Terascale Data Organization for Discovering Multivariate Climatic Trends

Wesley Kendall, Markus Glatter, and Jian Huang
Department of Electrical Engineering and Computer Science
The University of Tennessee, Knoxville
Knoxville, TN 37996
{kendall, glatter, huangj}@eecs.utk.edu

Tom Peterka, Robert Latham, and Robert Ross
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{tpeterka, robl, rross}@mcs.anl.gov

ABSTRACT

Current visualization tools lack the ability to perform full-range spatial and temporal analysis on terascale scientific datasets. Two key reasons exist for this shortcoming: I/O and postprocessing on these datasets are being performed in suboptimal manners, and the subsequent data extraction and analysis routines have not been studied in depth at large scales. We resolved these issues through advanced I/O techniques and improvements to current query-driven visualization methods. We show the efficiency of our approach by analyzing over a terabyte of multivariate satellite data and addressing two key issues in climate science: time-lag analysis and drought assessment. Our methods allowed us to reduce the end-to-end execution times on these problems to one minute on a Cray XT4 machine.

Keywords

Query-Driven Visualization, Parallel I/O, Temporal Data Analysis, MODIS

1. INTRODUCTION

To understand the underlying structures and relationships of variables in datasets through space and time, it is often necessary to analyze and visualize the full spatial and temporal extent of the dataset. The need for scalable methods to perform such full-range analysis tasks is growing as scientific simulations produce datasets ranging to the terascale and beyond. This need is particularly acute as visualization applications, especially those that handle large-scale time-varying data, are increasingly being dominated by I/O overheads to a degree that impedes their practical use.

As an example of the challenge that I/O presents in visualization, [1] has shown that rendering a single timestep of a 2048^3 volume can be optimized to take only a few seconds on an IBM Blue Gene/P, however, reading that volume from disk takes at least a minute. Efficient I/O solutions are needed if one wishes to perform more complex visualization and analysis tasks that involve many timesteps to be loaded at once. In order for visualization to make a greater impact on terascale computational science, I/O cannot be treated in isolation.

In this work, we developed a system that closely integrated techniques in parallel I/O with concepts from parallel query-driven visualization. Our contributions include: first, to alleviate I/O bottlenecks in full-range analysis through advanced methods of performing I/O on common scientific data storage formats; second, to scale parallel extraction of salient spatial and temporal regions in scientific datasets by improving current parallel data querying techniques; and third, to assess multivariate relationships in terascale observational data by using our techniques to alleviate computing and I/O overheads.

The scalability of our system enabled us to explore over a terabyte of multivariate satellite data from NASA's Moderate Resolution Imaging Spectroradiometer (MODIS) [2] project. The amount of data and the desired combinatorial way of exploring for conceptual events, such as prolonged periods of drought and event relationships, required using computing resources at large scale.

With scalability up to 16K cores, our full-range studies on ≈1.1 TB of MODIS were optimized to take around one minute. This time includes all stages of the pipeline, from the initial I/O operations to the resulting visualization. The ability to perform these complex analysis tasks at large scales on datasets directly from simulations provides a flexibility that is not offered by current visualization tools. The following discusses our work in reducing the end-to-end latency of these complex analyses.

SC09 November 14-20, 2009, Portland, Oregon, USA.

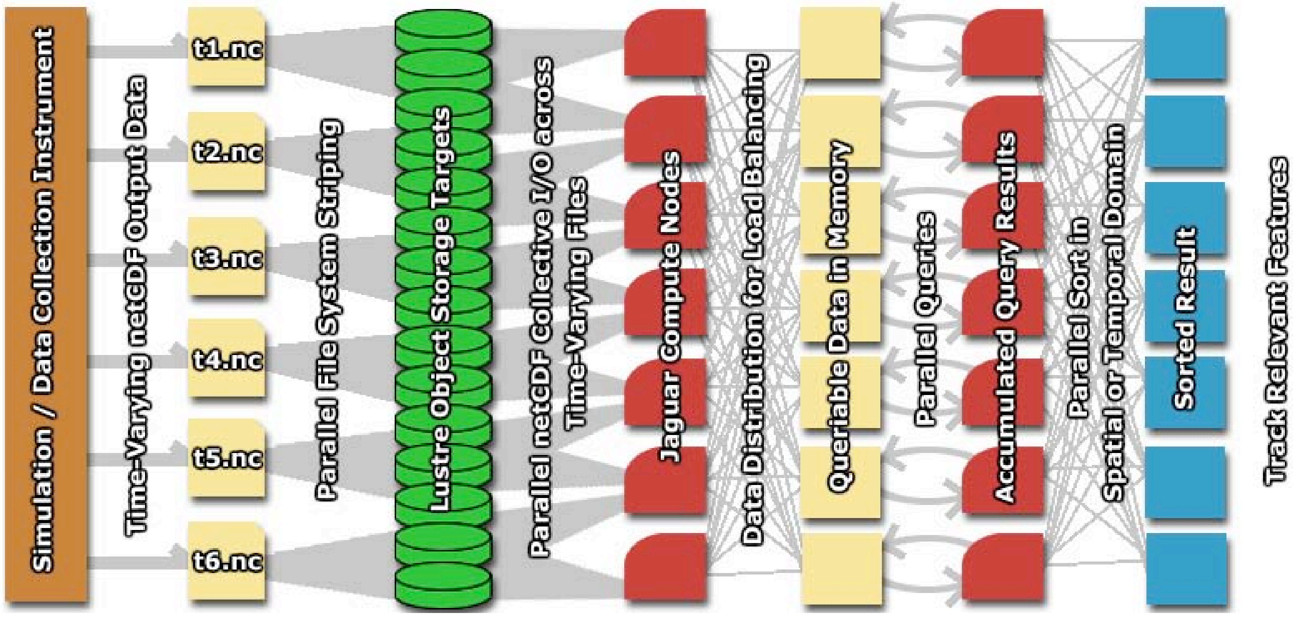
Simulation / Data Collection Instrument
Time-Varying netCDF Output Data
t1.nc
t2.nc
t3.nc
t4.nc
t5.nc
t6.nc
Parallel File System Striping
Lustre Object Storage Targets
Parallel netCDF Collective I/O across Time-Varying Files
Jaguar Compute Nodes
Data Distribution for Load Balancing
Queriable Data in Memory
Parallel Queries
Accumulated Query Results
Parallel Sort in Spatial or Temporal Domain
Sorted Result
Track Relevant Features

Figure 1: An overview of our system.

2. OVERVIEW AND RELATED WORK

Our work primarily spans the fields of parallel I/O, feature tracking, and parallel query-driven visualization. Little research exists that systematically combines these areas, and each area is crucial for our driving application. Before discussing the relevant related work, we present a synopsis of the driving application and our system, along with a description of the Cray XT4 machine we used.

