SAND2009-3874C

Exploring Data Warehouse Appliances for Mesh Analysis Applications

Craig Ulmer, Greg Bayer, Yung Ryn Choe, and Diana Roe
Sandia National Laboratories
Livermore, California

Abstract

As scientific computing users migrate to petaflop platforms that promise to generate multi-terabyte datasets, it is becoming increasingly apparent that simulation datasets are too cumbersome to process using traditional, offline techniques. Instead, there is a strong need within the community to be able to embed sophisticated analysis algorithms in the storage systems that house the data for the computing platform. Data Warehouse Appliances (DWAs) are an attractive option for this work because they enable researchers to take advantage of existing data-parallel compute architectures that can process massive, out-of-core datasets efficiently

While DWAs have proven to be effective in data mining and informatics applications, there are relatively few examples of how DWAs can be integrated into the scientific computing environment and utilized to process simulation data effectively. In this paper we present our experiences in adapting two mesh analysis algorithms to function on two different DWA platforms: a SQL-based Netezza database appliance and a Map/Reduce-based Hadoop cluster. While the main contribution of this work is insight into the differences between the two platforms' programming environments, we present performance measurements for entry-level systems to help provide a first-order comparison of the hardware.

1. Introduction

In nearly all scientific domains, researchers employ modeling and simulation tools to help answer complex research questions. In general, scientific users typically model a situation that is of interest and then utilize simulation tools to evaluate the model. Once a simulation completes, visualization and analysis tools are used to help users interrogate the results generated by the simulation in order to gain insight into the details of what took place during the simulation. Depending on model fidelity and computational complexity, simulations running at full scale on high-end computing platforms may take days or weeks to complete and can produce terabytes to petabytes of output data. The shear volume of this data makes it unwieldy to manage and analyze through traditional, offline approaches.

1.1. Mesh-Based Simulations

Sandia National Laboratories has a long history of using simulation tools on high-performance computing (HPC) platforms to solve complex problems relating to national security. In the Advanced Simulation & Computing (ASC) effort [1-2], Sandia has developed many parallel simulation codes for evaluating mechanical, thermal, and electrical properties of complex systems that are subjected to harsh environmental factors. These simulations help determine how well a system will be able to complete its intended function under different operating conditions. Simulation ultimately provides analysts with an opportunity to gain insight without necessarily requiring the expensive testing of actual components.

Many mechanical and thermal simulation tools at Sandia are based on meshed representations of physical objects. For example, finite element method (FEM) [3] codes discretize each physical object in a simulation into a mesh of elements and then apply numerical analysis on each element to determine how the object interacts with other objects as the simulation progresses. Mesh-based simulation datasets are generally comprised of two types of data: structural data and variable data. Structural data provides geometric information about the objects and is organized in a hierarchical form: a simulation contains multiple objects, an object is defined by its mesh, a mesh is defined by its elements, and an element is defined by its vertices. While all elements in a mesh share the same geometric shape (e.g., hexahedron or tetrahedron), Sandia's applications typically employ unstructured meshes with non-uniform elements. The variable data portion of a dataset contains data values that were calculated during the simulation. A variable (e.g., pressure, temperature, or displacement) is associated with either elements or vertices in the mesh. A variable's data is generated at each timestep of the simulation for all elements or vertices. As such, variable data is often much larger than structural data in a dataset.

Table 1: A 100M element simulation can generate many terabytes of data.

Tables		Rows/Bytes
Structural Data	Element	100M 3.2GB
	Vertex	100M – 800M 2.4GB – 19.2GB
Variable Data	Element	20G 2.5TB
	Vertex	20G – 160G 2.5TB – 20TB

High-fidelity simulations can produce datasets that are very large. While typical production runs may be on the order of just a few million elements, leading edge computing platforms have been used to process simulations with more than 100 million elements. As a means of illustrating how quickly datasets grow, Table 1 lists the amount of data that would be hosted in a hypothetical dataset for a 100M element simulation where 16 element and 16 vertex variables are traced for 200 timesteps. While variable data is much larger than structural data, it is important to note that the structural data can be larger than main memory in a single host computer. Many existing data analysis tools cannot process datasets that are this large because the analysis algorithms were written in an in-core fashion that assumes structural data can be housed in host memory. As such, there is a strong need in the scientific community for post-processing analysis tools that can handle massive datasets. At Sandia, ParaView [4] has become the distributed analysis framework of choice. However, other solutions are of interest.

