LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-201058

Simulation Data as Data Streams

Ghaleb Abdulla, William Arrighi, Terence Critchlow

November 19, 2003

SIGMOD RECORD

Simulation Data as Data Streams

Ghaleb Abdulla, Terence Critchlow, William Arrighi
Lawrence Livermore National Laboratory
abdulla1@llnl.gov, critchlow@llnl.gov, wjarrighi@llnl.gov

Abstract

Computational or scientific simulations are increasingly being applied to solve a variety of scientific problems. Domains such as astrophysics, engineering, chemistry, biology, and environmental studies are benefiting from this important capability. Simulations, however, produce enormous amounts of data that need to be analyzed and understood. In this overview paper, we describe scientific simulation data, its characteristics, and the way scientists generate and use the data. We then compare and contrast simulation data to data streams. Finally, we describe our approach to analyzing simulation data, present the AQSim (Ad-hoc Queries for Simulation data) system, and discuss some of the challenges that result from handling this kind of data.

1- Introduction

Scientific simulation is used in a variety of scientific and engineering disciplines. In general, a simulation performs an experiment computationally rather than physically. In scientific fields such as physics, chemistry and climatology there is substantial interest in the development and verification of simulations that accurately model the associated physical phenomena. In some cases, the phenomena of interest may not be experimentally controlled. In other cases, experimentation maybe prohibitively expensive, dangerous or physically impossible. These models then play a critical role in understanding and predicting the behavior of complex systems or experiments, and in scientific discovery in general. Examples of simulation experiments include, but are not limited to:

- Climate simulations which are capable of predicting both large-scale global and small-scale local weather patterns.
- Computational biology simulations that predict the interaction of new drugs with receptor sites.
- Astrophysical simulations that describe a phenomena ranging from the formation and evolution of single stars, to galaxies and ensembles of galaxies.
- Nuclear weapons simulations that ensure the safety and efficacy of the nation's weapons stockpile without testing.

These examples demonstrate the spatial and temporal ranges over which scientific simulations may be applied. Time scales can range from femtoseconds to decades and distances can range from microscopic to galactic [1]. Running a simulation can require thousands of processors and produce petabytes of data. Such massive computations have become feasible due to the current generation of massively parallel computers and their associated file systems.

This ability to generate complex and vast amounts of data, however, will overwhelm current computational infrastructure. Effective storage and archival algorithms must be devised to cope with these volumes of data while ensuring the data is retained for both future reference and validation of new results. Querying the simulation results, either while they are being generated or after they have been written to the file system, is an enormously expensive task. Imagine trying to locate a single piece of information in several hundred data sets where each data set is on the order of petabytes.

One possible solution is to summarize such data sets and save a compressed yet representative summary. Another interesting approach is to treat the data set as a data stream, processing the query in real-time as the data is being generated. While not all queries can be easily executed against these data streams, those that can are able to scale to the large data sets that the simulation produces

In this paper we discuss the challenges of querying simulation data, our approach to handling them, and how this approach is similar to the techniques used by the data streaming community.

2- Scientific Simulation Data

The nature of scientific simulation data is strongly dependent on the numerical technique employed by the simulation. Numerous formats

are used to describe and store simulation data. One such format developed and widely used by Lawrence Livermore National Lab is called *Silo*. For a complete description of the *Silo* abstraction, the reader is advised to look at the web address http://www.llnl.gov/bdiv/meshtv/manuals.html

Files in the *Silo* format may be viewed in 3-D using a viewer called Mesh-TV. For practical reasons, we have adopted the Silo data format for our work, although our approach is generally applicable to any mesh format. The images shown in this paper were created using meshTV. For a more detailed description of mesh data see [2].

Types of Meshes

Scientific simulations are discrete approximations to problems defined on continuous media often called a "base space." Obviously, it is impossible to compute and store the values of the variables of interest (e.g. temp, pressure, velocity) at the infinitely many points present in any problem.

Instead, the base space is discretized into a finite number of volumes or "zones." For 3-D data sets, each zone is comprised of a set of 2-D faces, which are themselves made up of an ordered collection of vertices (or points in 3 dimensions). For example, a base space may be discretized into cubes each of which is made up of 6 faces, 12 edges and 8 nodes. The collection of zones used by the simulation to define the 3-D space being modeled is typically called a mesh.