2.1 The Driving Application

Our driving application is to study observational data in NASA's MODIS database to discover multivariate climatic trends. The dataset used in our study consists of a 500 meter resolution sampling of North and South America, creating a 31,200 by 21,600 grid. The dataset is continuously updated, and we used 417 timesteps of 8 day intervals from February 2000 to February 2009. MODIS data is stored in various wavelength bands which may be used to compute other variables. By computing two variables that are related to our studies in Section 5 and storing them as short integers, the entire dataset totals to ≈1.1 TB. With a dataset of this magnitude, we have two primary goals: to provide visualization and analysis methods for cases when many of the timesteps of the dataset need to be loaded, and to deliver usability and near-interactive functionality specifically for application scientists.

The visualization aspect of this task is very demanding for several reasons. First, extracting isocontours with an inherent temporal component is rather new in the field. Drought, for instance, is not a single timestep event and requires "abnormally low rainfall" to have lasted for a period of time. Although tracking contours over time has already been solved [3], visualization typically still treats contour extraction statically in the spatial domain [4]. We instead seek for spatial locations in a contour that fit two criteria: thresholds for scalar variables, and the number of continuous timesteps that meet the first criteria. Advanced data structures are necessary to allow for such high dimensional searches. Unfortunately, the most likely methods and data structures, particularly those used in query-driven visualization, incur preprocessing overheads on the order of several hours for datasets of 100 GB [5, 6, 7].

Second, current data analysis tools are not efficiently integrated with system level tools such as parallel I/O. This causes a severe bottleneck in efforts to shorten the end-to-end latency as outlined by the vision of in situ analysis and visualization [8]. In addition, few current data analysis tools have the scalability to leverage modern systems, such as the Cray XT4 and the IBM Blue Gene/P. Not being able to fully leverage systems of that caliber, especially the next generation storage and parallel I/O systems, imposes yet another bottleneck in obtaining the full potential of data analysis tools in production scientific use.

Third, typical user work flow includes an often neglected but very expensive component. That is, to get the datasets from an application-native format into a format that is the most amenable to the parallel data analysis or visualization. To minimize the end-to-end latency, this step must be studied in depth. In this work, we specifically focused on one of the most common cases of our collaborators, where individual timesteps are initially stored in separate netCDF files.

Figure 1 illustrates the overall architecture of our data analysis system. When data is produced, it is stored in an application-native format and striped across the parallel file system for higher access bandwidth. Our system reads and distributes data in parallel to prepare it into a queriable form with maximal runtime load balance. The relevant data is then queried and spatial or temporal analysis may occur by sorting the data in the proper ordering.

The system provides a means for analysis to take place on datasets immediately after being written to disk, and it also reduces computing overheads by using query-driven concepts to access only the data deemed relevant. By using these concepts, problems of full-range analysis and visualization on terascale datasets can be solved in feasible amounts of time. In the following, we further discuss our targeted infrastructure and the previous works that have influenced our design decisions.

2.2 Targeted Infrastructure

In our study, we used the Jaguar Cray XT4 machine located at Oak Ridge National Laboratory as a testing ground for our methods. Jaguar consists of 7,832 quad-core 2.1 GHz AMD Opteron processors with 8 GB of memory. The system totals to 31,328 cores with 62 TB of main memory. The parallel file system on Jaguar is the Lustre file system [9]. Lustre is an object-based parallel file system, consisting of Object Storage Servers (OSSs) that handle I/O requests from the clients to the actual storage. Each OSS serves one or more Object Storage Targets (OSTs), which are the physical disks. Lustre uses one Metadata Server (MDS) that holds the file information for the entire system. Jaguar's Lustre file system contains 72 OSSs that serve 144 OSTs.

2.3 In Situ Processing and Visualization

Ma et al. [8] described the various bottlenecks present in popular methods of large data processing and visualization. The authors emphasized the importance of being able to couple visualization and analysis into the simulation pipeline. This allows for reduction of data and valuable analysis during the simulation that can further accelerate the scientific discovery process.

The authors further discussed various postprocessing steps that are common in the visualization and analysis pipeline. With in situ visualization, many of these steps can be reduced or even taken out. Although we do not couple our code directly with simulation code, we provide a means to interact with data written directly from the simulation, along with conducting postprocessing on the fly.

2.4 Parallel I/O

Significant algorithmic advances have improved the usability and portability of parallel I/O across high-performance systems, and user-friendly interfaces have been developed to aid in achieving higher I/O bandwidths. I/O interfaces built on top of the MPI-2 [10] standard, such as ROMIO [11], have shown various optimizations that can be applied to the I/O phase for various parallel data access patterns. Collective I/O and data sieving are such examples [11]. We used optimizations such as collective I/O as a means to achieve better bandwidth rates in our I/O phase.

I/O has recently been gaining more attention in visualization research. Ma et al. [12] first showed how overlapping I/O with rendering could significantly reduce interframe delay in the parallel rendering of large-scale earthquake simulations. Yu et al. [13] extended this by presenting I/O solutions for a parallel visualization pipeline. Yu et al. [14] also presented two parallel I/O methods for the visualization of time-varying volume data in a high-performance computing environment.

Peterka et al. [1] have recently performed extensive work on large-scale parallel volume rendering of astrophysics data on the IBM Blue Gene/P. Their method of visualization focused on efficiently using parallel I/O to increase frame rates. By using a system like the Blue Gene/P, the authors showed that a simulation-caliber resource is a valuable alternative to a graphics cluster for visualization, especially when I/O is the bottleneck.

Although our primary systems need is to efficiently read data from storage and organize it in memory, we are aware of systematic efforts underway to address challenges of outputting data by scientific simulations at the petascale. One effort in that regard is the Adaptive I/O System (ADIOS) [15], which provides input and output performance gains and a flexibility of runtime choices of different I/O methods and data formats. Systems like ADIOS can treat visualization as just another I/O method, enabling in situ visualization.

2.5 Feature Extraction and Tracking

One of the most widespread uses of feature detection in visualization is to extract isocontours from scalar fields. Isocontour extraction can be explicit or implicit. Volume rendering methods [21] implicitly extract isocontours by using transfer functions to assign non-zero opacity only to a few selected continuous narrow ranges of scalar values. The marching cube algorithm [4] explicitly constructs the geometry that corresponds to one single scalar value or a continuous range of scalar values.

Silver et al. first tracked explicitly extracted features over time [3]. By finding spatial overlaps between features from neighboring timesteps, their methods discovered the evolutionary history of contours. Tzeng and Ma [22] were the first to demonstrate that features evolve not only spatially but also in value space over time. In other words, the values that define the contours of a feature do not remain constant across all timesteps. Thus, features cannot be reliably tracked without considering the variations in the values of the isocontours. We know of no previous research that has considered directly extracting contours with an inherent temporal element, such as the drought problem discussed in Section 5.

2.6 Query-Driven Visualization

Query-driven visualization [5, 16, 17, 6, 7] has recently become a popular research topic. Query-driven methods are related to feature extraction in several of ways. First, query-driven methods rely on translating a user interest into a compound Boolean range query. In doing so, it offers a natural extension of feature extraction capabilities in multivariate data. Second, regardless of the underlying methods, such as a distributed M-ary search tree [5], bitmap indexing [7] or bin-hashing [6], query-driven visualization has been shown to accelerate contour extraction in large datasets. The methods for query-driven visualization can be split into two categories: tree-based and index-based methods.

