ALTERNATIVE APPROACHES TO PARALLEL GIS PROCESSING

by

Nathan Thomas Kerr

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2009

ALTERNATIVE APPROACHES TO PARALLEL GIS PROCESSING

by

Nathan Thomas Kerr

has been approved

August 2009

Graduate Supervisory Committee:

Dr. Daniel Stanzione, Chair
Dr. Robert Pahle
Dr. Yi Chen

ACCEPTED BY THE GRADUATE COLLEGE

ABSTRACT

Abstract goes here

Dedication goes here

ACKNOWLEDGEMENTS

Acknowledgments go here.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

Geographic Information Systems (GIS) were designed to model aspects of the world around us. From roads to temperature, GIS data can be used to represent a large range of real-world objects, allowing for sophisticated analysis and processing. Data is usually stored as raster or vector data. Raster data represents the world as a grid, which each grid cell containing attributes associated with that area. The grid is georeferenced and has a specific resolution, which is the amount of space each grid cell represents. Vector data is comprised of a set of geometries such as points, lines, or polygons that are georeferenced. Popular forms of georeferencing include latitude and longitude, zip codes, and street addresses.

The implementations used in this thesis only deal with vector data, though the principles discussed can be applied to other data types.

GIS analysis and simulation are used to understand and develop the environment around us. A common use of GIS data is found in the GPS based car navigation systems common today. City planners will use GIS data in applications such as population growth models, land use planning, and traffic management.

The amount of data to be processed increases as populations grow, communities become more complex, or the size of the processing area grows. In the year 2000, Maricopa County, Arizona, mapped 1.2 million parcels of land in its 9,224 square miles of land. Each parcel of land is represented as a georeferenced polygon along with various attribute data such as a unique parcel id, ownership information, and zoning codes. Parcels of land is just one layer of data that is kept for Maricopa County. Other layers include roadways, railways, rivers, schools, voting districts, etc.

When the data required to complete some GIS processing grows beyond the memory available to a single processor, the processing method must be adapted to fit within that

limitation. A common method is to not load the entire dataset into memory, but to read it off disk as needed and not keep what is not. This method works well if the data is only needed once, as reading from disk is slow.

One application, which is looked at in more detail later, associates businesses with parcels of land (the data was geocoded differently). The algorithm is quite simple: for each business, find the nearest parcel of land with a compatible zoning code. In Maricopa County in the year 2000, there were 34,302 businesses and 1,218,130 parcels of land. While these particular datasets fit into memory today, they did not just a few years ago. Given increased parcel density, or a larger area of land, the memory capabilities of a standard computer are quickly reached. While there are specially made computers with extraordinarily large amounts of memory, they are also extraordinarily expensive.

In addition to memory limitations, the processor also limits the processing that can be done. In this case 41,784,295,260 comparisons between businesses and parcels is made. If a computer is able to make one million comparisons per second, the processing will take approximately 11 hours, 37 minutes. A factor increase in speed to ten million comparisons per second reduces the time to 1 hour 10 minutes. If only there was a single computer that would perform 100 billion of these comparisons per second, then the processing would be done in less than half a second!

As an infinitely fast computer with infinite memory does not exist, more effort is required to perform this processing in a reasonable time.

1.1 Parallel GIS Processing

Using multiple interconnected computers together to complete the required GIS processing allows for increased memory capabilities along with increased computation power. Processing algorithms must be reworked to allow for this collaboration between comput-

ers.

Programming for multiple machines is not the easiest of tasks. A common paradigm is Single Program, Multiple Data[**?**] (SPMD), uses a single program that is run on all the computers included in the parallel computations where each instance of the program operates on different data, for example a different set of businesses. This paradigm simplifies parallel algorithm complexity while providing enough power to increase memory capability and computation power.