1.2. Capability Computing Platforms

A universal constant in HPC is that scientific users will always need more computing power than available systems offer. When new HPC platforms are brought online, they are strategically designed to improve either the computing capacity of a site or its computing capability. Capacity computing utilizes a new HPC platform's resources to process existing simulations in a shorter amount of time than was previously possible. In contrast, capability computing systems are designed to allow users to increase the fidelity of their simulation models. Capability systems typically push the boundaries of what can be accomplished through modeling and simulation, and are therefore the focus of a considerable amount of research in HPC.

Leading-edge capability systems are custom-built, massively-parallel processing (MPP) systems. In order to increase the amount of computing that can take place in a given footprint, capability system architectures typically separate computing resources from storage resources. For example, Sandia's Red Storm [5] platform employs close to 13,000 diskless compute nodes to perform an application's parallel processing, a separate 1.7PB disk farm for managing persistent storage, and 320 I/O nodes for handling an application's disk requests. Storage systems in HPC systems typically employ a cluster file system such as Lustre [6] to allow data to be striped across many disk nodes and cached in order to hide the high cost of disk access. As such, today's storage systems are often dedicated clusters running a parallel storage application that services load/store requests in a high-performance manner.

1.3. Data Warehouse Appliances

The availability of massive customer databases in industry has resulted in a strong demand by businesses for data mining tools and systems that can allow a company to discover consumer trends in real time. In response to these data mining needs, a number of companies have developed stand-alone products known as data warehouse appliances (DWAs) that enable users to accomplish data analysis operations on parallel platforms without requiring a detailed knowledge of parallel processing techniques. DWAs typically provide (1) a large amount of parallel storage devices that enable high-performance disk use, (2) near-storage processing in order to execute computations where the data resides, and (3) a programming interface that allows users to pose data processing commands. DWAs are very appealing to industry because they can be treated as appliances. In many cases, a company can simply purchase one or more racks of a SQL-based DWA and get genuine speedups over a sequential database without having to make significant modifications to the SQL program.

There are many DWA systems available today. We divide these systems into two categories, based on their programming interfaces. First, SQL-based DWAs are designed to be plug-in replacements for existing database systems. SQL is well known and provides users with a robust data-processing language that fosters data-parallel operations that can in turn be parallelized by a database. Netezza, XtremeData, Oracle, Teradata, and Greenplum all

offer SQL-based DWA products. Second, dataflow-based DWAs are designed to provide a flexible platform for manipulating large-scale datasets when users are more willing to express their data-process commands in other data-parallel languages. Examples of these systems include Hadoop and LexisNexis.

2. DWAs for Scientific Analysis

A number of institutions in the scientific community are currently designing and deploying a new generation of petaflop-class computing platforms that will be capable of processing larger, higher-fidelity models than ever before. While these capability systems will enable researchers to conduct higher-quality science, we make three observations about trends in the HPC landscape that directly impact how scientific users will make use of capability systems in the next decade:

Increased Dataset Sizes: Increases in simulation size and fidelity correspond to increases in the amount of data generated by a simulation. With current capability systems already generating multi-terabyte datasets, we expect that simulations in the near future will generate tens to hundreds of terabytes of data.

Constant Disk and LAN Rates: While fast solid-state storage devices (SSDs) are slowly making their way to market, traditional hard drives are still the dominant storage option for massive datasets. Given that hard drives transfer rates have not improved significantly in a number of years, future storage systems will employ larger disk arrays in order to meeting simulation I/O rates. In a similar manner, link rates for transferring data out of a capability system are not likely to improve substantially in the upcoming years.

Capability Computing Consolidation: While the per-core cost for computing systems has dropped dramatically over the years, the total cost to build, power, and maintain a capability system has not. Under pressure to construct new resources as cost effectively as possible, multiple institutions and laboratories are pooling their efforts to create shared capability platforms. These consolidations emphasize the need for better distance computing practices, where processing and analysis is conducted on equipment that is housed at a location that is far away from the user.

Based on these observations, we assert that the age old model of moving data to the user's analysis code is infeasible and must instead be reversed. In order to accommodate data analysis on massive datasets, the storage systems for capability systems must be enhanced to provide processing within the storage system. Similarly, these new systems must take advantage of data-parallel programming interfaces that shift the burden of parallel programming from the user to the storage system when possible. DWAs represent an attractive option for this work, because they are already widely deployed to solve similar problems in other fields.

While DWAs have been successfully utilized to solve many informatics problems, there have been relatively few efforts that have examined whether they are applicable in scientific problems with massive datasets. The intent of this work is to gain insight into the tradeoffs involved in utilizing a DWA to analyze scientific datasets. For this paper we have adapted two mesh-based analysis operations to function on two different DWAs. In addition to describing each algorithm's implementation, we provide basic performance measurements for both platforms that give insight into system bottlenecks. Finally, we present a number of observations based on our experiences with the DWAs in hopes of motivating future work in this field.