In indexed-based methods [18, 7, 19], bitmap indexing is used to accelerate the search phase. To build the index, each record r with v distinct attribute values generates v bitmaps with r values each. This allows for bitwise logical operations to be performed on range queries, making multivariate range queries simple linear combinations of single-valued queries [7]. In tree-based methods, data is sorted and indexed with tree structures such as B-trees or M-ary search tree structures [5].

It has been shown that tree-based methods suffer from the "Curse of Dimensionality", where adding more dimensions results in an exponential growth in storage and processing requirements [7]. Although this is true for a naive tree method, Glatter et al. [5] showed how an M-ary search tree structure could be used to reduce the storage overhead to less than 1% of the dataset size and also showed that the overhead is not dependent on the dataset size. This factor is attractive to our method of querying since we load the data in main memory. It was also shown in [5] that queries could be load balanced by distributing data on the granularity of single voxels. Although parallel methods of index-based query-driven visualization have been established [18, 19], these do not focus on the issue of load balanced data extraction. Since load balancing is crucial to the performance of our system, we used [5] as the background for our parallel querying model.

2.7 Parallel Sorting

We used parallel sorting for applying two of our load balancing schemes discussed in Section 4, and for performing parallel temporal analysis discussed in Section 5. Because of the results in the seminal work by Blelloch et al. [20], we chose to use the parallel sample sort algorithm.

The parallel sample sort algorithm randomly samples the dataset choosing samples in such a way to linearly split the entire dataset. The processes then bin their data based on the splitter samples and sort their bins, creating a globally sorted list. Parallel sorting is an inherently network-intensive process, and this algorithm sums up all of its network communication in one MPI_Alltoall call to bin the dataset. To locally sort the bins, we used the quick sort algorithm.

3. THE I/O COMPONENT

Visualization applications are often dominated by I/O time when working with terascale datasets, and this can be exacerbated with simplistic methods such as pushing the entire I/O request through one node [23]. We used an advanced method for performing I/O on application-native formats to reduce I/O costs and allow for analysis with no postprocessing step.

3.1 Design Considerations

It is common for our collaborators to use the netCDF [24] format to store their data, so we used the Parallel netCDF library [25] as a layer for our disk access. This library is built on top of MPI-2 [10], allowing it to execute on a wide variety of high-performance architectures independent of the parallel file system that is installed.

Another common storage method for our collaborators is to store a separate file for each timestep or variable. Thus, it is necessary to have an algorithm that performs I/O on multiple files in an efficient manner. The research that most closely resembles our work is by Memik et al. in [26], where they examine reducing I/O and data transfer times in collective methods to multiple files. They showed this problem was NP-complete when the files were arbitrary sizes and provided various solutions to the problem. In our system, the assignment of data to processes is irrelevant, and we only require that all the data be loaded in memory after the I/O step. Because of this, the step of minimization of transferring data to the correct process shown in [26] was not needed, and we were able to create a more general algorithm.

It was shown in [26] that a greedy heuristic could be used when assigning processes to files. We used a similar approach and applied a greedy heuristic that maximizes the amount of complete files that can be read at one time. We used this heuristic because it allows us to perform large collective reads on the files and efficiently use collective I/O strategies implemented in ROMIO [11]. Collective I/O is useful primarily for three reasons. First, it was shown in [27] that parallel opening of shared files is much more effective on large amounts of OSTs. Second, collective I/O can result in larger reads that result in higher bandwidths. Third, we can tune the size of the collective I/O buffer for better bandwidths. We used a buffer size of 32 MB because of the results obtained in [27].

3.2 Greedy I/O on Multiple Files

Our approach for performing I/O on multiple files is not limited by the number of processes or the number of files. The routine proceeds in the following steps:

1. A configuration file is read with information about the dataset. This information includes the directory where data is stored, the variables of interest, restrictions or subsampling rates on the dimensions of the variables, and the ranges of variables that are unimportant.
2. Each file is assigned round robin to the N processes. The files are opened by the respective processes, and the metadata about the all the variables is gathered. The total size of the dataset D is computed, and each process must read in approximately Q amount of data where $Q = \frac{D}{N}$. Each process P contains D_P amount of data that has been read in, with D_P starting at 0.
3. The first process P that has $D_P < Q$ reserves the largest chunk possible of the first file F that has not yet been fully read in. If $sizeof(F) \leq Q - D_P$, P will be able to reserve all of the data available to read in F. If $sizeof(F) > Q - D_P$, P will reserve $Q - D_P$ amount of data from F. This will continue on the remaining processes where $D_P < Q$ until all of F is reserved or until no more processes are available.
4. Once a group of processes has been assigned to a file, an MPI communicator is created for the group and collective parallel I/O is performed on the variables. For each process P that performed I/O, D_P is incremented by the amount of data read in.
5. If $D_P < Q$, process P repeats steps 3 and 4 with the remaining files in subsequent stages. This continues until all processes have read in Q amount of data.

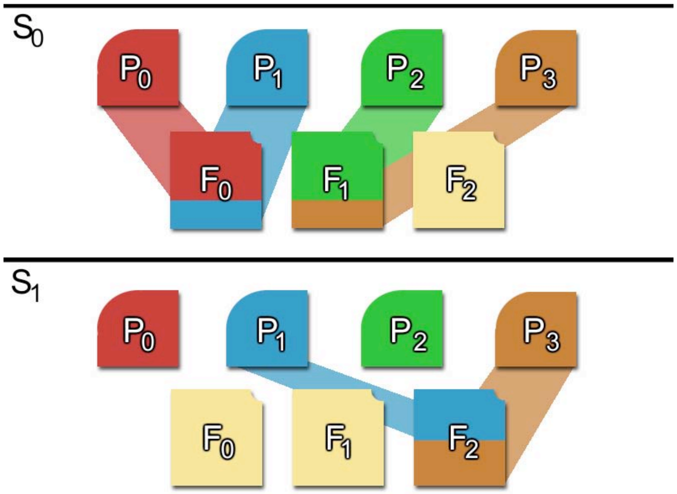

Figure 2: Example of the greedy I/O algorithm with four processes and three files.

The greedy approach is best shown by a simple example in Figure 2. This example conducts the algorithm on three files while using four processes. In stage S_0, P_0 and P_2 are able to read in their respective Q amount of data. P_1 and P_3 are not able to do this, so they continue to the next stage S_1. In this stage, they read in F_2 and finish reading in their respective Q amount of data.

The main limitation of this algorithm is the amount of data that can be loaded in the aggregate memory of the nodes. Two strategies are provided to address this issue. First, users are able to specify ranges of variables that are irrelevant to their analysis in the dataset configuration. Since the algorithm executes in stages, these values are discarded in between stages, thus using memory more efficiently. Second, subsampling rates or restrictions may be specified in the dataset configuration and applied to the dimensions, enabling a reduced version of the dataset to be read in.

3.3 Results and Comparisons

We tested the bandwidth rates for this algorithm on varying scales with the MODIS dataset. If the aggregate memory of the nodes was insufficient, data was discarded after being read in for bandwidth testing purposes. It is difficult to compare the bandwidth rates of our algorithm with benchmark results on the Jaguar machine because netCDF benchmarks do not exist for this machine. Therefore, we compare our results with the IOR benchmark [28] results obtained in [27] as a comparison for how our algorithm performs at large scales.