The first step in working with an SPMD program is splitting the data. GIS data can be split in several different ways. The specific data splitting method used is dependent on what data is needed on what machine. As GIS vector data is based on a record that contains a geometry with associated attributes, one method that can be used is to evenly distribute the records among the participating machines. Another method takes into account the varying size of geometries in memory by splitting the data up by size while taking into account record boundaries. Another class of data distribution is to geographically split up the dataset, for instance into quadrants with each machine having responsibility for one or more quadrants. This method is more complicated in the setup required and the exceptions that need to be handled such as how to handle geometries that span more than one quadrant.

After the data is distributed, the processing methodology often needs to be adjusted because access to the full dataset at once is no longer possible. Combined with the data distribution, these two additions are needed for a parallel implementation, but not the serial case. This is the overhead required by the parallel case.

Parallel performance is measured through speedup:

$$\text{speedup}(n) = \text{time}(n)/\text{time}(1) \tag{1.1}$$

which compares the execution time of the process on n processors to the execution time on 1 processor. Ideal speedup is n. The quality of speedup gained is efficiency:

$$\text{efficiency}(n) = \text{speedup}(n)/n \tag{1.2}$$

Ideal efficiency is 1.

The scalability of a parallel solution can be seen by graphing speedup by number of processors used. The best realistic case is to have linear scalability, meaning that speedup is directly proportional to the number of processing elements used.

1.2 Alternative Approaches to Parallel GIS Processing

This thesis implements and evaluates two parallel, dataset centric approaches to processing large geospatial datasets on clusters. The first approach, called HadoopGIS, uses the Hadoop[1] map/reduce framework. The second approach, ClusterGIS, uses the more traditional approach to programming for clusters, MPI. Both approaches provide the geospatial operations required by the Open Geospatial Consortium's Simple Features[2] standard. By applying data parallel programming methods and dispensing with record centric processing methods, both these methods create a fairly easy environment to program in while providing significant speedup and scaleup.

HadoopGIS adds GIS capabilities to the Hadoop map/reduce framework. Hadoop is based on Google's MapReduce[3]. Map reduce defines a two-phase method to working with data. The first phase, map, applies a function to every record in a data set. The map function can output one or more key-value pairs. Map is an inherently parallel process, as no record is need to process any other record.

After the map phase, the generated key-value pairs are aggregated by key and passed to reduce processes, one reduce process per key. There are generally many fewer reduce

operations as compared to map operations. Reduce operations are inherently serial, as the operation must have access to all the key-value pairs associated with the key being processed.

ClusterGIS is a library of functions based on MPI[**?**] and the GEOS[**?**] library. MPI is a message passing interface standard that allows multiple computers to collaborate on solving a problem by passing messages between themselves. GEOS is an open source geometry engine that handles the geometric calculations required by GIS processing. The combination of MPI and GEOS creates a cluster based, parallel GIS processing environment.

MPI has become the standard way of programming for clusters. Because ClusterGIS harnesses the power of MPI, a wide variety of parallel algorithms and configurations can be made, while maintaining the ability to execute on most clusters.

The rest of this thesis is organized as follows: Chapter 2 explores related efforts in GIS processing. Chapter 3 then defines the specific requirements needed for a good parallel GIS processing engine. Chapter 4 details the design choices for HadoopGIS and ClusterGIS. Chapter 5 defines a set of tests and list individual results. Chapter 6 evaluates the two implementations by comparing performance results against the requirements defined in chapter 3. Chapter 8 summarizes conclusions drawn from the evaluation.

CHAPTER 2 RELATED WORK

This thesis applies parallel processing techniques to the field of GIS processing. I will first look at the state of GIS processing, then Parallel Processing in general, then parallel GIS processing.

2.1 GIS Processing

Geographic Information Systems (GIS)[**?**] have been in use since the 1960's with the Canada Geographic Information System (CGIS)[**?**] and then moving from the mainframes to current desktop applications like ESRI's ArcGIS[4] product, which began development in the 1980's.

