorithm.

ISABELA-QA [12] is a parallel query processing engine which exploits *knowledge priors*. It enables efficient processing for both spatial-region and variable-centric queries using compressed scientific datasets and indices.

Welton *et al.*, developed a set of compression services which transparently performs compression operations [18]. Their services run on data nodes which handle I/O requests. Authors show that compression is more beneficial when I/O rate of the environment is low, and the computational demands of the compression algorithm is small.

Our compression system focuses on implementation of domain-specific compression algorithms, and improving their efficiency through concurrency in different layers. We specifically focus on multidimensional scientific datasets in which such domain specific compression algorithms can easily be

implemented and result in high compression ratios and performance.

Data-intensive computing is one of the areas which uses compression extensively. Apache has initiated several projects for large scale data analysis and storage, such as Hadoop and HBase. These systems use generic compression algorithms, such as LZO, in order to optimize I/O operations. Similarly, Google's data storage and analysis system, MapReduce[8] and BigTable[6], take advantage of different compression algorithms, *e.g.* Snappy. Another data analysis system, DryadLINQ[20], also uses compression for its intermediate data exchange, thus enhancing the I/O throughput.

During the evaluation of our work, we ported our system into PNetCDF. Our results show that applications can further maximize the utilization of I/O resources through *domain-specific compression algorithms* and *parallelization*.

Compression has also been used for enhancing the performance and storage of cache and memory[10], [17], [19]. A recent work of Makatos et al. introduced FlaZ, a transparent compression system, which enhances the performance of SSD-based caches [14]. FlaZ, works in the kernel level and resides in I/O path of user application and SSD. It provides two generic compression algorithms, zlib and LZO, and I/O concurrency. If the storage medium can handle parallel I/O requests, e.g. SSDs, concurrent operations can significantly increase the performance. Our work also considers the high-degree of parallelism in these type of storage units. Furthermore, it uses prefetching and in-memory caching to increase performance.

There have been studies about the effects of using compression on energy efficiency of data-intensive applications and modern datacenters. In [7], authors propose a decision algorithm which lets users identify if compression is beneficial or not in terms of energy consumption. Authors implement their system in MapReduce and show that compression can increase the energy efficiency up to 60%. Our system uses CPU cycles for concurrent decompression and I/O requests in order to increase the performance of the system. This can result in higher energy consumption, however we believe that the gained data transfer time through compression can amortize and further improve the energy efficiency.

III. Proposed Compression Framework for Scientific Datasets

In this section we describe our compression framework and method which can be applied to different types of multidimensional scientific datasets.

A. Proposed Compression Framework

Our compression framework provides transparent (de)compression operations with highly parallel computation and I/O capabilities. Therefore, clients can interact with compressed datasets as if they are original dataset. Our framework offers several important features:

An API which eases the implementation of different compression algorithms:
Our compression framework requires two basic functions to be implemented, *encode* and *decode*. The *encode* function is responsible for compressing the given data chunk, whereas the *decode* function decompresses the compressed data chunk. After these functions are implemented, the framework automatically divides user provided data into slices and applies these functions. For example, consider a climate dataset which consists of temperature information of an area at different times. This information can be stored in three dimensional array, namely *time*, *longitude* and *latitude*. The compression framework can slice the dataset on time and longitude dimensions. Then, it can transparently (de)compress these chunks during read and write operations.

Optimizations on data access:
Our framework applies several optimizations in order to increase the performance of data access operations. The first optimization is the pipelined chunk processing. Since the dataset is divided into small independent chunks, they can concurrently be processed. Specifically, our system can fetch multiple chunks at a time, and process them simultaneously. Second, our framework provides an interface in which user can analyze the previously accessed chunks, and define customized prefetching algorithms. This enables our system to request compressed chunks in advance, and cache them. The results of this part can be found in our previous work [5].

