Benchmarking the Performance of a Cluster-Based Geospatial Database System

Venkata Mahadevan	Mahdi Abdelguerfi	Kevin Shaw
Dept. of Computer Science	Dept. of Computer Science	Naval Research Laboratory
University of New Orleans	University of New Orleans	Stennis Space Center
New Orleans, LA 70148	New Orleans, LA 70148	MS, 39529
Email: venkata@cs.uno.edu	Email: mahdi@cs.uno.edu	Email: shaw@nrlssc.navy.mil

Abstract

The advent of inexpensive high performance computing clusters has opened the doorway to a wide range of applications that hitherto had been inconceivable. Such clusters can provide certain applications with a substantial boost in both compute and I/O performance. Therefore, it seems logical that Geographic Information Systems (GIS) applications would be an ideal fit for such systems. This paper presents a scheme to distribute geospatial data across several compute nodes in a cluster. Several variations of this scheme are presented and the performance of each is benchmarked by using various geospatial queries to retrieve the data.

1. Introduction

In recent years, several remarkable phenomena have coalesced to pave the way for a revolution in the field of high performance computing. These include the emergence of relatively inexpensive but powerful off-the-shelf desktop computers, fast interconnect networks such as Fast Ethernet and Gigabit Ethernet, and the rise of the GNU/Linux operating system. At the same time, the rapid proliferation of Geographic Information Systems (GIS) technology and an increasing need for more computing power to process and query the massive amounts of information stored in such systems has provided an ideal setting for evaluating the performance of a geospatial database distributed across a compute cluster.

A compute cluster, for all intents and purposes, can be defined as a "Pile-of-PCs" [1] interconnected via some sort of network. Each node in the cluster usually has its own processor, memory, and optionally an I/O device such as a disk. However, it is important to note that a cluster is not simply a network of workstations (NOWs) – in a cluster, the compute nodes are delegated only for cluster usage and nodes typically have a dedicated, "cluster-only" interconnect linking them. While there are many clustering schemes available, by far one of the most popular and well-supported paradigms is that of the "Beowulf Cluster" [1]. Indeed, any discussion on GNU/Linux clustering is usually permeated with references to such systems. In the Beowulf paradigm, all of the compute nodes (also called slave nodes) are isolated on a high-speed private network that is not directly visible to the outside world. A single computer called the "master" or "head-end" provides a single entry point to the cluster from the external network. This machine is sometimes referred to as the login or submit node. Essentially, the master node is a system with 2 network interfaces – one connected to the private Beowulf network and the other connected to the regular LAN. Users of the cluster will typically log in only to the master node. From here, they can spawn processes that will execute on the slave nodes. Beowulf clusters are typically applicable to any area of research where a speedup in program execution time is possible by splitting a large job into several sub-tasks that can

run concurrently on the compute nodes. This capability provides an intriguing setting for evaluating the use of such a cluster for hosting large GIS databases.

Geospatial data is constantly being gathered via a variety of sensors, satellites, and other devices. The quantity of data collected by these units is staggering – NASA's Earth Observing System (EOS), for example, generates 1 terabyte of data on a daily basis [2]. However, it is not sufficient to simply have this data in "raw" form; it needs to be organized so that useful information can be extracted from it – this is where GIS technology comes into the picture. GIS provides, among other things, methods to organize geospatial data into features. Features can represent real-world objects such as rivers or roads and have both a location component (with the Earth as a possible frame of reference) and an attribute component. Using this information, it is possible to construct *geospatial queries* that extract useful information from a GIS database. By using geospatial queries, it is fairly straightforward to narrow down a search for features in a GIS database to return only features located within a certain area (usually specified by a bounding box or bounding rectangle) or at a certain distance from a specific point. This provides an incredibly powerful and intuitive mechanism for mining relevant data from large databases. On the other hand, due to the large size of many GIS databases, geospatial querying can suffer from performance problems. The work presented in this paper attempts to improve the performance of geospatial querying by distributing data among the compute nodes in a cluster and then executing geospatial queries across these nodes.

The remainder of this paper will focus on the performance aspects of geospatial querying as they pertain to a geospatial database distributed across a Beowulf cluster and is organized as follows. In Section 2 we describe the fundamental design of our Beowulf cluster and the environment in which geospatial data is stored and organized. Section 3 discusses the test data selected for use in the benchmarking process. In Section 4 we explain the methodology used to experimentally gauge geospatial querying performance. An analysis of the results obtained from benchmarking is provided in Section 5. In Section 6, we conclude with a discussion of the results and suggestions for future work.