In general, there are two types of GIS data: raster and vector. Raster data is a set of cells, such as pixels in a picture, that have one or more attributes (e.g., temperature, humidity, elevation). Each attribute covers the entire area of the pixel. The entire raster dataset is spatially located and states how much area is covered by each cell.

Vector data is comprised of spatially referenced geometric objects such as points, lines, and polygons. Each object represents something in the real world, and has associated attributes.

Most GIS processing systems are able to handle both raster and vector data sources.

2.1.1 Desktop GIS

Desktop GIS packages such as ArcGIS[4], QuantumGIS[5], and GRASS GIS[6] are commonly used for GIS processing and analysis. While these programs provide graphical interfaces to their GIS capabilities, their capabilities are limited by the computers they run on. Datasets can be too large for their memories and computations can take too long to be

practical.

Desktop packages are often supplemented with a database component such as ArcSDE[7] or PostGIS[8] which acts as a centralized repository for GIS data which can be shared between a workgroup. It is important to note that the database is generally used for storage and sharing, not for computation. Computation is performed by the desktop package.

2.1.2 Database GIS

An alternative to performing GIS processing in a desktop program like ArcGIS, is to employ a geospatial database like PostGIS[8], ArcSDE[7], or Oracle Spatial[9]. Geospatial databases allow centralized access to, and processing of, geospatial data through query languages such as SQL. As data is stored and managed by the database software, advanced database features such as indexes can be utilized to speedup data access and processing.

PostGIS is utilized as the core component of the Urban Systems Frame-work[10] (USF) designed by the Digital Phoenix[11] project group at Arizona State University. Digital Phoenix tries to integrate 3D visualization technology with simulated and gathered GIS data to better understand the impacts of urban planning decisions.

2.1.3 GIS Simulation and Analysis

GIS simulation and analysis can also be done using more specialized environments. UrbanSim[12] is a popular simulation tool for growth models.

GeoDa[**?**] is a spatial analysis tool that includes spatial regressions. PySal[**?**] is a python library that builds on the work done with GeoDa.

2.1.4 GIS Libraries

The Open Geospatial Consortium (OGC) defined a core set of geospatial processing operations in their Simple Features[2] standard. These core operations allow for most geospatial processing needs. One of the main libraries that provides these operations and related data types is the Java Topology Suite[**?**] (JTS). Though written in Java, the JTS has been used as the basis for ports into other languages. The Geometry Engine, Open Source (GEOS) library is a C++ port of the JTS that also provides a C interface. PostGIS is implemented using the GEOS library. The NetTopologySuite[**?**] (NTS) is a port of the JTS into .NET.

As quality libraries are available for a variety of languages, there is no need to re-implement the functionality they provide. This thesis makes direct use of the JTS and GEOS libraries.

2.2 Parallel Processing

2.2.1 Serial and Parallel Shared-Memory Applications

Computer programs are executed by processors. The simplest of programs is made up of a series of operations that are executed in order by the processor. As this type of application is comprised by a series of operations it is known as a serial program. Serial programs are not able to take advantage of more than one processor. To utilize more than one processor at a time, a set of serial programs that can communicate with each other is required. Each serial program in this set is referred to as a thread. Thus multi-threaded programs can utilize more than one processing core. The simplest method of communication between threads is to share a common section of memory. Therefore a parallel shared-memory application could utilize at most the number of processors able to be connected to a single

section of memory.

Both serial and parallel shared-memory applications are limited to a single computer, where computer means a group of processors that are connected to the same memory. Most modern computers provide this capability.

2.2.2 Parallel Distributed-Memory Applications

To overcome the limitations of a single computer, the more scalable architecture of a parallel distributed-memory machine was created. The basic unit of this architecture is a processing element (PE) comprised of one or more processors couple with memory. In other words, a PE is a shared-memory machine. The PEs are interconnected using some sort of networking technology. This architecture scales as well as the interconnect does. Common interconnect technologies in use today are Gigabit-Ethernet and InfiniBand. Clusters are parallel distributed-memory machines.