Meshes are generally categorized as either structured or unstructured. Structured meshes are regularly ordered collections of the same element. Quadrilateral meshes are a specific example of structured meshes, in which there are a prescribed number of hexes or quads in each dimension. Unstructured meshes by contrast are collections of arbitrarily shaped zones with complex connectivity relationships. Such meshes require an explicit description, usually in terms of the nodal connectivity of each zone, and are thus usually much more expensive to represent than structured meshes.

Types of Variables

Variables are represented as values at any point in the mesh. The two most common types of variables are "nodal" and "zonal" variables. In the case of nodal variables, the values are associated with each node in the mesh. The variable value at points between the nodes can be found by interpolating the nodal values. These interpolating functions are typically linear although higher order interpolants are sometimes used. In the case of zonal variables, the values are associated with each zone and the variable is typically constant throughout the zone. In some cases, variables have complex values, such as vectors. Figure 1 shows a region of a mesh for an exploding star. The colors (or various degrees of shade) in the figure reflect a variable called *xdot* (the speed in the x direction). The arrows on the figure are the *velocity* vectors

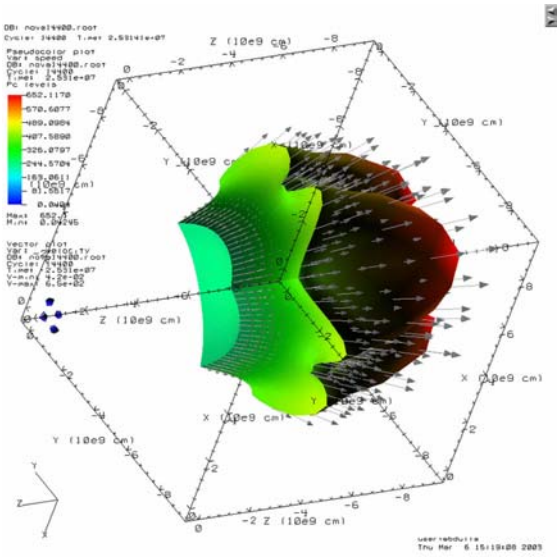

Figure 1: A 3-D plot of an exploding star with the speed and velocity shown on its surface.

To minimize storage requirements it is common to store only the fundamental variables from a simulation. Ancillary results that may be derived from the fundamental variables are not explicitly stored to save space. An example of such a derived variable is the scalar quantity *speed* which is easily derived from the *velocity* vector shown in Figure 1.

This complex structure of the data causes the data size to grow rapidly and makes querying and visualizing the data a challenging task.

3- Data Streams and Scientific Simulation Data

Data streaming is an area of active research in the database community. Data streams differ from conventional stored data in several ways [7]:

1. The data elements in the stream arrive online.

2. The system has no control over the order in which data elements arrive (within and across data streams).
3. Data streams are potentially unbounded in size.
4. Once an element from the data stream is processed, it is discarded or archived and cannot be retrieved easily.

While technically finite in size, simulation data can grow to be arbitrarily large; multi-gigabyte data sets are considered tiny and multi-terabyte data sets are common. While most simulation data sets are archived on tertiary storage after analysis, some are discarded because of storage limitations.

Scientists would benefit from being able to quickly answer some of their queries while data is being written. For example, queries that test for tangled meshes or extreme variable values can be used to determine if the simulation is executing properly or not. By identifying a problem early, scientists can abort erroneous runs and restart the simulation with new initial conditions. This is important because simulation runs are very expensive and they take a long time to finish.

There are two ways we can utilize data streaming algorithms to help scientists:

1. Preprocessing, summarizing, and compressing, the data to query or analyze in the future. To do this effectively requires a one pass algorithm, similar to those used by data streams. The requirements for one pass algorithms is due to the data size.
2. Executing queries on the original data while it is being generated. These queries will be executed in near-real-time, and would be similar in capability to data streams.

In the next section we will discuss the details of using these two approaches

4- Querying Scientific Simulation Data

Simulation data is highly dependant on the application, hence, it is hard to come up with one set of requirements for data processing across applications.