2. Overview of System Organization

The Beowulf cluster housed at the Department of Computer Science at the University of New Orleans consists of 72 compute nodes, 1 login/submit node, and 1 file server. 63 of the slave nodes are 2.2 GHz Intel Pentium IV systems with 1 GB of memory, 20 GB of local disk storage, and Fast Ethernet networking; the remaining 9 nodes are 2.4 GHz Intel Pentium IV systems with 1GB of memory, 20 GB of local disk storage, and Gigabit Ethernet networking. The file server is a dual 1.4 GHz SMP Intel Xeon system with 2 GB of memory, 500 GB of RAID-1 disk storage, and Gigabit Ethernet networking. Last, but not least, is the master node which is a dual 2.2 GHz SMP Intel Xeon system with 2 GB of memory, 300GB of disk storage, and 2 Ethernet interfaces. The interface that links the cluster with the private Beowulf network is Gigabit Ethernet; the external interface is 100BaseTX. All of the systems in the cluster are networked together by a Cisco Catalyst 4000 series switch with 10/100/1000 auto-sensing ports and a 12-Gbps backplane. Redhat Linux 7.3 is the operating system for all nodes in the cluster and the Warewulf Clustering System [3] is used to provide support utilities for cluster monitoring and maintenance. This cluster was recently benchmarked using HPL 1.0, a portable,

freely available implementation of the standard High Performance Computing Linpack Benchmark. HPL solves a random, dense linear system in double precision (64 bits) arithmetic on distributed memory computers [4]. Benchmarking yielded a "theoretical peak" performance of approximately 63 Gigaflops. The amount of raw computing capacity provided is therefore quite substantial.

In order to store geospatial data and issue geospatial queries on it, a DBMS that provides spatial extensions and support for geospatial querying is required. For this purpose, the PostgreSQL object-relational DBMS [5] was selected. PostgreSQL is a good choice for use on a GNU/Linux system (such as our Beowulf cluster) because of the robust, native support that it provides on this platform. PostgreSQL also has a spatial extension called PostGIS [6] that follows the OpenGIS Consortium's "Simple Features Specification for SQL", a proposed specification to define a standard SQL schema that supports storage, retrieval, query, and update of simple geospatial feature collections via the ODBC API [7].

Figure 1 shows the organization of our proposed cluster-based geospatial database. Each of the slave nodes executes the PostgreSQL server engine and has its own database instance stored on its local disk (the file server is not used because of the potential performance penalty associated with a large number of nodes accessing a shared filesystem simultaneously). The master node uses the UNIX remote shell command *rsh* [8] to spawn geospatial query processes on each of the slave nodes. After each query process finishes executing on each of the slave nodes, the results are returned to the master, which in turn assimilates them to provide the final output for a particular geospatial query.

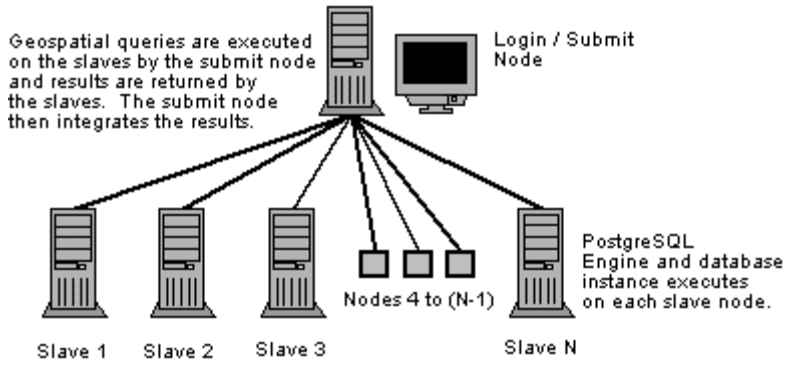

Figure 1. System Organization

Geospatial data is distributed as evenly as possible across the slave nodes; depending upon the number of slave nodes participating in the system, each node may hold more or less data in its local database. The *entire* data set is organized by feature id, feature type, and feature geometry. The feature id is simply an integer that uniquely identifies each feature in the database. Since it is the primary key, data (features) is split among the nodes based on the number of unique features comprising the entire data set i.e. there is no replication or duplication of data. The feature type may be one of 7 different types specified by the "Simple Features" specification of the OpenGIS Consortium [7]. These include *point*, *linestring*, *polygon*, *multipoint*, *multilinestring*, *multipolygon*, and *geometrycollection*. Finally, the feature geometry is the set of geospatial coordinates

comprising a particular feature. It is represented in the database as human readable text and the coordinates may be either 2-dimensional or 3-dimensional.