The IOR benchmark provides the ability to test aggregate I/O rates with MPI collective I/O calls on various file access modes and access patterns. In [27], the authors used the IOR benchmark on Jaguar to show how the bandwidth scales when using varying amounts of OSTs to read in data. We use the bandwidth results obtained from using 144 OSTs in [27] as a comparison since we also used 144 OSTs. Two benchmarks from [27] using 1K cores showed bandwidth rates of ≈42 GB/s when reading in one shared file and ≈36 GB/s when reading in one file per process. Other benchmarks were conducted that used greater than 1K cores, but the amount of OSTs used was not explicitly stated and the bandwidth rates were less than those of the IOR benchmark.

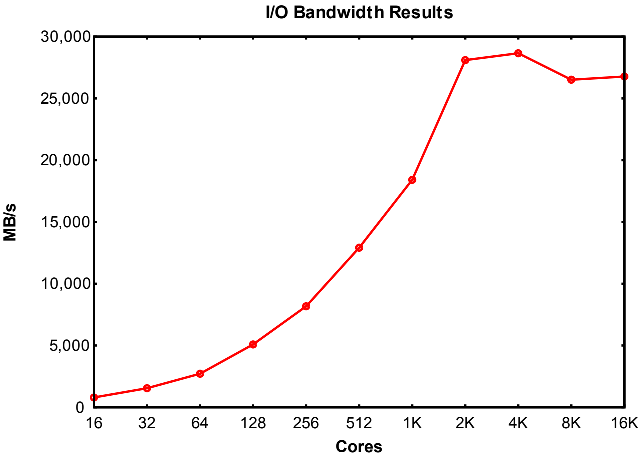

Figure 3: I/O bandwidth results for the greedy algorithm on the MODIS dataset.

Bandwidth results of our algorithm are shown in Figure 3. When timing the results, the files were first opened and then timing was started after an MPI_Barrier. After all the I/O was complete, another MPI_Barrier was called and timing stopped. We achieved up to ≈28 GB/s on 4K cores, roughly 75% of the 42 GB/s benchmark comparison. The bandwidth bottomed out around 2K cores, and performance degradation was seen when scaling to 8K and 16K cores. This same trend was observed in [27] on a separate benchmark. We hypothesize that the MDS is being overwhelmed with requests when going to this scale, but further testing is required to confirm this.

When scaling to 4K cores, we were able to minimize the I/O time on the entire 1.1 TB MODIS dataset to ≈37 seconds. The results show that using collective I/O combined with a greedy approach to saturate the underlying I/O system with requests can achieve significant bandwidth rates, and data in application-native formats can be read in a practical amount of time for an application setting. Because we can access datasets stored in these application-native formats, common postprocessing steps taken in large data visualization [8] can be taken out of the analysis pipeline, saving valuable time and allowing for prototyping and analysis immediately after the simulation is complete.

4. SCALABLE DATA EXTRACTION

Because of the low storage overhead and load balanced data extraction capabilities discussed in Section 2.6, we chose to use the work by Glatter et al. [5] as a basis for our query-driven data extraction. However, the methods in [5] were only scaled up to 40 servers, studied only one approach of load balancing, and used a server-client method that was inherently bottlenecked by one client collecting queried data over a network. The following discusses our methods to extend these concepts on large systems and to further improve the scalability and timing of our query-driven data extraction.

4.1 Our Query-Driven Model

Instead of using a server-client concept that returns data over a network from many servers to one client as in [5], we modeled our query-driven system as a set of independent processes issuing the same query on local data and accumulating the result in memory.

For fastest querying rates, it would be ideal to have equal amounts of returned data on every process for all the queries issued. Although it is impossible to know which queries will be issued, it has been shown that distributing the data in a spatially preserving manner across all the processes results in near-optimal load balance regardless of the query [5]. However, distributing data in [5] involved costly computation of Hilbert indices while doing a sort of the entire dataset.

It is then a question of trade-off between the overhead to distribute data after I/O versus the overhead of a large number of runtime queries. To find a practical optimum in this trade-off, we studied the design in [5] plus four alternative designs, each incurring contrasting overheads of distribution and runtime querying. The starting point of all designs is the same: each node has read in contiguous segments of data from disparate netCDF files. In the following, we describe the load balancing schemes. We refer to a data point being distributed as an *item*, which is the multivariate tuple on each (x, y, t) location in the MODIS dataset.

Hilbert-Order: The original design in [5]. The items are globally sorted by their computed indices along the Hilbert space-filling curve, and then distributed by a round robin assignment to all processes.

Z-Order: The same as Hilbert-Order, except computing indices along the Z space-filling curve.

Round Robin: Each item is distributed by a round robin assignment to all processes.

Random: The items are randomly shuffled locally and then divided into N chunks, where N is the number of processes. Each process P will then gather the P^{th} chunk from each process. This entire method is performed twice.

None: No data distribution.

4.2 Load Balancing Tests and Results

We tested these load balancing schemes to arrive at quantitative conclusions on which ones are best for given applications. We assessed these techniques based on two criteria: the cost associated with distributing the data, and the average time spent querying.

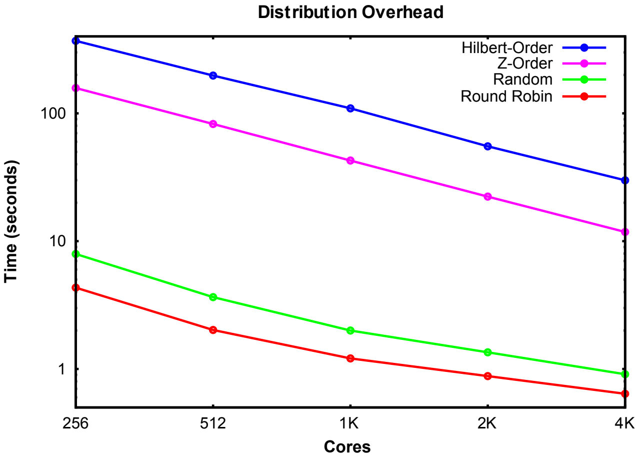

Figure 4: The overhead times for the load balancing schemes. Time is shown on logarithmic scale.

The cost of distributing the data is simply the distribution time. The average time spent issuing a query is calculated by taking the maximum query time of the individual processes, and then averaging it over all the queries. Doing this provides a measurement of how load imbalance in the searching phase will affect the overall parallel querying rates.

We tested the load balancing schemes by performing two separate tests that issued 1,000 randomly generated queries, each with the same random seed. The first test randomly restricted the time and spatial dimensions, and the second test randomly restricted the variable ranges. These tests used a ≈100 GB subset of the MODIS dataset of 40 timesteps, and the final results were averaged. The items returned from the individual queries ranged from 0.001% to 20% of the dataset.