Determining the best compression algorithm:
A user can implement a set of compression algorithms using our compression framework. However, finding the optimal compression algorithm for a given dataset is a challenging problem. Considering the size of the scientific dataset, it is typically infeasible to apply all compression algorithms. In order to find the best compression algorithm, datasets can be sliced and sampled. Then implemented compression algorithms can be applied to the samples. Once the best performing algorithm is determined, this information can be stored as metadata information, and the remaining chunks can be compressed.

Finding the optimal chunk sizes:
Compression algorithms are sensitive to chunk sizes. Typically, larger chunk sizes result in better compression ratios. However, this might translate into longer compression times and redundant data decompression. For example, the compression framework needs to decompress the whole chunk, even though user wants to access a small portion. Therefore, it is important to find the optimum chunk size for best performance.

Application of compression algorithm:
Similar to detect the best compression algorithm, it is also important to determine how the algorithm is being applied to the dataset. The slice operation can be applied to different dimensions. Moreover, the data items in a slice may logically be related. For instance, if we consider the aforementioned climate dataset again, we can expect temperature values to be more related on latitude dimension. This information can be

exploited in order to increase the compression ratio.

We implemented and ported our prototype in Parallel NetCDF. PNetCDF uses ROMIO, an MPI-IO library, for concurrency. Thus, it can provide efficient access to scientific datasets. Our system performs (de)compression operations before MPI-IO calls. Therefore, only the compressed data is transferred through network, which maximizes the bandwidth utilization.

B. Compression Method

Our compression method is based on *differential compression*. After analyzing the scientific datasets which are stored in multidimensional arrays, we observed that adjacent data elements' values are typically related with each other. Our compression methodology exploits this observation and provides high I/O throughput along with decent compression ratios.

As an example, consider the aforementioned climate dataset values which consist of temperature of different altitudes at a given area in different time slices. Since adjacent locations are expected to have similar temperature values, a dataset specific differential compression algorithm can be developed.

Algorithm 1: Climate Data Compression

Input: temp_latitude, n_longitude, n_cells
Output: comp_temp

for $j = 0$ **to** n_cells $- 1$ **do**
  comp_temp$_{0,j}$ = temp_latitude$_{0,j}$;
for $i = 1$ **to** n_longitude $- 1$ **do**
  for $j = 0$ **to** n_cells $- 1$ **do**
    cell$_0$ = temp_latitude$_{i-1,j}$;
    cell$_1$ = temp_latitude$_{i,j}$;
    comp = cell$_0 \oplus$ cell$_1$;
    n_zeros = clz(comp);
    append(comp_temp, pack(n_zeros, comp));

In Algorithm 1, we show the pseudocode of our climate data compression. The algorithm reads the temperatures of different altitudes with *temp_latitude* matrix. The columns and the rows of the matrix correspond to the longitudes and altitudes, respectively. The algorithm uses *exclusive or*, $\oplus$, operation to relate neighbor cells. Notice that the computed value, *comp*, depends on the cells that are on the same latitude and altitude, but on different longitude. This approach exploits the temperature information of the cells. Specifically, the temperatures of neighboring cells which are at the same altitude and latitude are close to each other. Therefore more information can be extracted and used for compression through relating these cells. The xor operation results in a variable which can be examined in two parts: *leading zeros* and *difference*. The difference part represents the bits that differ $cell_0$ from $cell_1$ whereas the leading zeros show the overlapping data. Next the algorithm counts the number of leading zeros and packs them with difference part. We assume that *append* function adjusts the *comp_temp*, thus the next iteration writes to the next available bit position. For single-precision floating point numbers, the total number of zeros can be up to 32 which can be represented by 5 bits. Therefore, this approach can compress climate data with a maximum of 32:5 compression ratio. Decompression algorithm reverses the operations. First, the compressed values are read, and the leading zeros are expanded. Then, $\oplus$ is applied to the resulting values and the original data is derived.

Since both compression and decompression operations rely on the xor operation, the performance of the algorithm is very high. The effectiveness of the compression method and its comparison with other algorithms can be found in [5].