While developing a new simulation code, the scientists must verify both the underlying mathematical model and the implementation. As a result, they spend a lot of time *verifying* the simulation. The algorithm for running the simulation can be summarized as follows

```
While (true)
{   Compute;
    If (remainder (time / N) ==0)
        Write restart file;
        Visualize();
        Modify parameters_if_needed;
        Restart simulation;
    Else
        If (remainder (time / M) == 0))
            Write plot-files;
        endif
    endif
}
```

The simulation writes out *restart files* every N cycles (or simulation timestep). The restart files contain enough detailed information about the simulation status to allow it to be restarted from the current time. *Plot* files are written every M cycles where M < N. To determine the validity of their simulations, scientists visualize the dumped files with a tool such as Mesh-TV. If they encounter an error, they modify the simulation code, the input parameters, or both and rerun the simulation. Given the volumes of data involved, waiting until the entire simulation completes to evaluate it is inefficient. Furthermore, manually examining the images one by one and this is a lengthy and error prone process.

This example shows that real-time analysis of simulation data is needed to help the scientists verify the simulation code. In addition it is needed after the simulation has been verified in order to *steer* the computation by changing the simulation input parameters. This allows the scientists terminate an experiment earlier if the results are deviating from what is expected because input parameters need to be changed and not because of error in the code.

Real-time analysis is very hard and may require coupling the analysis algorithms with the simulation computation. However, there are some success stories for computational steering with quasi real-time visualization. An example of such successful application is the Cactus system where the code was configured to allow remote visualization and steering [12]. It is difficult to apply this approach to our environment for several reasons. First, our data is highly dimensional. Second, our users are interested in relations between variables, not only the values of certain variables. For example, they might want to see what is happening to variable “P” if variable “T” is increasing. Finally, the size of the generated data sets does not allow real-time visualization.

Once the simulation is complete, the scientists are interested in analyzing the data and comparing it with experimental results or with other simulations. This analysis is done on the

simulation results (plot and restart files) after they have been generated. In this mode, the scientists might be interested in more complex queries, such as following a certain phenomena or tracing the values of a variable over the course of time, or comparing it to experimental results.

Ad-hoc Query Simulation System (AQSim)

AQSim is an ongoing effort to design and implement a system to analyze terabyte-sized scientific simulation data sets (Figure 2). The

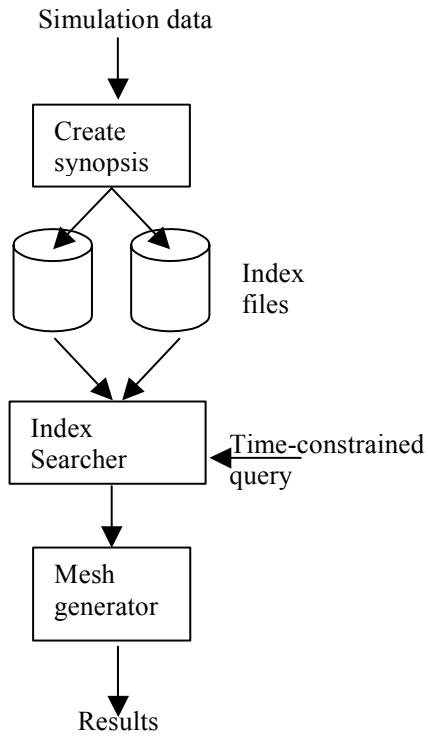

Figure 2: A diagram of AQSIM data flow.

main goal of AQSim is to reduce data storage requirements and access times while permitting ad-hoc queries using statistical and mathematical models of the data

The design of AQSIM is heavily influenced by the following three facts

1- The size of the data generated is huge and scientists are not able to keep it on secondary storage for a long time.
2- Scientists are willing to accept approximate answers when returned in a reasonable time, as long as there are no false negatives.
3- Scientists do not like to sample the data because sampling cannot capture the outliers that are important to them.

To satisfy these requirements we create an approximate, multi-resolution synopsis of the data in a format that maintains the outliers. We use this approximation to compress the data and create efficient indexes for query processing. AQSIM is designed to work off-line after the data has been generated and it requires the synopsis to be generated before the data is queried [3,4,5,10]. The implementation has evolved over the last two years to include a simple statistical model, a wavelet model, and a clustering model. For more details the reader is referred to [4,5]. The latest AQSIM implementation utilizes a bottom-up agglomeration algorithm that uses the topology of the mesh to create the multi-resolution hierarchy. The algorithm starts by collecting the fine cells in the grid into a coarser cell. The algorithm is a two-pass algorithm to guarantee that it will produce the best topology representation. The data within the cells are propagated upwards to the coarser cells using simple statistical information such as number of points, min, max, mean, and standard deviation. The coarse cells use bounding boxes to define their associated spatial regions. Because the finer resolution cells will not generally fill this bounding box, each cell also has an associated percentage filled. After that the resulting tree can be pruned in order to save space. The pruned tree supports processing the queries faster, however, most answers will be approximate.