3. Data Warehouse Appliances

For the purposes of this paper we chose to focus on two of the more common DWAs that are in use today: a SQL-based Netezza system and a dataflow-based Hadoop cluster. We acquired an entry-level platform for each, and then adapted our algorithms to run in the two environments.

3.1. Netezza

In 2002 Netezza became one of the first companies to push data warehouse systems as true appliances. The Netezza Performance Server [7] can be described as a custom-everything system. In terms of hardware, Netezza employs a large array of disk blades called snippet-processing units (SPUs). As illustrated in Figure 1, each SPU is comprised of an embedded PowerPC processor, 1 GB of DRAM, an FPGA-based I/O controller, and a SATA hard drive. Overall, the system connects a large number of SPUs through a Gigabit Ethernet (GigE) network. While the custom hardware adds to the overall expense of the system, Netezza can do many unique operations that commodity systems cannot. For example, the I/O controller can be configured to perform data decompression and column

filtering as the data is read off the disk. This capability enables the controller to supply data at rates that are higher than what the disk or SATA link can physically support. Observing that the hard drives are often the bottleneck in I/O intensive applications, this optimization can result in a significant performance gain for the overall system.

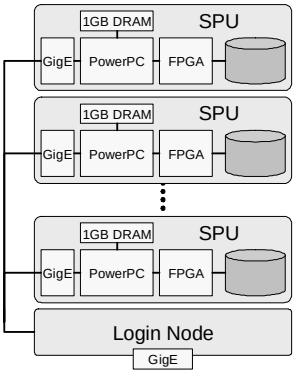

Figure 1: The Netezza system utilizes multiple snippet processing units (SPUs) to process data in parallel.

In terms of software, Netezza employs a parallel database engine that allows the system to stripe data across disk blades and execute a query in parallel. Rather than rely on indexing to help the database locate relevant records, Netezza performs scans on entire tables. While this brute-force approach at first may appear to ignore the fundamentals of modern database design, it results in a scalable system with predictable performance. If a user increases the dataset size or needs better performance, the user simply purchases and uses more nodes.

Similar to other databases, Netezza allows users to supplement the SQL syntax with user-defined operators. These operators are written in C++ and are either user-defined functions (UDFs) or user-defined aggregates (UDAs). UDFs operate on a single row at a time, producing one output value for a given list of column inputs (e.g., compute the squared sum of C column values). UDAs perform the same work as a UDF, but also provide means of collapsing the results for all rows into a single value (e.g., find the second-largest squared sum of C columns in the entire table). In our experience, UDFs and UDAs are a convenient way of expressing data-processing computations that would otherwise be difficult to construct using only the built-in operators provided in SQL. Our strategy has been to construct complex computations in UDFs and UDAs, and then utilize SQL as a means of sequencing the flow of execution required in an algorithm.

3.2. Hadoop Clusters

Hadoop [8] is an open source framework for performing data-parallel operations on commodity cluster hardware. Hadoop was initially constructed by researchers at Yahoo! as an open source Java clone of the Google File System (GFS) [9] and Google's Map/Reduce [10] framework, but has since grown into the basis for a number of different out-of-core data processing projects. At its core, Hadoop is comprised of two components. First, the Hadoop distributed file system (HDFS) [11] provides a scalable, general-purpose file system for distributing data across the local disks in a cluster in an efficient and reliable manner. HDFS operates on large blocks of data (64 MB by default) and is responsible for transferring and synchronizing data between nodes in the cluster as needed by applications. Second, Hadoop provides a Map/Reduce framework for performing computations, where a collection of map tasks perform computations on independent regions of data and reduce tasks combine the results of the map tasks. In between these operations, users may also apply a local reduction through a combiner task in order to decrease the amount of data that is transmitted over the network. The Hadoop Map/Reduce framework requires users to organize their data into a key-value data format. This format enables the framework to provide users with built-in support for common data processing operations (e.g., sorting data values and removing duplicates), and enforces data parallelism thinking that enables the framework to manage cluster resources (e.g., assigning map tasks to processors).

