

Expressive feature characterization for ultrascale data visualization

W Kendall,¹ M Glatter,¹ J Huang,¹ T Peterka,² R Latham,² and R B Ross²

¹Department of Electrical Engineering and Computer Science, The University of Tennessee at Knoxville, Knoxville, TN 37996, USA

²Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA

E-mail: kendall@eecs.utk.edu

Abstract. The grand challenge for visualization is to cast information into insightful content so scientists can test hypotheses and find phenomena not possible otherwise. This challenge is faced with a critical gap in the scientists' abilities to succinctly characterize phenomena of interest in their datasets. Furthermore, as applications generate larger and more complex datasets, efficiently retrieving the features of interest becomes a significant problem. In this paper we discuss recent achievements and new capabilities in the characterization and extraction of qualitative features. We also outline the necessary components needed by a backend system to use this new capability and operate in the same environment as the scientific applications. Our methods have allowed us to scale to high process counts on leadership computing resources and have also allowed us to keep pace with the growing size of scientific datasets. We discuss our efforts of examining qualitative events in cutting-edge climate data.

1. Introduction

Visualization is the primary tool for scientists among a wide array of domains to gain insight into complex phenomena, whether it is the core collapse of a supernova or major ocean eddies and currents. One of the primary components that is tied in directly with the visualization process is the ability describe these types of features in a way that is understandable to the human and the computer. Furthermore, the retrieval of the features plays a crucial role in the interactivity of visual analysis. With dataset sizes increasing to hundreds of terabytes and beyond, feature retrieval becomes a first-class problem along with characterization.

Our recent research on feature specification methods has led to new capabilities to concisely define and visualize qualitative user interests, for example, the beginning of spring or the extinction regions through time in fuel combustion. In one novel approach, we used a regular expression language for finding temporal trends in datasets [3]. In another, we used statistical distributions of variables in neighborhoods for visualizing local characteristics of volumetric datasets [5]. These methods, along with many other works, have one main and simple generic requirement – the issuing of Boolean range queries. This simple requirement allows for the building of powerful systems on top of the methods.

Besides new functionalities for expressive feature characterization, we also focused on scalability in anticipation of extreme-scale challenges. Our prototype system, outlined in

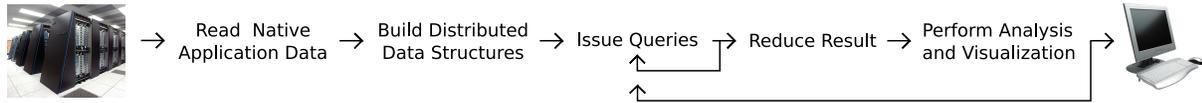


Figure 1: An overview of our end-to-end system. Data stored from a simulation is first read using advanced I/O techniques. The necessary data structures are then built for load-balanced querying to take place. After this, the user can issue queries using a higher-level language and aggregate the results. A reduction can be applied to the results before analysis and visualization.

Figure 1, successfully extracted climatic events from over a terabyte of observational satellite data in less than one minute using 16 K processes; this time included parallel I/O, parallel on-the-fly processing, and parallel visualization [6]. Harnessing this type of computing power offers a virtually unlimited ability to search for and extract features of interest. Scalability such as this is also needed for visualization to keep pace with peta- and future exascale supercomputers. In the following, we provide a holistic overview of the various components that are required by such a system, details of our methods for expressive feature characterization, and sample results from our techniques when applied to terascale observational climate data.

2. System Components

To be successful on the user and application level, the design of our end-to-end system is driven with usability and scalability in mind. We also keep the issuing of range queries as a black box for encapsulation by other higher-level tools. Figure 1 overviews the system in terms of three major components: parallel I/O, distributed data structures, and analysis.

Parallel I/O The I/O component is the most challenging part of the system. The scalability of I/O systems is hampered by many factors, and even seemingly advanced I/O techniques might not be effective for certain problems. For usability, the system requires handling data formats familiar to the scientist, which makes the problem even more difficult.