Users can also limit the time if they are not willing to wait for an answer. In this case, the result will obviously be a lower resolution mesh. A priority is used at query time to decide which cell should be traversed first. This priority is based on a combination of a cell's percentage filled and how well it answers the specific query being asked. This priority based traversal of the multi-resolution tree allows us to generate the best possible mesh for the given time constraint. Currently, we are focusing on improving the efficiency of our out of core algorithms and parallelizing the implementation.

Using data stream terminology, we are processing our data in a batch mode. This choice was made because we cannot control the rate of the data dumps, we have to save the restart files under any circumstances, and the time consuming step of creating the data summary is considered part of the query. In other words, this fits the traditional batch processing mode in which updating the data is faster than query processing.

We are looking into the possibility of extending the system by adding a *direct* query capability that allows querying the data in its

original format. Since this would drop the preprocessing step, computing the answer to the many queries (including writing the data) will be relatively fast. In this case we could run the query quasi real-time on the data. It is important to note, however, that some queries, such as those requiring extensive topology information or a value to be computed globally, cannot be answered using this configuration. For example locating zones where the temperature does not exceed half of the maximum temperature could not be executed because it requires knowing the global maximum of a variable.

Results

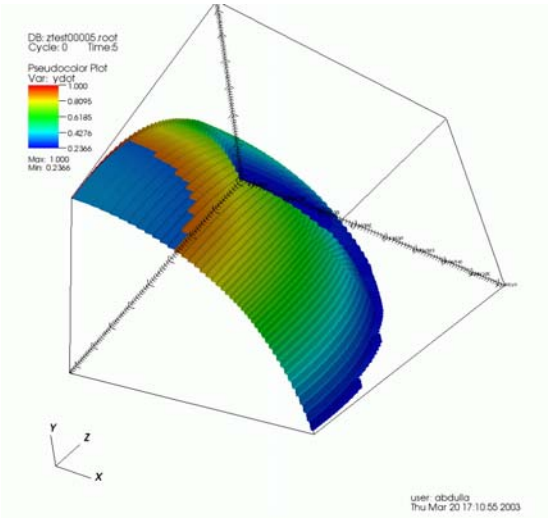

Figure 3: Query: "ydot > 0.4 and ydot < 0.5 and X > 0 and Y > 0 and Z > 0".

We have an early prototype of AQSIM that can be used to query simulation data. We shared our initial results with some scientists and they are excited about it. The system will provide opportunities for understanding the huge amounts of data that they have been collecting. To enable data analysis, we use a simple query language that can support user-defined functions.

Figure 3 shows the results obtained from the first query listed in Table 1 on a 5-GB astrophysics simulation of a star. This data set contains approximately 26 million zones spread across 16 time steps. The resulting mesh, shown in Figure 3, has the highest resolution possible since the query was not time constrained. Although, the query was performed on the data synopsis, the resulting mesh is extremely close to the original one. The query retrieved 38,898 zones in approximately 48 seconds.

Table 1 lists a set of queries, the time it took for each query in seconds, and the number of elements retrieved. These queries were run on a 1.5-GHz Pentium processor Linux workstation with 512 MB of memory and using a single processor. The data was accessed over a 1-Gb

Query	Time	Zones
`ydot > 0.4 and ydot < 0.5 and X > 0 and Y > 0 and Z > 0`	48.86	38898
`ydot > 0 and X > 0 and Y > 0 and Z > 0`	279.43	380035
`ydot > 0 and Y > 0 and Z > 0`	633.18	738265

Table 1: Query time and selectivity (number of zones retrieved).

Ethernet connection. Table 1 shows that when the query retrieves more zones, the time increases but not linearly. This is primarily a result of the query engine moving from in-core to out-of-core data structures and the associated disk accesses. To speed up the queries we are working on creating a more efficient index format, and parallelizing the approximate query engine.

We have also prototyped a parallel implementation of the direct-query capability. We ran the same set of queries on MCR (a linux cluster) distributing the data over 16 processors (one time step per CPU). The direct-query prototype took 38 seconds to execute each query. The time was consistent because the direct-query does not use a multi-resolution hierarchy, instead it linearly scans the entire data set for each query. From the query time perspective, our initial approach will outperform the direct approach when the query selectivity is high enough to require scanning only a small part of the tree.