Hadoop has received considerable interest in the data processing community recently because it is very accessible. In addition to being open source and free to use, Hadoop can be installed and evaluated on a wide variety of platforms. Hadoop was designed with low-cost, commodity cluster hardware in mind. While Hadoop clusters typically expect compute nodes to be equipped with local disks, researchers have demonstrated that high-end, disks-

on-the-side clusters can also be utilized effectively [12]. Hadoop is also supported in several commercial cloud-computing endeavors such as Amazon's elastic compute cloud (EC2) [13]. Thus it is possible for researchers to pay a vendor to run a large Hadoop application at scale if needed. The main drawback of utilizing a Hadoop cluster as a DWA is that users must convert their data processing applications to a form Hadoop can utilize. Additionally, users often find it is challenging to configure Hadoop in a way that maximizes performance.

3.3. DWA Test Systems

As a first step in evaluating whether DWAs can be utilized effectively in mesh analysis applications, we established entry-level DWA test systems for both Netezza and Hadoop. It is important to stress that these platforms are *minimal systems* for DWA work. While they provide basic parallel-processing platforms for out-of-core experiments, they offer only a fraction of the performance that production-level systems can achieve. However, the intent of this paper is to evaluate the challenges involved in adapting scientific analysis algorithms to these systems, and therefore the platforms are sufficient.

For the Netezza experiments we utilized a Netezza Performance Server 10050. This half-rack system employs 54 active SPUs to present the user with over 5 terabytes of database space. A built-in PC with dual AMD Opteron processors functions as the head node and access point for the database. While users can connect to the database remotely through ODBC connections, we performed testing directly on the head node in order to remove network transfer effects. For debugging purposes, we also utilized a standalone Netezza SPUbox test system which features four SPUs. This test system allowed us to prototype and debug UDFs before they were executed on the full system.

For Hadoop cluster experiments we utilized a small cluster named Decline that was recently retired from service as a visualization system. Unlike the majority of Sandia's clusters, Decline's nodes are equipped with local hard drives. A node in this cluster features two 2 GHz AMD Opteron processors, 4 GB of memory, a pair of RAID SATA hard drives, and both GigE and 4x InfiniBand network connections. Benchmarks of the local hard drives revealed that a C program could read data at up to 140 MB/s when transferring large blocks of data. The cluster currently has 19 nodes that operate reliably. We installed the 0.18 release of Hadoop on the cluster and configured it to utilize the GigE network for communication, given that most Hadoop clusters are designed to use low-cost hardware. Similar to other installations, HDFS was configured to use the local disks for storing data, with a default block size of 64 MB.

While we provide performance measurements for these two systems, it is important to note that we have not normalized the results in any way to provide a fair comparison. In terms of compute power, our Hadoop cluster employs a small number of powerful nodes while the Netezza system utilizes a large number of embedded nodes. Both systems employ the same, basic GigE interconnect.

4. Mesh Schemas

The first challenge in adapting mesh analysis algorithms to run on DWAs is to determine a suitable means of representing the mesh data in the DWA's native format. An ideal schema strikes a balance between representing the data in a space-efficient manner and representing it in a way that makes it easy for users to access. For example, consider the problem of defining the coordinates for all elements in a mesh. Given that elements share vertices with their neighbors, the space efficient means of representing this information is to use two tables: one for holding the coordinates of all the unique vertices and another for defining the vertex indices that belong to each element. While this approach minimizes data replication, it complicates the user's environment because multiple table lookups are required whenever an element's coordinates are required in a calculation. In contrast, a simpler schema would replicate the coordinate information in the element table. This approach simplifies the programming environment, but dramatically increases dataset size.

Mesh datasets contain two types of data: structural and variable. The structural portion of the dataset provides initial geometry information about the meshes, and is comprised of the initial coordinates for each vertex, the set of vertices that define each element, and the set of elements that belong to each mesh. We decided to represent each of these data values as its own table in order to reduce redundancy. The variable data portion of the dataset holds both element and vertex data values (e.g., pressure, temperature, or displacement) for each timestep in the simulation. For our schema, we use two tables to hold these variable data values: one for elements and the other for vertices. Each variable name is a column in the table, with a row providing all data variable results for a particular element or vertex at a particular timestep.

We constructed multiple dataset generators to create data for the experiments presented in this paper. These generators enabled us to create a variety of dataset sizes while avoiding export control issues involved in discussing Sandia's applications. Each dataset generator produces a set of tab-separated text files for the various fields in the

dataset. For the Netezza system, the tables are ingested using built-in command line tools that automatically convert the data the database's internal, distributed representation. For the Hadoop cluster, we first convert the text files to binary data files that are easier to access in a random fashion. Then, data files are pushed out to nodes and imported into each map task using a custom file reader.