The first experiment calculated the overhead times of distributing the data for the load balancing schemes. The Hilbert-Order and the Z-Order schemes required a parallel sort of the dataset, and the parallel sample sorting algorithm discussed in Section 2.7 is used to do this. The timing results are shown in Figure 4. The Hilbert-Order and Z-Order schemes scaled linearly. The Random and Round Robin schemes almost scaled linearly, but it is not expected for them to have scaled perfectly since they are already under one second at 4K cores. The Hilbert-Order scheme incurred the most overhead because of the time involved in computing and sorting the Hilbert indices. The Z-Order indices were easier to compute, thus the scheme had a smaller overhead than the Hilbert-Order scheme. The Round Robin and Random distributions were almost a factor of 10 faster at all scales when compared to the distributions that require sorting.

To further assess if these distribution costs are outweighed by the benefits of faster querying, we computed the average time per query. The results, displayed in Figure 5, showed us

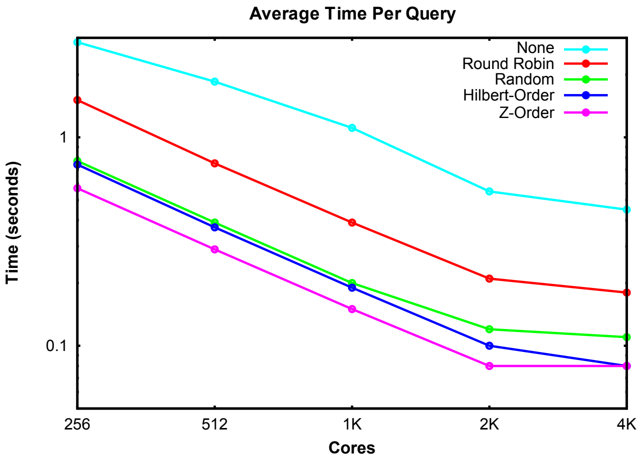

Figure 5: The average time per query for the load balancing schemes. Time is shown on logarithmic scale.

that the Hilbert-Order scheme is never the optimal scheme to use, because the overhead time was always the slowest and the resulting querying times were never the fastest at any scale. In all of the cases, the Z-Order distribution gave the fastest query times, and the Random distribution followed closely.

From these results, we concluded that the Random scheme is the best to use for general applications that will not be issuing queries on the order of the thousands. The Z-Order scheme showed the fastest querying rates, but the cost associated with the distribution will only be outweighed by the querying times when very many queries are issued. Even though the Round Robin scheme showed a very small overhead, the resulting time spent querying was costly as the number of queries increased. We believe this is because the Round Robin scheme is highly dependent on the layout of data in memory after the I/O step. As a comparison, applying no load balancing scheme to the dataset resulted in noticeably poorer querying rates.

Although randomness is not guaranteed to preserve spatial locality as mentioned in [5], the costs of distributing the data along with the resulting performance outweighed the Hilbert-Order scheme that was used in [5]. We chose to use the Random scheme for our analysis in Section 5 because of the significantly small overhead and fast querying rates.

5. MULTIVARIATE CLIMATIC TRENDS

We used our system to discover climatic trends in two variables from the MODIS dataset. Our approach has specific motivations in climate science, but can be applied to a broad variety of application areas. The two problems we addressed in climate science are drought assessment and time-lag analysis.

5.1 Variables

The two variables we used are computed from the satellite bands of the MODIS dataset. The first variable is the Normalize Difference Vegetation Index (NDVI). This variable is computed with the red band (RED) and the near infrared band (NIR) with the following equation:

$$NDVI = \frac{RED - NIR}{RED + NIR} \tag{1}$$

NDVI measures the changes in chlorophyll content by the absorption of the visible red band and in spongy mesophyll by the reflected near infrared band. Higher NDVI values usually represent the greenness of the vegetation canopy [29].

The other variable we computed is the Normalized Difference Water Index (NDWI). This variable is computed using the red band (RED) and the short wave infrared band (SWIR) with the following equation:

$$NDWI = \frac{RED - SWIR}{RED + SWIR} \tag{2}$$

NDWI is a more recent satellite derived index that reflects changes in water content by the short wave infrared band [30].

5.2 Drought Assessment

Drought is one of the most complicated and least understood of all natural hazards, and much research has gone into using satellite derived data as a means for drought assessment. NDVI and NDWI together have been used to monitor drought in several different studies [29, 31, 32]. In particular, NDVI and NDWI were combined in [29] to form a Normalized Difference Drought Index (NDDI) variable. NDDI is computed as:

$$NDDI = \frac{NDVI - NDWI}{NDVI + NDWI} \tag{3}$$

It was shown in [29] that NDDI is a more sensitive indicator of drought, especially during the summer months in the Central United States. In [29], they used NDVI and NDWI values to find drought in the Central Great Plains of the United States and then used the NDDI value to assess the severity of the drought. We used a similar approach and applied this concept to the entire dataset. By doing this, it is possible to determine if NDDI along with NDWI and NDVI is an acceptable measure for drought. It is important to assess these variables to better understand their usefulness in drought monitoring and prediction. Based on these variables, we used five criteria to find periods of drought:

> $NDWI\_thresh$: To be considered as a location where drought is occurring, the location must have NDWI below this threshold.
>
> $NDVI\_thresh$: Similarly, it must have NDVI below this threshold to be considered as a location where drought is occurring.
>
> $NDDI\_range$: If the NDDI value is in this range, it is marked as a location of drought. A higher NDDI range indicates more severe drought.