For a program to run on a parallel distributed-memory machine, it must be able to work with multiple thread of operation where communication between the threads goes over the interconnect. For the purposes of discussion, threads running on different PEs are called tasks.

The main task in designing a parallel program is figuring out how to split up the work between tasks. One method is to split up the processing between tasks. Each task would generally have the same data and perform variations of an operation on the data; for instance, running multiple scenarios. This methodology is called Task Parallel. An example of task parallelism is simulating the effects of various weather patterns on an urban environment. Each task would have a copy of the environment and run its variety of weather on it. The capability of executing each scenario is limited to the capabilities of a single processing element, but many scenarios can be executed at the same time.

Data Parallel methods split the data up between tasks an perform the same, or similar, operations on each piece of data. To calculate the effects of a weather pattern on an urban environment, the environment would first be divided between the tasks with each task responsible for one part of the environment. Each task would then calculate the effects of the weather on its section of the environment, communicating with the other PEs as needed to share information related to edge conditions, etc.

2.2.3 Message Passing Interface

The Message Passing Interface (MPI) standard has become the normal way of programming parallel distributed-memory applications. MPI works by starting processing tasks on each of the PEs alloted to the MPI process. These tasks are then able to send and receive messages between themselves allowing for inter-task communication.

As MPI defines a simple paradigm for inter-task communication, any parallelism in the application must be explicitly specified and programmed. Thus MPI programming is not simple or easy, though it is not without benefits.

MPI libraries are able to utilize advanced network layers such as InfiniBand without changes to the application, except for recompilation against the library. This design allows for advances in technology to be passed onto parallel application with little effort.

MPI is also able to take advantage of parallel filesystems such as Lustre[13]. Parallel filesystems spread file data between several file servers. Client programs are then able to access each part of the file in parallel, speeding file reading and writing while enabling each task to read or write a portion of data while not conflicting with the other tasks in the same MPI process.

MPI is generally deployed on clusters, making programs that are based on MPI able to run on a variety of machines.

2.2.4 Map Reduce

Map reduce is a functional programming idiom that has recently become popular due to Google's extensive use first described in a 2004 paper[3]. While Google's MapReduce environment is proprietary, the concepts and idea have passed into the opensource Hadoop[1] framework developed by the Apache Software Foundatation[**?**] with major support by Yahoo! and Cloudera[**?**].

Map reduce is relatively simple to program for. The programmer only needs to supply two functions: map and reduce. The map function takes as its input a record of the input data, record splitting is handled by the framework, and outputs zero or more key-value pairs. The reduce function takes a key and a set of values associated with that key and outputs zero or more key-value pairs.

Unlike MPI, parallelism is handled by the map-reduce framework which loads a portion of the entire input dataset on each of the machines participating processing elements, splits the data into records, executes the map function on each record, aggregates all the key-value pairs produced by the map functions and runs the reduce function on each unique key, passing the associated values along. At last the output from the mappers is collected into the final output of the program. Parallel operations are done implicitly, and therefore do not need to be created by the programmer.

Map reduce was designed on the assumptions that disk is cheap, networking is expensive, and that large systems can be built with inexpensive hardware, as long as no piece of hardware in indispensable. With these restrictions map reduce makes use of a distributed file system that replicates data between several of these inexpensive computers. Map reduce tasks are then ran on one of the computers that has the block of data it needs. The blocking used to spread files on the distributed file system are also used as the basis for parallelizing the map reduce process.

The map reduce process works in two phases, map and reduce. In the map phase the input data is split, usually automatically, into records. Each record is processed individually by the map function. The map function takes a record as input and outputs zero or more key-value pairs. The map process is inherently parallel and is executed where a copy of the data is (the program is moved to the data).