By enabling the direct-query capability on parallel machines, the scientists can devise queries that allow them to verify the simulation without examining the individual files or performing an expensive preprocessing step. On the other hand, the approximate approach is more appropriate for highly selective queries (which we expect to be the typical case), complex queries such as those requiring topological or global information, and offline analysis since the data set size will be much smaller and manageable.

5- Related Work

The database community has been very active in the area of high-dimensional data and

work on both managing and providing efficient algorithms to help understand it. The literature has many papers that describe data streams and how are they different from the traditional database approach. [7] is an excellent paper that summarizes the research in data stream systems.

[8] discusses the feasibility of building data stream systems for Online Analytical Processing (OLAP). To our knowledge, it is the only paper that discusses data streams applied to scientific data, and we are not aware of any work that looked into simulation data from this perspective.

There are several efforts such as [9], to describe data models to support data generated from physical experiments. The most relevant effort is the sheaf data model, which has evolved from the fiber bundle model that was introduced by Butler and Pendley in 1989 [11].

6- Conclusions

In this paper, we introduced scientific simulation data, the way it is generated, and how it is used. Important similarities exist between simulation data sets and data streams to the point where simulation data can be viewed as a special case of data streams. We used these similarities to show how are we utilizing data streams concepts in building AQSIM. We showed initial results and discussed some of our plans to extend the system. The initial response from our prospective users was very promising.

7- Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405- Eng-48. The authors would like to thank Celeste Matarazzo, Tina Eliassi-Rad, Susan Hazlett, Chuck Baldwin, and Megan Thomas from the Center for Applied Scientific Computing at the Lawrence Livermore National Laboratory for their advice and assistance regarding this research.

References

[1] Louis, S., The NNSA ASCI Program: Advanced Simulation and Computing, presented at the October 9, 2001 THIC Meeting WestCoast Silverdale Hotel, Silverdale WA 98383-9191

[2] Musick, R., and Critchlow, T. Practical lessons in supporting large-scale computational science, In *Proceedings of SIGMOD Record 1999*, ACM Press, 28(4): 49-57.

[3] Abdulla, G., Baldwin, C., Critchlow, T., Kamimura, R., Lozares, I., Musick, R., Tang, N.A., Lee, B., and Snapp, R. Approximate ad-hoc query engine for simulation data, In *Proceedings of JCDL 2001* (Roanoke VA, June 2001), ACM Press, 255-256.

[4] Baldwin, C., Abdulla, G., and Critchlow, T. Multi-Resolution Modeling of Large Scale Scientific Simulation Data, In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, New Orleans, LA, November 3-8 2003.

[5] Eliassi-Rad, T., Critchlow, T., and Abdulla, G., Statistical modeling of large-scale simulation data, In *Proceedings of ACM SIGKDD 2002* (Edmonton Canada, July 2002), ACM Press, 488-494.

[6] Lee, B., Critchlow, T., Abdulla, G., Baldwin, C., Kamimura, R., Musick, R., Snapp, R., and Tang, N.A., The framework for approximate queries on simulation data, *International Journal of Information Sciences*, Elsevier Sciences, forthcoming.

[7] Babcock, B.; Babu, S.; Datar, M.; Motwani, R.; Widom, J. *Models and Issues in Data Stream Systems*, Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002).

[8] Chen, Y.; Dong, G.; Han, J.; Pei, J.; Wah, B. W; Wang, J.; Online Analytical Processing Stream Data: Is It Feasible? 2002 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'2002).

[9] Malon, D.; May, E.; Critical Database Technologies for High Energy Physics, In Proceedings of the 23rd VLDB Conference Athens, Greece, 1997.

[10] Eliassi-Rad, T., Critchlow, T., Multivariate Clustering of Large-Scale Simulation Data, LLNL Technical Report, UCRL-JC-151860, 2003.

[11] Butler, D. M., Pendley, M. H., A Visualization Model based on the Mathematics of Fiber Bundles, Computers in Physiscs, September/October 1989.

[12] Laszewski, G., Insley, J. A., Foster, I, Bresnahan, J., Kesselman, C., Su, M., Thiebaux, M., Rivers, M. L., Wang, S., Tieman, B., McNutly, I., Real-Time Analysis, Visualization, and Steering of Microtomography Experiments at Photon Sources, In *Proceedings of Ninth SIAM Conference on Parallel Processing for Scientific Computing*, (San Antonio, TX, March 1999).