Working with data directly from simulations, i.e. “native application” form, requires performing I/O on higher-level file formats like netCDF and HDF stored across multiple time-varying files. Our system elegantly handles data in this format by first assigning files to processes in a way to maximize contiguous I/O requests and then using the Block I/O Layer (BIL) to read data across multiple files and variables. Significant I/O bandwidth increases can be obtained when using multicollective I/O techniques [7] for reading multiple files and variables at once, especially when smaller files would not individually warrant the use of high parallelism.

This method allowed us to achieve up to 28 GB/s bandwidth performance on the Cray XT4 Jaguar machine at Oak Ridge National Laboratory, which is 75% of reported IOR benchmark performance tests [2]. Performance such as this in an application setting is critical in reducing the latency between the end of simulations and analysis and visualization.

Distributed Data Structures In our studies [4], we have found that using a structure similar to a B-tree for query processing adheres closest to our system design goals. The ability to search arbitrary amounts of data with unlimited cardinality provides the required usability, and the potential to easily partition the tree in an embarrassingly parallel fashion supplies the needed scalability. The storage overhead of our tree structure is also minimal and usually less than one percent of the dataset size.

Beneath the tree, the data is stored at the granularity of individual items that contain the following queryable elements: scalar or vector values of the voxel they represent, spatiotemporal quantities, and other on-the-fly derived properties. The items are distributed to each process in

such a way to enhance load-balanced query results; we found that randomly distributing items achieved a low distribution overhead while giving ideal load balance. This method has allowed us to achieve significant performance increases at every scale up to 16 K processes [6]. With such scalability, the querying component of our system is effectively unbounded and allows for more sophisticated analysis and querying languages to be built on top of it.

Analysis The analysis component of the system is built on top of a familiar parallel programming paradigm known as MapReduce [1]. In MapReduce, the user maps a value to a key and then the keys are sorted so that a reduction can be applied to the values from each key in parallel. This programming concept is applicable to a wide variety of problems in data mining and document indexing, and it can also be applied to many types of scientific analyses.

When querying has been completed in our system, the relevant items can then be sorted by a key, whether it is the spatial or temporal dimensions or variable values from the query. This allows for more sophisticated types of analyses such as temporal differencing or clustering to be applied. The usability of the system is further extended to more powerful querying languages that can take advantage of ordering of results; one such example from climate science that requires this feature, drought detection, and other examples are outlined in the following section.

3. Feature Characterization and Visualization

Our feature characterization is driven by the fact that the scientists' definition of the features they expect to find may be precise or vague. In both of these scenarios, the complexity of the dataset plays a major role in feature exploration. When simulations produce datasets that have many variables, timesteps, and even many models, the use of a higher-level ability to quickly and comprehensively explore data is necessary. Specifically, the ability to provide "wildcards" allows for exploring a greater degree of uncertainty about the dataset.

Regular Expression Framework A language with the qualities of regular expressions fits in this framework and provides the ability to succinctly characterize features. The language is modeled after the work of [3]. This research showed the effectiveness of using a language to discover temporal events. For example, the query $???[SNOW \leq 0.7]^*T[SNOW > 0.7]^?*$ detects the first large snowfall. The first three question marks indicate that it does not matter what values of snow are present during January, February, and March. $[Snow \leq 0.7]^*$ represents a low snowfall for zero or more months after March, and $T[SNOW > 0.7]$ returns the first month T in which the snowfall reached the 0.7 threshold. The months after this do not matter and are represented with the question mark followed by the asterisk. T values and items that matched the query are returned to visualize the event of first snowfall.

We are currently extending this language and developing components that will allow discovery of more sophisticated features. One such example, $?*[VEG < 0.5 \ \& \ WATER < 0.3]^4?*$, is a query for drought. The caret followed by the four indicates that low water and vegetation must occur for at least four timesteps in a row to be considered a drought. We are also exploring another extension to the language that will permit querying of neighborhood and cluster-based features. Describing features in this way showed a wide variety of uses in [5] and it would be useful to interactively perform these queries on large datasets.