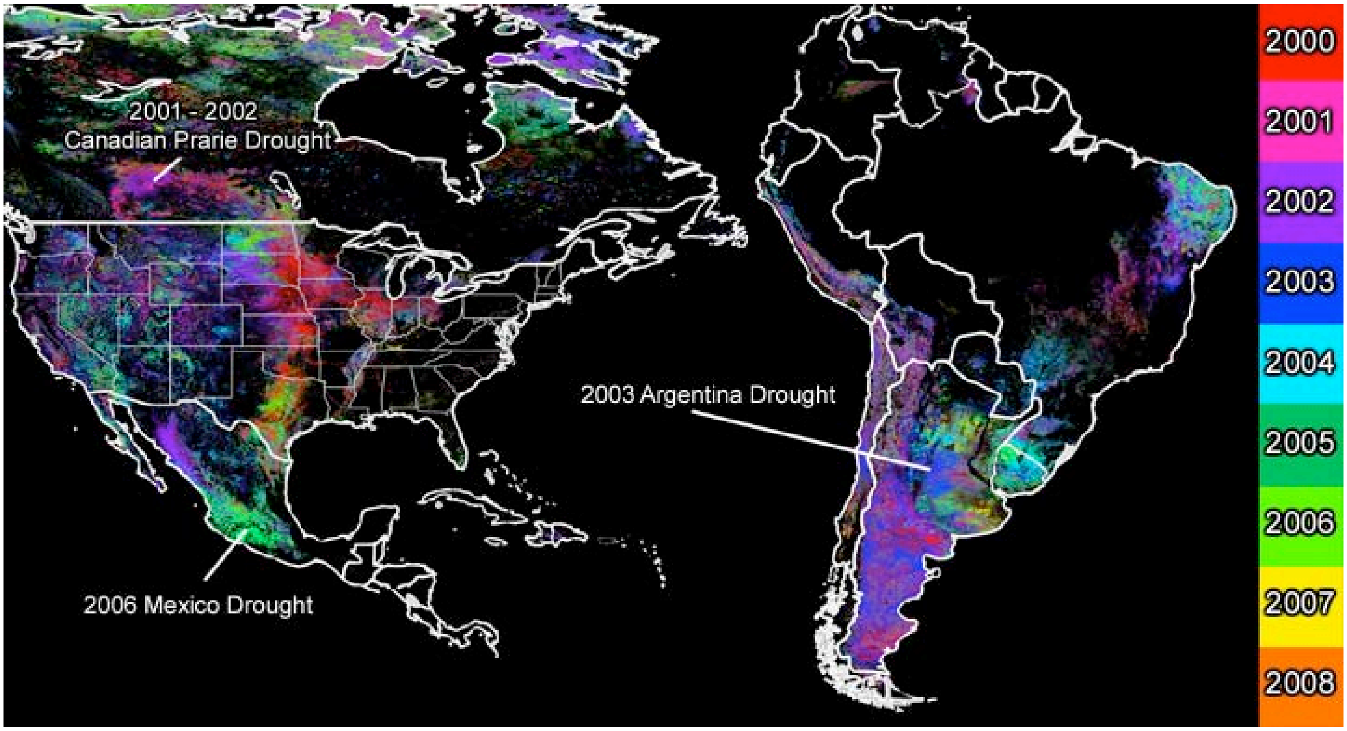

Figure 6: Drought analysis of NDVI < 0.5, NDWI < 0.3, and NDDI between 0.5 and 10.0. These are periods of drought that lasted for at least a month and occurred up to two years for any given region. Several regions are marked where drought has happened, most notably the 2006 Mexico drought. The image was colored based on the year that the longest drought occurred.

Cores	Read	Write	Filter	Distribution	Query	Sort	Analysis	Total Time
1K	85.23	0.68	47.11	22.40	1.01	14.17	0.22	170.82
2K	48.62	0.30	24.49	14.19	0.71	11.31	0.21	99.83
4K	41.27	0.31	11.80	8.78	0.34	9.18	0.10	71.78
8K	48.78	0.32	6.30	5.60	0.16	6.50	0.08	67.74
16K	46.56	0.30	3.45	3.02	0.08	4.20	0.07	57.68

Table 1: Timing results (in seconds) of the drought application.

min_time_span: A location marked as a drought must occur for at least this time span within a year to be considered as an extended period of drought.

max_years: If the location meets all the restrictions above, but happens more than a given number of years, that area is discarded as it is considered normal for it to have the other restrictions.

The first three criteria define the multivariate contour in value space. The fourth criterion defines the expanded temporal dimension for the event. This added dimension causes a much increased amount of complexity. The last criterion is a minor component. Its sole purpose is to filter out regions composed of barren lands, where analyzing drought is not as meaningful.

By using these criteria, we extracted the periods of extended severe or moderate drought. We used our system to query below *NDWI_thresh* and below *NDVI_thresh* on the growing season of the Northern Hemisphere (May - October) and the growing season of the Southern Hemisphere (November - April). After accumulating the queried data over the growing seasons of all the years, a parallel sort was performed in spatial and temporal ordering. We then calculated the time span and number of years that the restriction on *NDDI_range* occurred for each spatial point by stepping through the data on each process. If *min_time_span* and *max_years* was met, the point was colored based on the year that the longest drought occurred. The final image was written in parallel.

We tested our analysis by iteratively stepping through various thresholds of NDVI and NDWI and various ranges of NDDI. Figure 6 shows the resulting visualization from setting *NDVI_thresh* = 0.5, *NDWI_thresh* = 0.3, *NDDI_range* = 0.5 – 10.0, *min_time_span* = 0.3, and *max_years* = 2. In [29], the same values for NDWI and NDVI were used to assess drought. The results show various regions where drought was a real-world problem, most notably an abnormal drought in Mexico during 2006. This is one of the examples of how using analysis like this could help find acceptable parameters to use in drought monitoring tools.

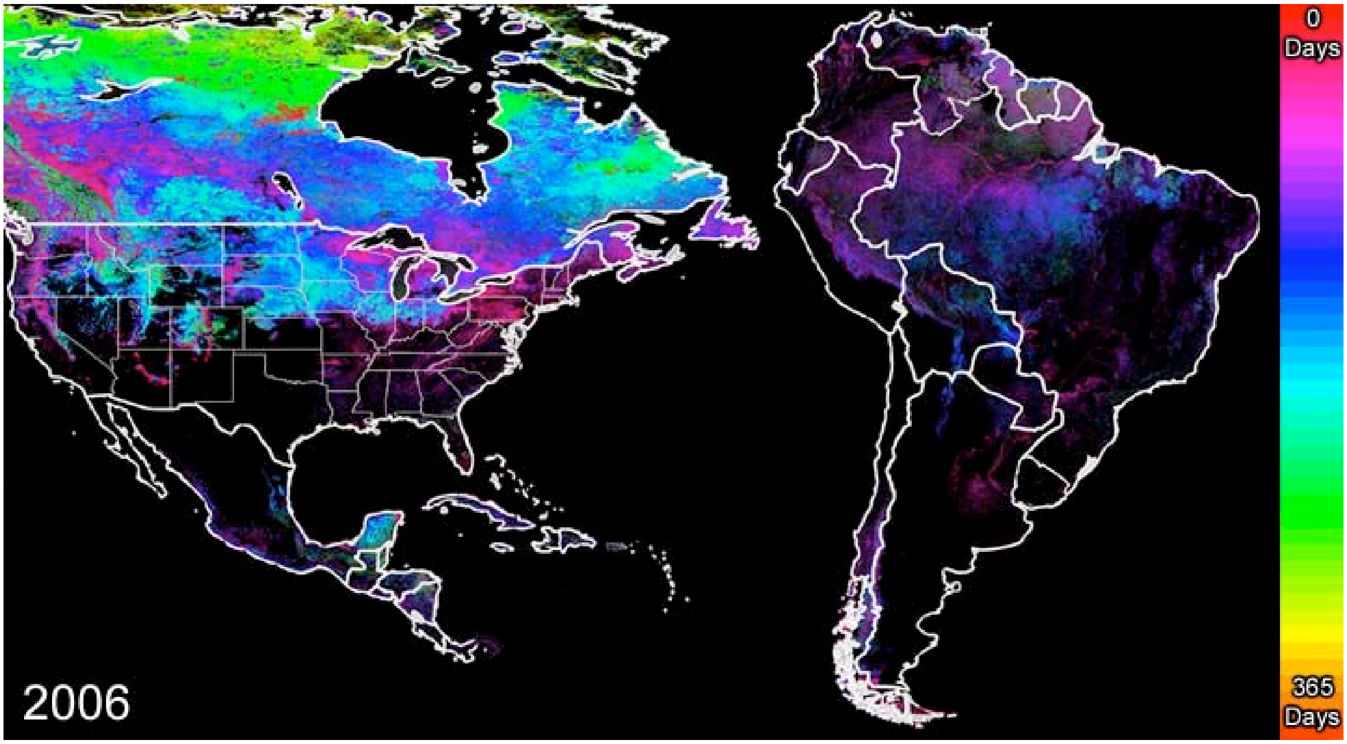

Figure 8: Time-lag between the first occurrence of 0.7 < NDWI < 0.9 and the first occurrence of 0.4 < NDVI < 0.6 in 2006. The color represents the length of the time-lag. Calculating this allowed us to assess the duration from the first snow to the beginning of vegetation green-up.