The resulting key-value pairs are aggregated by the map reduce framework by key and passed to the reduce function, which has access to all the key-value pairs associated with its key. The reduce function then outputs zero or more key-value pairs, which are the output of the entire map reduce job.

2.3 Parallel GIS Processing

Attempts have been made to overcome the limitations of single machine implementations. Of primary interest are those extending current methods to use multiple computers.

2.3.1 Problems with Desktop Programs on Clusters

Current desktop approaches to GIS processing such as ArcGIS are unable to make use of the multi-machine processing environment that compute clusters provide. While reworking these programs to utilize these extended resources is possible, it is non-trivial.

GRASS GIS was reworked[14] to use its collaboration features to distribute subqueries among computers. The method described in this paper utilizes multiple instances of GRASS in a master-slave configuration where all participants access a shared data repository or filesystem. The geometries are portioned between the various nodes. Operations are done on the subsets, and the results are merged to produce the final result.

While the method used to extend GRASS GIS will in fact speed up GIS processing, it has two flaws. First, GRASS is designed to be used in an interactive mode rather than

a batch or script driven approach. Second, the entire set of data used must still fit on a single computer to move it in or out of the environment.

2.3.2 Parallel Databases

Parallel databases such as TeraData[15] and Oracle[9] use data parallel methods. Paradise[16, 17] spreads data between computers using round-robin, hash, or spatial partitioning. Because the data is able to be distributed between multiple computers, the processing is able to scale to larger datasets.

When a query is processed across the database, a task is created for each fragment of the data. Thus as the data grows larger, the processing capabilities of the system also increase. If a particular processing operation requires relatively less computation for each data record this is fine. However, after the amount of computation per data record increases beyond a certain point, which is dependent on the speed of the machine used, the processing operation can be sped up by utilizing more processors. Major factors in determining how much data should be processed on each machine, and therefore how the data should be spread between machines, are memory, computation, and communication overhead to move the data and computation to another machine. The ratio of computation to memory and commutation requirements is often referred to as grain size. Coarser grained processes have more computation per data record, while finer grained processes have little computation for each data record.

Databases excel at working with indexed data while allowing multiple users to interact with the data in a concurrently safe manner through the use of atomic transactions. The requirements placed upon database systems to handle these situations slow down computations that don't utilize indexes or work on an entire dataset at once. The processing operations this research examines do not require these restrictions, and as such a more

efficient system can be created.

Many universities and research institutions already have significant investment in compute clusters. These clusters are groups of computer linked together with high speed network interconnects and high performance parallel filesystems such as Lustre[13]. By separating compute and storage resources at the cost of a high speed network, compute clusters are able to separate computation from data storage.

Parallel filesystems allow the data to be separated from the computer where the processing will be executed by spreading files across multiple network connected fileservers allowing access that can be faster than utilizing a computer's local disk for storage while also enabling processing to spread across the available compute resources based entirely on the process' grain size.

2.3.3 Problems with Current Parallel Approaches

While desktop GIS programs are relatively easy to use, their current implementations are fundamentally unable to make use of parallel computation resources. The attempt to use GRASS was not entirely successful due to its lack of a non-interactive mode and inability to work on datasets larger than could be handled by a single computer.

Parallel databases require dedicated resources and expertise to be useful. Even so, database are limited in their ability to work with a large range of process granularity as data redistribution is expensive in terms of rebuilding indexes and configuration changes. Furthermore optimizing SQL often requires knowledge of the data distribution and configuration of the database. Most companies that use databases as a core technology have dedicated staff to manage these systems.

Chapter 3 details the requirements of a good parallel approach to GIS processing.

CHAPTER 3 REQUIREMENTS

Chapter 2 related the current state of parallel GIS processing, from which the limitations of current approaches can be seen. From the capabilities and limitations of current approaches, the requirements of a good parallel GIS processing engine can be defined. Some of these requirements are derived from what makes good cluster based software, such as batch mode processing and scalability.