3. Description of the Test Data

Two large geospatial data sets were obtained from the Bureau of Transportation Statistics (BTS): the 2002 National Transportation Data Hydrographic Features and the Railway Network (1:100,000 base scale). The hydrographic features are a state-by-state database of both important and navigable water features creating a single, nationwide hydrography network containing named arcs and polygons [9]. The size of this data set is approximately 310 MB in compressed ESRI Shapefile format. When imported into a PostgreSQL database, the size is about 300 MB with 517537 unique features. The railway network is a comprehensive database of the railway system of the United States at the 1:100,000 scale. The data set covers the 48 contiguous states plus the District of Columbia [10]. The size of this data set is approximately 41 MB in compressed ESRI Shapefile format and 92 MB in a PostgreSQL table with 166687 unique features. Figure 2 shows a high level snapshot of this data. The geometry of the features in both data sets is represented as object type multilinestring from the OpenGIS Specification aforementioned. Both these data sets contain a large number of features with each feature possessing a sufficient degree of complexity to warrant distribution of the data set across multiple nodes.

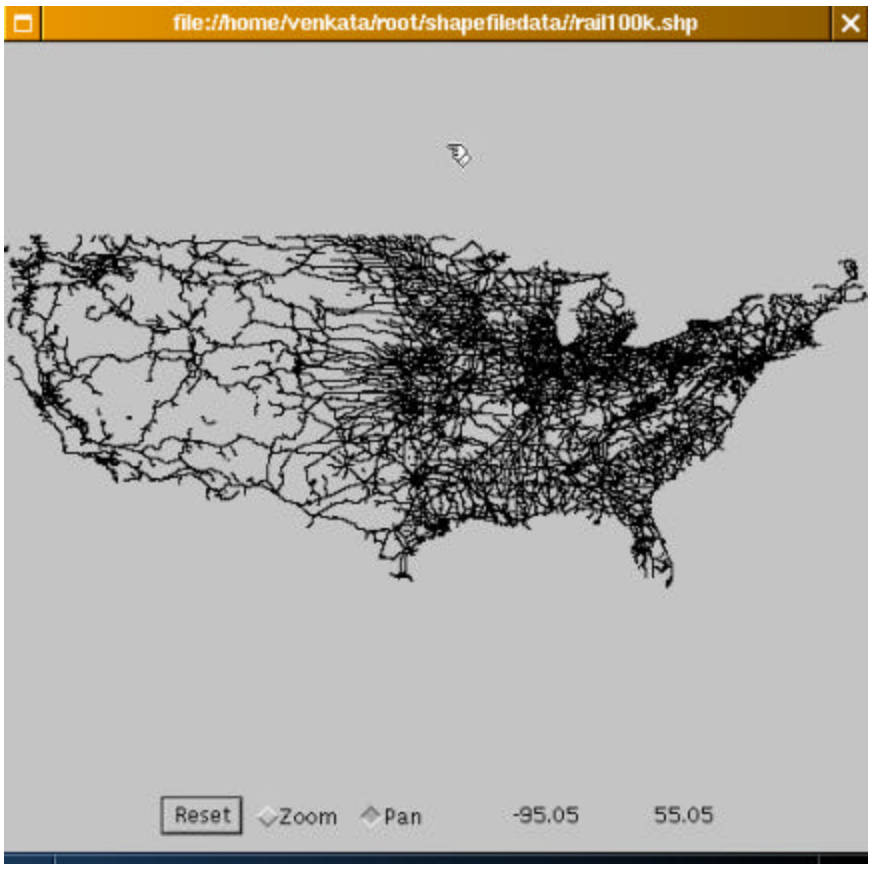

Figure 2. The U.S. Railway Network

4. Benchmarking Methodology

The first step in the benchmarking process was to load the test data into the various database instances dispersed across the cluster nodes. This was accomplished as follows: a system script executes at the master node which partitions the total data set into roughly equal pieces keyed by feature id; the pieces are then batch loaded into each database instance in the cluster. The number of pieces of course depends on the total number of slave nodes participating in the system. Once the test data has been loaded, queries can be executed on the slave nodes. Geospatial queries are similar to standard Structured Query Language (SQL) queries but provide a richer set of functions for dealing with geospatial data. For example, it is possible to express queries such as "return all features within 100 units of a particular point" using the PostGIS extensions to PostgreSQL. The following queries were selected for use in our evaluations:

Distance Query:

```
select gid, the_geom
from <table_name>
where distance (the_geom, GeometryFromText(
                 'POINT(X Y)', -1)) < ?;
```

Window Query:

```
select gid, the_geom
from <table_name>
where the_geom && GeometryFromText('BOX3D
(X1 Y1, X2 Y2)'::box3d,-1);
```

Combination Query:

```
a combination of the above 2 queries i.e.
return all the features within a specified
distance from a certain point that are also
contained in the specified bounding box.
```

It is conceivable that the distance function will be a computationally intensive operation within the queries because a separate distance calculation is required for finding the distance between the specified point and each point in the table (in other words, a separate distance calculation for each row of the table is needed). For both data sets, the distance value was set to 500 units; this is more than large enough to cover the entire spatial domain (bounding coordinates) of each data set. The bounding boxes for both data sets were also set to reflect their respective bounding coordinates [9, 10] and the north-west coordinates of the bounding boxes were chosen as the point of reference for distance queries. These particular values were used in order to ensure that the system is heavily loaded i.e. many thousands of rows are encompassed within a query and each query executes for a measurable time duration.

Each query was executed on 3 different cluster configurations: 16 slave nodes, 32 slave nodes, and 64 slave nodes. The reference system was the master node which holds an entire copy of both data sets – it was benchmarked separately. The benchmarking process itself works as follows: a process on the master system executes each query concurrently on each of the slaves (via *rsh*) and forks a sub-process to keep track of the execution time. Each slave processes the query independently and returns the results to the master. When the last slave in the group has finished processing, the master process combines the results and its timing sub-process computes the time in seconds that the

entire operation took to complete. In the next section, we will compare and contrast the benchmark results obtained under different cluster configurations as well as those of the reference system.

5. Results

Figure 3 summarizes the results obtained from benchmarking using 16, 32, and 64 slave nodes (the reference system is also included for comparison purposes) on the first data set (hydrographic features). Each benchmark was run several times for all the configurations; only averages are reported.

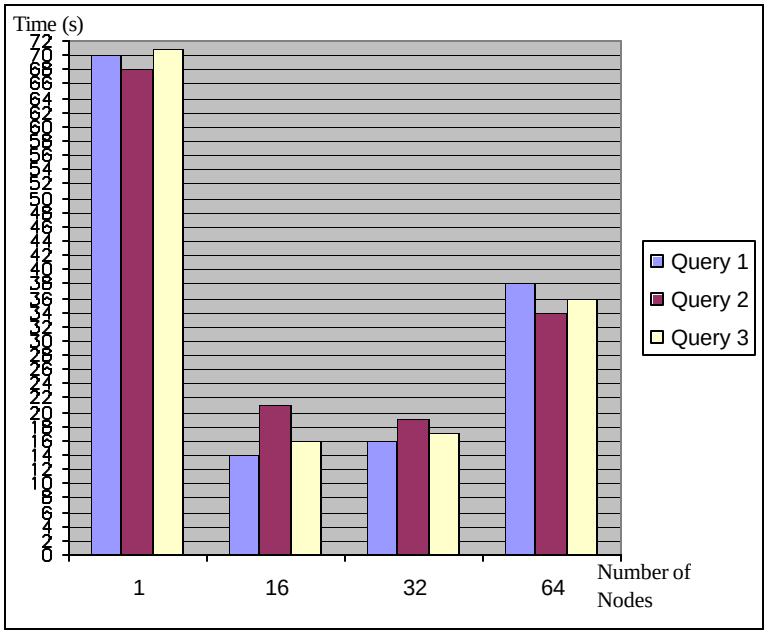

Figure 3. Benchmarks for Data Set 1

The results presented in the above chart show that there is certainly a tangible performance gain that is associated with distributing and querying the data set across multiple nodes. With 16 or 32 nodes, queries are executed in roughly 25% of the time that it takes with the reference system. The surprising results are the increase in processing time that occurs with 64 nodes and the lack of improvement in processing time when using 32 nodes instead of 16. These "anomalies" may be attributable to several factors including the data set itself or the distribution of the data among the nodes, but more likely the reason is probably due to the increase in communication overhead that occurs when query results are returned to the master node from a larger number of slave nodes i.e. network communication overhead negates any processing gains achieved by having more processors and disks in the overall system.