Cores	Read	Write	Filter	Distribution	Query	Sort	Analysis	Total Time
1K	80.63	1.96	47.90	24.20	6.11	26.21	0.25	187.26
2K	51.54	2.03	22.22	18.90	3.01	20.25	0.15	118.10
4K	45.48	2.05	12.60	12.56	1.64	15.30	0.08	89.71
8K	47.43	2.04	7.00	7.14	0.90	8.90	0.05	73.46
16K	46.34	2.06	3.65	5.02	0.51	4.80	0.05	62.43

Table 2: Timing results (in seconds) of the time-lag application.

5.3 Time-Lag Analysis

Along with drought assessment, another widely studied problem in climate science is the time-lag among variable changes. Studying time-lag is important for obtaining better understandings of how variables like NDVI are affected by other conditions. In particular to the drought assessment problem, studying past and present droughts in relation to these conditions could enhance the capability to monitor vegetation and develop better early warning systems [33].

We defined the problem of time-lag in the following manner: calculate the time between the first or last occurrence of variable v_0 in range r_0 and the first or last occurrence of variable v_1 in range r_1. We specifically focused on the problem of time-lag between the first snow fall and the first sign of green-up from vegetation. Figure 7 shows averaged NDVI and NDWI variables from a region in Saskatchewan Canada to illustrate our problem. NDWI shows abnormally high values when snowfall occurred, and NDVI shows abnormally low values. By calculating the time between when NDWI reached these abnormally high values and when NDVI first broke out of the low values, the time between first snow fall and beginning of vegetation green-up can be computed.

To solve this, we first queried on ranges r_0 and r_1 for v_0 and v_1 for each year. After querying, the appropriate data was sorted in parallel in spatial and temporal ordering. Each process then computed the time-lag between the first occurrence of v_0 and the first occurrence of v_1 in their respective ranges. The resulting image was then colored based on the time-lag and written in parallel.

Figure 8 shows the resulting visualization for $0.7 < \text{NDWI} < 0.9$ and $0.4 < \text{NDVI} < 0.6$. The time between the first snow fall and vegetation green-up for various regions in Northern Canada was almost an entire year. Smaller time-lags around the Central and Southeastern United States were found, normally ranging from about one to two months between the first snow fall and the beginning of vegetation green-up.

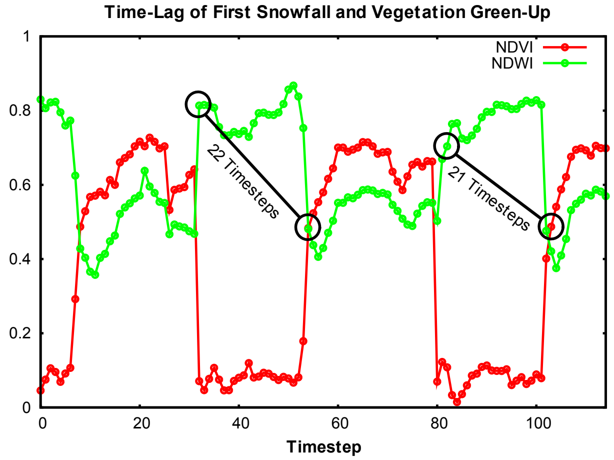

Figure 7: Averaged NDVI and NDWI variables from a region in Saskatchewan Canada. Circled points show the first occurrence of NDWI in the 0.7 − 0.9 range and NDVI in the 0.4 − 0.6 range.

5.4 Application Timing Results

Timing results were gathered from the two separate applications. Detailed results of the drought application are shown in Table 1, and results from the time-lag application are shown in Table 2.

In the drought application, queries were issued on the growing seasons of the Northern and Southern Hemispheres for all growing seasons, resulting in 18 total queries. One parallel sort was issued before analysis, and one final image was written. In Table 1, the aggregate time of all 18 queries is shown, along with timing for the other aspects of the application. With the parameters used to generate Figure 6, ≈11 billion relevant items were returned from the queries and sorted before analysis. At 16K cores, the application aggregately queried ≈137.5 billion items per second and sorted ≈2.6 billion items per second.

Similarly for the time-lag application, queries were issued for each year on the Northern and Southern Hemispheres with the beginning of each year starting in the Fall. For each year and hemisphere, a query was issued on the appropriate range of NDVI and another was issued on the appropriate range of NDWI to calculate time-lag. Thus, four queries were issued each year. Since we were interested in time-lag on a yearly basis, we gathered the results separately for each year, resulting in 10 sorts being performed before analysis and 10 files being written.

The results in Table 2 show the aggregate times for the querying, sorting, and writing, along with the times for the other parts of the application. With the parameters used to generate Figure 8, ≈1.2 billion relevant items were returned, sorted, and analyzed for each year. Since we did this on all 10 years, ≈12 billion relevant items were returned, sorted, and analyzed for the total application. At 16K cores, the application aggregately queried ≈23.5 billion items per second and sorted ≈2.5 billion items per second.

Similar scaling results were observed in the applications. When scaling to 4K cores, I/O bandwidth reached ≈25 GB/s but then tapered off. This is because it was shown in [27] that using too many processes to perform I/O will often cause bandwidth degradation. It is beyond the scope of our work right now to address this issue in our reading phase, however, we did gather the final image to a smaller amount of processes in the writing phase. This is why the writing time remained near constant as process counts were scaled.

The aspects of the applications other than I/O showed scalability to 16K cores. The time for filtering the useless points in the dataset turned out to be a significant portion of the application, and more work will be needed to enable faster filtering of useless values. The analysis times were almost negligible. When scaling to 16K cores, both applications showed an end-to-end execution time of nearly one minute.

6. CONCLUSIONS AND FUTURE WORK

In this work we have shown the feasibility of performing sophisticated, full-range analysis of terascale datasets directly from application-native netCDF data. The end-to-end time for these analyses can be optimized to about a minute, which is acceptable to many scientific users. We have also systematically evaluated a few common design alternatives in parallel reading of data as well as data distribution schemes for load balancing. We found a greedily-driven assignment of file reads to be a practical trade-off between complexity of I/O methods and achievable bandwidth. We have also found that, for terascale datasets handled by thousands of cores, random distribution, while simplistic, actually demonstrated a trade-off advantage in user-perceived performance.

With this system and the small amount of time needed to perform queries, it is our future plan to study problems that will require many more queries to be issued. One example is a sensitivity-based study to determine which parameters are best for performing drought assessment, instead of giving static restrictions to the system. In addition, we also plan on extending our system to handle out-of-core datasets because future growth of data will likely continue to outpace the increases in system memory and processing bandwidth.