IV. Experimental Results

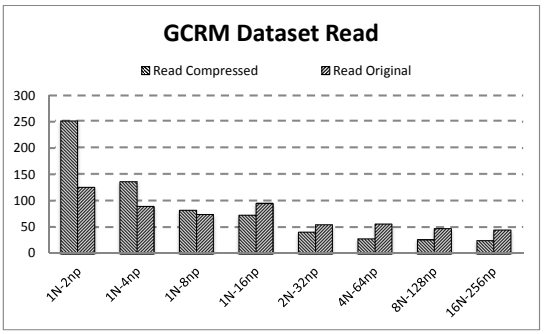
In this section, we present the results of our preliminary experiments. Specifically, we evaluated the performance of the aforementioned compression framework.

We used GCRM simulation dataset for our evaluation. This is a multidimensional dataset where the dimensions consist of time step, longitude, latitude and altitude information, respectively. Each of the cells in the dataset consists of a single-precision floating point number which refers to the temperature of a given location at a specific time. The total size of the dataset is 68GB which covers 28km with 27 layers, and the total number of time steps is 256.

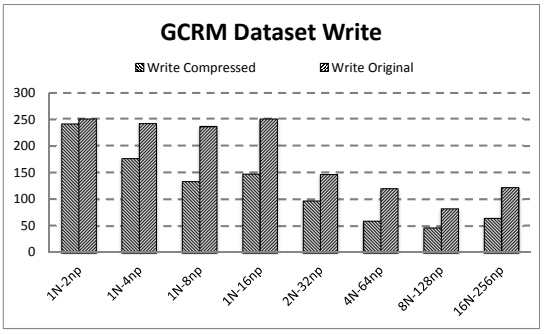
We performed our experiments on the Olympus cluster at Pacific Northwest National Labratories. Two versions of dataset are created, compressed and original. The compressed dataset size is 35.4GB. In order to provide high I/O throughput, datasets are stored in Lustre parallel file system. The total number of Object Storage Targets (OSTs) is set to 8, and the stripe size is configured as 1MB. We used up to 16 machines for evaluation of the performance of our approach. Each machine has 32GB memory and 16 physical cores, where each of these cores is an AMD Opteron 6272 with a 2.1GHz clock rate.

In Figure 1(a), we present the execution times of read operation with increasing number of compute nodes. The *Read Original* version shows the read performance of the original data, whereas the *Read Compressed* represents the compressed dataset.The Y-axis shows the execution times in seconds. The *N* is the number of allocated nodes, and *np* is the total number of processes that run on these nodes. For example, 2N-32np represents the configuration where 2 nodes are allocated, and 32 MPI processes are launched.

The original data read operation performs better than the compressed read operation from 2np to 8np configuration. However, after 8np the compressed version provides better I/O throughput. Since the number of OSTs is set to 8, the original dataset can be read efficiently up to 8 processes. However after this configuration, the I/O resources become scarce. Therefore, the read operations start benefiting from the compression. The speedups of using compression after 8np configuration range between 1.31 and 1.98.



(a) Read Operations



(b) Write Operations

Fig. 1. Evaluation of Data Access Operations with Compression

In Figure 1(b), we show the performance of write operation with and without compression. The compressed version performs better than original version in all configurations. Since the write operations can return before the data is actually written to the disk, compressed version shows better performance.

V. Conclusion

In this work, we focus on management of multidimensional scientific datasets using domain specific compression algorithms. We have built and ported our framework into PNetCDF. Our preliminary results show that modified version of differential compression is effective in both compression ratio and performance.

In the future, we would like to investigate different ways to improve compression ratio and performance. These include implementing more domain-specific compression algorithms and finding the best one for a given dataset; detecting the best chunk size for compression; and finding the application direction of compression algorithm to the data chunks.

Acknowledgment

This work was carried out in collaboration with David Chiu at Washington State University, Jian Yin and Karen Schuchardt at Pacific Northwest National Laboratory.

References

[1] Parallel NetCDF: A High Performance API for NetCDF File Access. http://www.mcs.anl.gov/parallel-netcdf, 2012. [Online; accessed September-2012].