Benchmarks from Data Set 2 (U.S. railway network) are illustrated in Figure 4. Given the relatively small size of this data set when compared to the previous one, query processing times were greatly reduced across all cluster configurations and query processing on the cluster was about 3 times as fast as that of the reference system. The speed variations among the various cluster configurations, however, were insignificant –

the small size of the total data set is possibly the largest contributing factor in this case as it does not benefit greatly from being distributed on a larger number of CPUs nor does it cause an undue amount of network communication overhead when distributed on additional nodes.

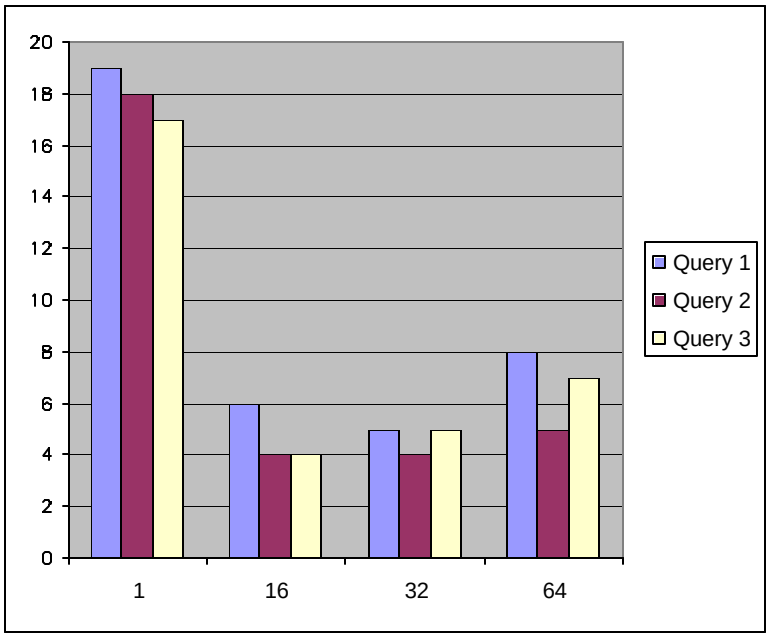

Figure 4. Benchmarks for Data Set 2

6. Conclusions

A scheme to distribute geospatial data and execute geospatial queries across the compute nodes of a Beowulf cluster was discussed in this paper. The performance of this scheme under various cluster configurations was benchmarked, and the results certainly advocate using a compute cluster for mining geospatial data – large data sets such as the hydrographic data can benefit greatly from a reduction in query processing time when distributed across multiple nodes. However, it is expected that such a cluster will really make its mark when handling even larger data sets, on the order of tens of gigabytes or even terabytes of data, volumes that are simply unsustainable on even the most powerful non-parallel systems.

There is still a great deal of work that can be done in addition to that presented in this paper. We are planning to develop a software toolkit to support easy distribution of geospatial data across a Beowulf cluster and the visual querying of this information. These tools will also be adaptable to support applications other than GIS. In particular, we would like to make them available to researchers in fields such as bioinformatics, or in any field that would benefit from an easily deployable solution that effectively exploits the power of parallelism.

References

[1] Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling, “Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs”, Goddard Space Flight Center, CACR Caltech, Proceedings IEEE Aerospace, 1997

[2] Shashi Shekhar, Sanjay Chawla, “Spatial Databases: A Tour”, Prentice Hall, 2002

[3] Greg Kurtzer, “The Warewulf Clustering System” http://www.runlevelzero.net/greg/warewulf/stuff/lbnl-pres/

[4] Jack J. Dongara, Piotr Luszczek, Antoine Petitet, “The LINPACK Benchmark: Past, Present, and Future”, 2001

[5] John Worsley, Joshua Drake, Andrew Brookins (Ed.), Michael Holloway (Ed.), “Practical PostgreSQL”, Command Prompt Inc., 2002

[6] Refractions Research Inc., “PostGIS Manual” http://postgis.refractions.net/docs/, Refractions Research Inc., 2003

[7] OpenGIS Consortium Inc., “OpenGIS Simple Features Specification for SQL”, OpenGIS Consortium Inc., 1999

[8] Redhat Inc., “Redhat Linux 7.3 System Manual”, Redhat Inc., 2002

[9] Bureau of Transportation Statistics (BTS), “2002 National Transportation Atlas Data Hydrographic Features” http://www.bts.gov/gis/download_sites/ntad02/metadata/hydrolin.htm, BTS, 2000

[10] Bureau of Transportation Statistics (BTS), “2002 National Transportation Atlas Data Railway Network (1:100,000 base scale)” http://www.bts.gov/gis/download_sites/ntad02/metadata/Rail100k.htm, BTS, 2002