Case Study - MODIS We applied our backend system to over a terabyte of observational Moderate Resolution Imaging Spectroradiometer (MODIS) data. The dataset contains vegetation and water indices over a 500-meter resolution sampling of North and South America (31,200 x 21,600 grid) spanning 417 timesteps over 9 years; each timestep is stored in a separate

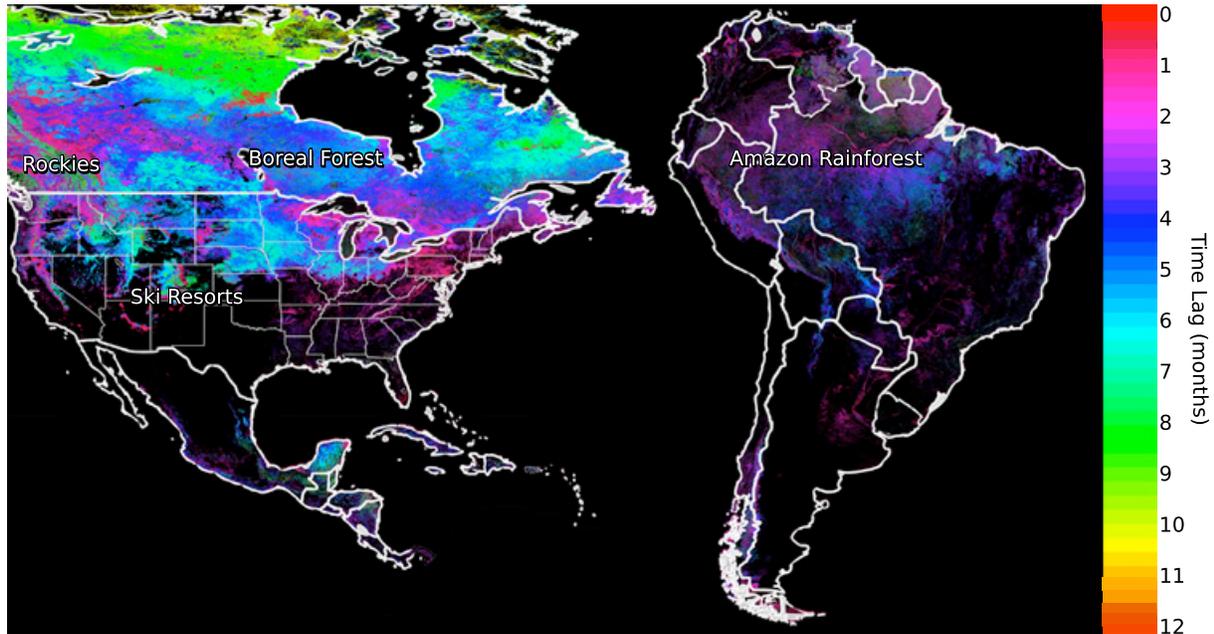


Figure 2: Overview of the length of snow season. This visualization was created by querying the time lag between the first occurrences of abnormally high water content (an indicator for snow) and vegetation green up. Some notable areas are marked such as the ski resorts in Colorado, which have long snow seasons. Other areas like the Amazon Rainforest are returned because of the naturally high amount of water content, our original indicator for snow.

file. We used the Jaguar XT4 machine at Oak Ridge National Laboratory which consists of 7,832 quad-core 2.1 GHz AMD Opteron processors and a Lustre parallel file system.

To obtain a better understanding of the length of a snow season in MODIS, we issued the query represented by $0.7 \leq WATER \leq 0.9$ and $0.4 \leq VEG \leq 0.6$. The returned T variables are the first occurrences of high water content (a likely indicator of snow) and vegetation green up; the difference of these provides an approximated length of the snow season. A visualization from the year 2006 is shown in Figure 2. Some obvious structures can be seen, such as an abnormally long snow season in the ski resorts of Colorado and the Northern Rocky Mountains in Canada. We can also observe distinct contours in snow season length when entering the Boreal Forest in Canada. Some anomalies like the Amazon Rainforest are returned because of the abnormally high water content in the area, our original definition for snow.

Application timing is shown in Figure 3. The reading time includes the entire time it took to query the netCDF data for variable information, assign I/O to processes, and then read the data in memory. The processing time includes filtering out useless content (e.g. ocean values), redistributing the data for load-balanced querying, and locally sorting the data. Querying, reducing, and writing the images are shown as an aggregate timing result, and the total application time is also shown.