[2] W. E. Allcock, I. T. Foster, V. Nefedova, A. L. Chervenak, E. Deelman, C. Kesselman, J. Lee, A. Sim, A. Shoshani, B. Drach, and D. N. Williams. High-performance remote access to climate simulation data: a challenge problem for data grid technologies. In *SC*, page 46, 2001.

[3] D. E. Bernholdt, S. Bharathi, D. Brown, K. Chanchio, M. Chen, A. L. Chervenak, L. Cinquini, B. Drach, I. T. Foster, P. Fox, J. Garcia, C. Kesselman, R. S. Markel, D. Middleton, V. Nefedova, L. Pouchard, A. Shoshani, A. Sim, G. Strand, and D. N. Williams. The earth system grid: Supporting the next generation of climate modeling research. *CoRR*, abs/0712.2262, 2007.

[4] T. Bicer, D. Chiu, and G. Agrawal. A framework for data-intensive computing with cloud bursting. In *CLUSTER*, pages 169–177, 2011.

[5] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt. Integrating online compression to accelerate large-scale data analytics applications. In *IPDPS*, 2013, To be published.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2), 2008.

[7] Y. Chen, A. Ganapathi, and R. H. Katz. To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency. In P. Barford, J. Padhye, and S. Sahu, editors, *Green Networking*, pages 23–28. ACM, 2010.

[8] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, 2008.

[9] F. Douglis. On the role of compression in distributed systems. *SIGOPS Oper. Syst. Rev.*, 27(2):88–93, Apr. 1993.

[10] M. Ekman and P. Stenstrom. A robust main-memory compression scheme. In *Computer Architecture, 2005. ISCA '05. Proceedings. 32nd International Symposium on*, pages 74 – 85, june 2005.

[11] R. Kettimuthu, A. Sim, D. Gunter, B. Allcock, P.-T. Bremer, J. Bresnahan, A. Cherry, L. Childers, E. Dart, I. Foster, K. Harms, J. Hick, J. Lee, M. Link, J. Long, K. Miller, V. Natarajan, V. Pascucci, K. Raffenetti, D. Ressman, D. Williams, L. Wilson, and L. Winkler. Lessons learned from moving earth system grid data sets over a 20 gbps wide-area network. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC 2010)*, Jun 2010.

[12] 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. Ross, and N. F. Samatova. Isabela-qa: query-driven analytics with isabela-compressed extreme-scale scientific data. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 31:1–31:11, New York, NY, USA, 2011. ACM.

[13] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Comput. Surv.*, 19(3):261–296, 1987.

[14] T. Makatos, Y. Klonatos, M. Marazakis, M. D. Flouris, and A. Bilas. Using transparent compression to improve ssd-based i/o caches. In C. Morin and G. Muller, editors, *EuroSys*, pages 1–14. ACM, 2010.

[15] B. Nicolae. High throughput data-compression for cloud storage. In A. Hameurlain, F. Morvan, and A. Tjoa, editors, *Data Management in Grid and Peer-to-Peer Systems*, volume 6265 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2010. 10.1007/978-3-642-15108-81.

[16] B. Nicolae, G. Antoniu, and L. Bougé. Blobseer: how to enable efficient versioning for large object storage under heavy access concurrency. In *Proceedings of the 2009 EDBT/ICDT Workshops*, EDBT/ICDT '09, pages 18–25, New York, NY, USA, 2009. ACM.

[17] L. Rizzo. A very fast algorithm for ram compression. *SIGOPS Oper. Syst. Rev.*, 31(2):36–45, Apr. 1997.

[18] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. B. Ross. Improving i/o forwarding throughput with data compression. In *CLUSTER*, pages 438–445, 2011.

[19] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis. The case for compressed caching in virtual memory systems. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '99, pages 8–8, Berkeley, CA, USA, 1999. USENIX Association.

[20] Y. Yu, M. Isard, D. Fetterly, M. Budiu, Ú. Erlingsson, P. K. Gunda, and J. Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. In R. Draves and R. van Renesse, editors, *OSDI*, pages 1–14. USENIX Association, 2008.