7. ACKNOWLEDGMENTS

Funding for this work is primarily through the Institute of Ultra-Scale Visualization (`http://www.ultravis.org`) under the auspices of the SciDAC program within the U.S. Department of Energy (DOE). Important components of the overall system were developed while supported in part by a DOE Early Career PI grant awarded to Jian Huang (No. DE-FG02-04ER25610) and by NSF grants CNS-0437508 and ACI-0329323. The MODIS dataset was provided by NASA (`http://modis.gsfc.nasa.gov`). This research used resources of the National Center for Computational Science (NCCS) at Oak Ridge National Laboratory (ORNL), which is managed by UT-Battelle, LLC, for DOE under Contract No. DE-AC05-00OR22725.

8. REFERENCES

[1] T. Peterka, H. Yu, R. Ross, and K.-L. Ma, “Parallel volume rendering on the IBM Blue Gene/P,” in *EGPGV ‘08: Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, April 2008, pp. 73–80.

[2] “MODIS,” http://modis.gsfc.nasa.gov.

[3] D. Silver and X. Wang, “Tracking and visualizing turbulent 3d features,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 3, no. 2, pp. 129–141, 1997.

[4] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” in *Proceedings of ACM SIGGRAPH*, 1987, pp. 163–169.

[5] M. Glatter, C. Mollenhour, J. Huang, and J. Gao, “Scalable data servers for large multivariate volume visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1291–1298, 2006.

[6] L. Gosink, J. C. Anderson, E. W. Bethel, and K. I. Joy, “Query-driven visualization of time-varying adaptive mesh refinement data,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, 2008.

[7] K. Stockinger, J. Shalf, K. Wu, and E. Bethel, “Query-driven visualization of large data sets,” in *VIS ‘05: Proceedings of the IEEE Visualization Conference*, October 2005, pp. 167–174.

[8] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova, “In situ processing and visualization for ultrascale simulations,” *Journal of Physics*, vol. 78, June 2007, (Proceedings of the SciDAC 2007 Conference).

[9] “Lustre,” http://www.lustre.org.

[10] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir, *MPI-The Complete Reference: Volume 2 - The MPI Extensions*. Cambridge, MA, USA: MIT Press, 1998.

[11] R. Thakur, W. Gropp, and E. Lusk, “Data sieving and collective I/O in ROMIO,” in *Proceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation*, 1999, pp. 182–189.

[12] K.-L. Ma, A. Stompel, J. Bielak, O. Ghattas, and E. J. Kim, “Visualizing large-scale earthquake simulations,” in *SC ‘03: Proceedings of the ACM/IEEE Supercomputing Conference*, 2003.

[13] H. Yu, K.-L. Ma, and J. Welling, “A parallel visualization pipeline for terascale earthquake simulations,” in *SC ‘04: Proceedings of the ACM/IEEE Supercomputing Conference*, November 2004.

[14] H. Yu, K.-L. Ma, and J. Welling, “I/O strategies for parallel rendering of large time-varying volume data,” in *EGPGV ‘04: Proceedings of the Eurographics Symposium on Parallel Graphics and Visualization*, June 2004, pp. 31–40.

[15] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, “Adaptable, metadata rich IO methods for portable high performance IO,” in *IPDPS ‘09: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, 2009.

[16] M. Glatter, J. Huang, S. Ahern, J. Daniel, and A. Lu, “Visualizing temporal patterns in large multivariate data using textual pattern matching,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1467–1474, 2008.

[17] L. Gosink, J. Anderson, W. Bethel, and K. Joy, “Variable interactions in query-driven visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1400–1407, 2007.

[18] K. Stockinger, E. W. Bethel, S. Campbell, E. Dart, and K. Wu, “Detecting distributed scans using high-performance query-driven visualization,” in *SC ‘06: Proceedings of the ACM/IEEE Supercomputing Conference*, October 2006.

[19] O. Rübel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. Weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel, “High performance multivariate visual data exploration for extremely large data,” in *SC ‘08: Proceedings of the ACM/IEEE Supercomputing Conference*, November 2008.

[20] G. E. Blelloch, C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. Smith, and M. Zagha, “An experimental analysis of parallel sorting algorithms,” *Theory of Computing Systems*, vol. 31, no. 2, pp. 135–167, 1998.

[21] M. Meissner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis, “A practical evaluation of the four most popular volume rendering algorithms,” in *Proceedings of the IEEE/ACM Symposium on Volume Visualization*, October 2000.

[22] F.-Y. Tzeng and K.-L. Ma, “Intelligent feature extraction and tracking for visualizing large-scale 4d flow simulations,” in *SC ‘05: Proceedings of the ACM/IEEE Supercomputing Conference*, November 2005.

[23] R. Ross, T. Peterka, H.-W. Shen, K.-L. Ma, H. Yu, and K. Moreland, “Visualization and parallel I/O at extreme scale,” *Journal of Physics*, vol. 125, July 2008, (Proceedings of the SciDAC 2008 Conference).

[24] R. Rew and G. Davis, “NetCDF: An interface for scientific data access,” *IEEE Computer Graphics and Applications*, vol. 10, no. 4, pp. 76–82, 1990.

[25] J. Li, W. Liao, A. Choudhary, R. Ross, R. Thakur, W. Gropp, R. Latham, A. Siegel, B. Gallagher, and M. Zingale, “Parallel netCDF: A high-performance scientific I/O interface,” in *SC ‘03: Proceedings of the ACM/IEEE Supercomputing Conference*, 2003.

[26] G. Memik, M. T. Kandemir, W.-K. Liao, and A. Choudhary, “Multicollective I/O: A technique for exploiting inter-file access patterns,” *ACM Transactions on Storage*, vol. 2, no. 3, pp. 349–369, 2006.

[27] W. Yu, J. S. Vetter, and S. Oral, “Performance characterization and optimization of parallel I/O on the Cray XT,” in *IPDPS ‘08: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–11.

[28] “IOR,” http://www.cs.sandia.gov/Scalable_IO/ior.html.

[29] Y. Gu, J. F. Brown, J. P. Verdin, and B. Wardlow, “A five-year analysis of MODIS NDVI and NDWI for grassland drought assessment over the Central Great Plains of the United States,” *Geophysical Research Letters*, vol. 34, 2007.

[30] B. Gao, “NDWI – A normalized difference water index for remote sensing of vegetation liquid water from space,” *Remote Sensing of Environment*, vol. 58, no. 3, pp. 257–266, December 1996.

[31] L. Wang, J. Qu, and X. Xiong, “A seven year analysis of water related indices for Georgia drought assessment over the 2007 wildfire regions,” *IEEE International Geoscience and Remote Sensing Symposium*, 2008.

[32] C. Liu and J. Wu, “Crop drought monitoring using MODIS NDDI over mid-territory of China,” *IEEE International Geoscience and Remote Sensing Symposium*, 2008.

[33] T. Tadessee, B. D. Wardlow, and J. H. Ryu, “Identifying time-lag relationships between vegetation condition and climate to produce vegetation outlook maps and monitor drought,” *22nd Conference on Hydrology*, 2008.