5. Threshold Volume Calculation

The first data analysis operation that we adapted to run on our DWAs was a threshold volume calculation. Analysts often need to determine how large a particular effect is within a mesh at a specific timestep. For example, one analyst working on safety factors for hydrogen refueling stations for automobiles conducted a simulation where an open nozzle leaked hydrogen gas into a room with multiple chambers. The analyst computed the total volume a particular mole fraction of the gas has occupied at each timestep as well the rate of change for the volume between timesteps. These numbers quantified the amount of leakage that took place during the simulation and were also used to determine when the simulation had reached a steady-state solution.

5.1. Algorithm

The threshold volume calculation can be implemented in a simple, brute-force manner through a marching-cube [14] style approach that sums the contribution of each element to the total volume estimate. For each element, the algorithm compares the element's variable(s) (e.g., hydrogen mole fraction) to a threshold value to determine if the element is above or below a cutoff. If the element is above the threshold value, its volume is added to the total volume. For this implementation, we assume that elements are hexahedra and that an element variable is used for thresholding. In order to accommodate the numerous forms of hexahedron, we decompose an element into six tetrahedra and apply vector math to compute the volume of each tetrahedron. We note that if vertex variables were used instead of element variables, the tetrahedra decomposition could also be used to provide better accuracy.

5.2. Netezza Implementation

We constructed three implementations for the volume calculation to experiment with different development approaches. The first implementation is a stand-alone C++ program that simply pulls relevant data from the database to a local computer and performs the computation in host memory. While this implementation was primarily constructed to verify results, its poor performance highlighted the fact that the network link between the database and the client can be a substantial bottleneck when working with large datasets. This obstacle motivates alternatives where the user performs as much work as possible on the actual database system.

The other two implementations perform the calculation in the database using either plain SQL or a combination of SQL and a UDF. Both of these implementations accomplish their work with a single database query that is comprised of three parts. First, both implementations must assemble the data values that are necessary for the computation. This operation requires nine joins: eight to convert the element's node IDs to coordinates and one to acquire the element variable that is needed for a particular timestep. Second, a simple threshold operation is performed to remove elements from the query that are below the desired cutoff. Finally, a running sum for the threshold volume is computed. In the plain SQL implementation, this operation is explicitly written in the SQL query with multiplication and subtraction operations that the database can interpret and process. The SQL/UDF implementation instead performs this computation in a custom C++ UDF. While this particular UDF did not provide any speed advantages over the plain-SQL implementation, we found that it greatly reduced the SQL and made the implementation much more readable.

5.3. Hadoop Implementation

The Hadoop adaptation of the threshold volume calculation performs all of its work in a single pass. A number of map tasks are used to extract information from the binary input data files in parallel and generate all of the data values that are needed for the computation. Thresholding is performed during this read operation in order to remove unnecessary data values as early as possible. These map tasks also compute the individual volumes of elements as they are read. As each volume is calculated, it is assigned a key of 1. A local combiner task is then used to condense the key-value results from a local mapper to single key-value pair by summing the element volumes and again assigning the result a key of 1. Finally, a global reduce operation merges the results of each combiner into a single sum for the entire cluster.

5.4. Performance Comparison

A synthetic mesh dataset generator was constructed for the threshold volume calculation that created datasets comprised of many, independent hexahedral elements that are randomly placed and annotated with variable data that is randomly generated. The data generator was configured to produce a variety of meshes, ranging from 500K elements to 20M elements. The data was then ingested into both the Netezza and Hadoop platforms.

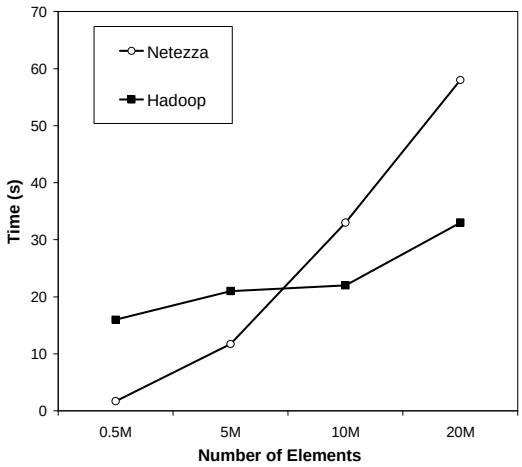

Figure 2: Performance measurements for the threshold volume on Netezza and Hadoop.