The application executed in slightly over one minute at the largest scale, a seemingly insignificant amount of time in comparison to the amount of data being processed. I/O was the dominant component of the application, taking 46 seconds at 16 K processes. We reached our peak reading rates at 2 K processes, and I/O performance degraded slightly when scaling higher; this phenomena has also been observed in other scenarios when using high process counts for I/O on Jaguar [2]. The other components of the system scaled well at high process counts. Processing the entire dataset took as little as

8 seconds. All 18 queries (2 hemispheres for each of the 9 years) were completed in 0.51 seconds and returned 11 billion relevant items. At 16 K processes, all items were reduced in 4.8 seconds for temporal analysis and it took 2 seconds overall to write the 9 images for each year.

4. Summary

The ability to describe complex events using a mini programming language is powerful. High-level languages enable intuition-led discovery of conceptual features that may involve a multitude of uncertainties. Supported by a scalable backend, our approach provides a powerful means of application science research, and also a promise of continued scalability at extreme scales.

We learned crucial lessons in the designing of an ultrascale visualization system that closely integrates sophisticated I/O together with full-range processing and analysis of data in native application formats. We have embedded our know-how into two open source packages: SQI – our backend querying system, and BIL – our multi-file parallel I/O library for visualization and analysis. Downloads are freely available at <http://seelab.eecs.utk.edu>.

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). System development was 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. MODIS data was provided by NASA (<http://modis.gsfc.nasa.gov>). Resources were used from the National Center for Computational Science at Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC, for DOE under Contract No. DE-AC05-00OR22725.

References

- [1] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI '04: 6th Symp. on Operating System Design and Implementation*, 2004.
- [2] M. R. Fahey, J. M. Larkin, and J. Adams. I/O performance on a massively parallel Cray XT3/XT4. In *IEEE Intl Symp. on Parallel and Distributed Processing*, pages 1–12, 2008.
- [3] M. Glatter, J. Huang, S. Ahern, J. Daniel, and A. Lu. Visualizing temporal patterns in large multivariate data using textual pattern matching. *IEEE Trans. on Visualization & Computer Graphics*, 14(6):1467–1474, 2008.
- [4] M. Glatter, C. Mollenhour, J. Huang, and J. Gao. Scalable data servers for large multivariate volume visualization. *IEEE Trans. on Visualization & Computer Graphics*, 12(5):1291–1299, 2006.
- [5] C. R. Johnson and J. Huang. Distribution driven visualization of volume data. *IEEE Trans. on Visualization & Computer Graphics*, 15(5):734–746, 2009.
- [6] W. Kendall, M. Glatter, J. Huang, T. Peterka, R. Latham, and R. Ross. Terascale data organization for discovering multivariate climatic trends. In *Proc. of ACM/IEEE Supercomputing*, 2009.
- [7] G. Memik, M. T. Kandemir, W.-K. Liao, and A. Choudhary. Multicollective I/O: A technique for exploiting inter-file access patterns. *ACM Trans. on Storage*, 2(3):349–369, 2006.

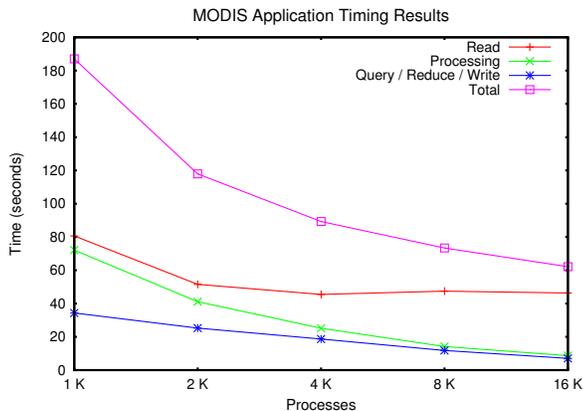


Figure 3: Timing results for the MODIS application. The results include component times for reading and processing data, along with an aggregate timing for querying, reducing, and writing results. The total application time is shown as a sum of these components.