The timings for various dataset sizes are presented in Figure 2 for both DWAs. As expected, both platforms were able to process data in a reasonable amount of time, due to the data-parallel nature of this algorithm. For small datasets, the Netezza outperforms the Hadoop cluster. This characteristic can be attributed to the fact that Netezza requires less overhead to launch any particular job than Netezza. However, as dataset size increases, the Hadoop cluster performs better. An examination of the performance for both implementations revealed that nearly all of the overhead of the operation is associated with performing joins to generate data values. This trait suggests that while separating structural data into separate tables saves space, doing so can greatly degrade performance.

6. Element Pairing

The second analysis algorithm that we adapted to our DWAs is an element-pairing algorithm that is part of a larger application that quantifies how much damage is caused when two objects collide. In the example simulation containing this algorithm, an indestructible object is rammed into a deformable object. In order to simulate realistic fractures, the deformable object is modeled as two separate meshes that are bonded together at the surface where the collision takes place. Analysts are interested in observing how far apart element pairs in the seam move apart as the simulation progresses. The distances between these element pairs help quantify cracks, tears, and holes created in the collision.

The central challenge in implementing this analysis is automatically generating the list of element pairs that are pressed against each other at the start of the simulation. Due to the way the meshes are constructed, we cannot assume that a pair of touching elements will share the same vertices or vertex coordinates. Additionally, numerical precision issues dictate that the distance between the faces of two touching elements will be small, but not necessarily zero. Equipped with no additional information about where the meshes are actually touching, the hardship of this pairing problem is that it may require on order of N^2 element face comparisons.

6.1. Algorithm

Our approach to implementing the element pairing algorithm involves computing the distances between the exterior faces of the two meshes, and then selecting the pairs with minimum distance between their faces. This work is divided into three phases. First, all faces are generate for each element in each mesh (e.g., six faces per hexahedron). Second, the face list for a mesh is reduced to the set of exterior faces for the mesh by removing all faces that appear more than once in the list. It is important to note that this reduction must take into account that a single face can be represented by different combinations of its vertices (e.g., ABCD is the same CDAB). Finally, each exterior face for the first mesh is paired to the face in the second mesh that is closest, geographically. For simplicity, we use the distance between face centers as the distance metric.

For the purposes of evaluating the computing performance of the DWAs in the worst case, we chose to deliberately ignore obvious optimizations that reduce the dataset early in the analysis process. For example, we developed a simple bounding-box filter for the first stage of execution that removed all faces in the first mesh that were too far away from the bounding box of the second mesh to be considered. This filter greatly reduced the face list sizes and thus dramatically improved performance. We exclude this filter from the experiments reported in this paper because (1) there are always canonical examples where these types of filters fail and (2) we wanted to present the pairing portion of the algorithm with a sizable amount of work to evaluate how well the systems perform at scale.

6.2. Netezza Implementation

The Netezza implementation of the element pairing algorithm was written as a sequence of SQL queries that store intermediate data values in temporary tables in the database. The first task of generating all possible faces for a mesh is performed through six insert statements (one for each face in each hexahedron). Each insert requires five joins to assemble the necessary data: one to locate all of the elements belonging to a mesh and four to generate the coordinates of the face's four vertices. During the insertions, the vertices for a face are averaged together in order to generate a center that can be used for the final distance estimation task.

The second task of reducing the face lists to just the external faces proved to be more challenging because of the vertex ordering problem. Our solution was to generate a unique key for each face that could be represented as a single data value in the database. We constructed a UDF that took the IDs of the four vertices as input, sorted them in numerical order, and then created a single binary blob value that the database could associate with the face. This blob functioned as a key that allowed us to use Netezza SQL's "Group By" and "Having (Count=1)" syntax to reduce the face list to face entries that appear once in the dataset and are therefore part of the mesh's exterior.

The final task of locating pairs of touching faces is the most time consuming portion of the algorithm. Similar to the previous step, we utilized the "Group By" SQL to compute the distance between every exterior face in one mesh and every exterior face in the other mesh, followed by a MIN() aggregate function to reduce the data to the minimal pairings. An additional threshold operation was applied to remove face pairs from the result that were separated by more than a specified amount of distance. Through experimentation, we were able to verify that the MIN aggregate properly discarded non-minimal values as it stepped through the dataset, as opposed to computing all values and then selecting the minimum results.

6.3. Hadoop Implementation

The Hadoop implementation of the element pairing algorithm is written as a series of four map/reduce phases:

1. The first phase uses multiple map tasks to read data from binary input files in parallel and generate Hadoop sequence files that contain all of the information necessary for the work. The output of the map tasks is written back to HDFS, resulting data blocks being distributed throughout the cluster.
2. The second phase utilizes multiple map tasks to generate all of the faces in the mesh, and then a reduce task to remove all but the exterior faces in each mesh. In order to identify faces regardless of vertex order, a key is generated for each face that is comprised of the ordered list of vertex IDs.
3. The third phase performs all distance comparisons between exterior faces. This operation proved to be a difficult problem to implement in map/reduce because it involves comparing results from two different data streams. Similar to other researchers faced with the same problem, we decided to implement this join operation somewhat outside the Map/Reduce paradigm: we designed a map task that loaded one mesh's face values into memory and then streamed the other mesh's face values through for comparison. In order to handle memory limitations, this phase was designed to support an iterative approach that loads only a portion of the first mesh into memory at a time. If multiple output files are acceptable, there is no need to perform a reduction here. The performance of the join in this phase was further improved by leveraging Hadoop's DistributedCache feature[] to efficiently distribute data to all maps in the cluster.
4. The final phase utilizes multiple map tasks to find the minimum distance pairs between all iterations of the previous phase, and uses a reduce operation to collect results into a single file.

6.4. Performance Comparison

A second synthetic dataset generator was constructed for the element pairing application. This generator produces datasets that are comprised of two meshes that are placed in close proximity to each other. The cubes are

offset by small amounts in each direction to reflect a more realistic example. The element pairing algorithm was run on datasets ranging from 0.5M elements to 20M elements on both DWAs.

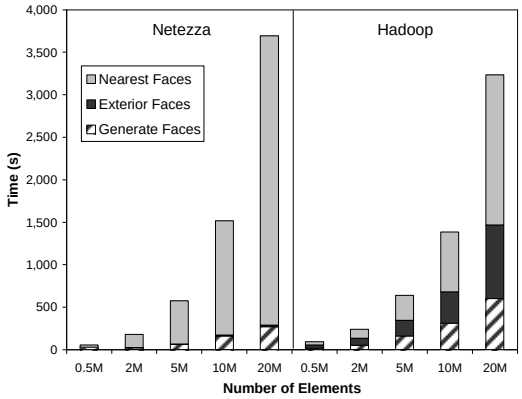

Figure 3: The performance of Netezza and Hadoop implementations for element pairing.

Timing measurements for the face-pairing tests are presented in Figure 3. We measured the amount of time required to process the three general phases of the algorithm. Netezza provided much better performance in the first two phases of the algorithm than Hadoop. In particular the Netezza implementation was able to perform the exterior face operation at nearly a constant rate. However, the final phase of performing the N^2 distance computations proved to be the dominant operation in the calculation, as well as the bottleneck in the Netezza implementation. By being able to explicitly pull the second face list into memory, the Hadoop implementation provided better performance. We are currently investigating whether these operations could be performed in Netezza utilizing UDFs that employ the SPUs memory for scratchpad data.

7. Observations

This work has provided many insights into utilizing DWAs for scientific dataset analysis. The following are observations that originate from adapting these algorithms, and DWAs in general.

Start Costs: Both Netezza and Hadoop have relatively high startup costs for performing operations. Similar to other databases, a Netezza query requires at least a half second to complete. Startup costs for Hadoop are on the order of 10-20 seconds. These overheads emphasize the importance of performing complex computations on the DWAs when possible.

Floating-Point Limitations on Netezza: The PowerPC processors in our current Netezza system do not provide hardware support for floating-point computations and therefore must implement the calculations in software. This issue is problematic for scientific datasets which are largely based on floating-point data. In order to get an estimate of how much this limitation impacted performance, we conducted an experiment on the Netezza where the element-pairing algorithm was changed to use integer data values instead of floating point. The integer implementation completed in a third of the time that the floating-point version required.

Tunability: In order to improve performance, we went through several development iterations on each platform. On the Netezza system, the major complaint was that there were relatively few means by which we could refactor the algorithm. Netezza is designed to do optimization for the user, and therefore does not provide many programming paths that a user could explore to improve performance. In contrast, Hadoop provides the opposite environment: users can easily be overwhelmed by the variety of ways they can refactor their algorithms to Map/Reduce operations.

Programming Interfaces: An ongoing question for our work has been whether SQL and Map/Reduce are sufficient data-parallel languages for implementing nontrivial scientific analysis functions. The two adaptations presented in this paper confirm that basic algorithms can be adapted to these languages and executed in a parallel environment. However, it is important to note that the majority of our effort involved finding ways to get around language limitations. For SQL, we feel that performance was compromised in the final phase of the element pairings because there was no obvious way to do an all-to-all computation efficiently. For Hadoop, we continuously struggle with the

problem of merging two data streams when random access is required. Similar to other developers, we solve this problem by going outside of the Map/Reduce paradigm.

Debugging: Debugging is challenging on any parallel platform. For Netezza, we found SQL debugging to be relatively straight forward, due to the robustness of the SQL standard. However, UDFs must be developed with caution, as programming errors can freeze a SPU or crash the database. Netezza's UDF interface provides basic support for forcing users to be cautious, with rigid input/output parameter checking and stack overflow guards. Hadoop developers have constructed a number of facilities for debugging. Hadoop also allows users to debug a job on a local machine before it is run on a cluster.

Portability: Ideally, scientific users need to be able to easily move analysis programs from one platform to another without drastic changes. Developing for Netezza SQL is appealing in that sense because SQL-compliant code should be portable to different databases. We initially prototyped our SQL algorithms on a MySQL database and then changed the program to use the Netezza. While this transition went fairly smoothly, there are subtle differences between each database that make portability challenging. Additionally, UDFs written for Netezza are not usable on any other platform. Hadoop on the other hand can be run on a wide variety of platforms. As discussed in the tunability observations, the issue is refactoring programs to maximize cluster resources.

8. Summary and Future Work

Scientific applications are generating massive datasets that are difficult to analyze utilizing traditional, offline approaches. An emerging class of systems known as Data Warehouse Appliances provides an opportunity to improve the scale at which automatic data analysis operations can be performed through the use of parallel storage hardware and data-parallel programming interfaces. In this paper we have explored how two scientific data analysis algorithms can be adapted to the Netezza parallel database and a Hadoop-equipped cluster. We have confirmed that the data-parallel programming interfaces for these platforms are sufficient for implementing our out-of-core algorithms, and that the parallel hardware of these systems could be utilized. However, it is important to note that the APIs for these platforms required a good bit of planning and experimentation in order to achieve good parallel performance.

The next step in this work is to scale the hardware of both platforms to observe how far the systems can scale before internal hardware bottlenecks surface. In a similar effort, it will be necessary to investigate how data can be transferred in and out of DWAs so that existing MPP platforms can leverage DWAs as high-speed data store for simulations. Finally, we see the value of emerging language efforts as an opportunity to simplify the programming effort for using DWAs. For example, Hadoop's Pig Latin [15] effort represents an easy-to-use scripting language for Hadoop that can serve as a convenient means for implementing dataflow without having to explicitly manage the sequence of Map/Reduce operations.

References

[1] R. Meisner, "A Platform Strategy for the Advanced Simulation and Computing Program," NA-ASC-113R-07-Vol.1-Rev.0, August 2007.

[2] K. Alvin, "ASC National Code Strategy Simulation-Based Complex Transformation," NA-ASC-108R-09-Vol.1-Rev.0, January 2009.

[3] O. Zeinkiewicz, R. Taylor, and J. Zhu, The Finite Element Method: Its Basis and Fundamentals, Sixth Edition, published by Butterworth-Heinemann, 2005.

[4] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre, "Remote Large Data Visualization in the ParaView Framework", in proceedings of Eurographics Parallel Graphics and Visualization, May 2006.

[5] J. Laros, L. Ward, R. Klundt, S. Kelly, J. Tomkins, B. Kellogg, "Red Storm IO Performance Analysis," in proceedings of 2007 IEEE International Conference on Cluster Computing, September 2007.

[6] P. Braam, "The Lustre Storage Architecture," available at www.lustre.org, 2002.

[7] G. Davidson, K. Boyack, R. Zacharski, S. Helmreich, and J. Cowie, "Data-Centric Computing with the Netezza Architecture," Sandia Technical Report SAND2006-3640, April 2006.

[8] A. Bialecki, M. Cafarella, D. Cutting, O. O'Malley, "Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware", Wiki at http://lucene.apache.org/hadoop

[9] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," in Proceedings of the 19th ACM Symposium on Operating Systems Principles, October 2003.

[10] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proceedings of the 6th Symposium on Operating System Design and Implementation, 2004.
[11] D. Borthakur, "The Hadoop Distributed File System: Architecture and Design,"
[12] W. Tantisiriroj, S. Patil, G. Gibson, "Data-intensive File Systems for Internet Services: A Rose By Any Other Name …", Carnegie Mellon University Parallel Data Lab Technical Report CMU-PDL-08-114. October 2008.
[13] Amazon Elastic Compute Cloud (EC2), http://www.amazon.com/ec2/ , June 2009
[14] W. Lorensen and H. Cline, "Marching cubes: A High Resolution 3D Surface Construction Algorithm," in proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, 1987.
[15] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, 2008.