The main requirements of a parallel GIS processing engine are that is supports standard geospatial operations, can be executed in a batch environment, makes effective use of the additional resources provided by the cluster environment, and is not too hard to use.

3.1 Standard Geospatial Operations

Any GIS processing application must have at least a core set of GIS processing operations. Without the capability to perform the required processing, such an engine would be useless. The Open Geospatial Consortium[**?**] (OGC) defines a set of such operations in their Simple Features[2] standard.

The Open Geospatial Consortium, Inc. (OGC) is a non-profit, international, voluntary consensus standards organization that is leading the development of standards for geospatial and location based services[**?**]. The OGC maintains a variety of standards that support GIS processing such as cataloging, KML, WMS, and others.

The most important standard for this thesis is the Simple Features Standard[**?**] (SFS). The SFS defines ways of storing and processing data including defining the Well Known Text (WKT) and Well Known Binary (KWB) representations of geospatial data. Along with data formats, a set of operations for working with geospatial data are specified.

The OGC SFS is just a standard, and so it needs to be implemented. The main open-source library implementing the standard is the Java Topology Suite[**?**] (JTS). While the

JTS is implemented in Java, it has been ported to other languages. For example PostGIS uses the C/C++ GEOS library[**?**].

Compliance to this requirement can be tested by checking an independent implementation against the standard, or assumed thought the use of a compliant implimentation such as the JTS or GEOS libraries, assuming full access to the library functionality is allowed.

3.2 Batch Mode Processing

As many clusters only allow non-interactive, or batch, processing modes, the parallel GIS processing engine must fit this criteria.

To be compliant with this requirement, an implementation must support at least the standard geospatial operations while in batch mode.

3.3 Scalable

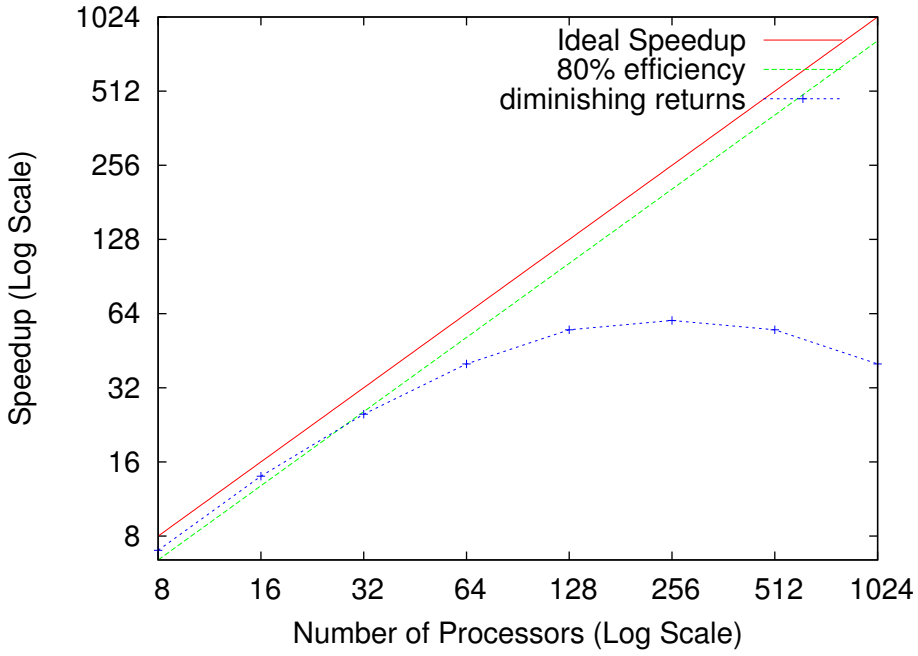
The goal of a scalable application is utilize the resources given it effectively. There are two aspects to scalability: capacity and speedup. Capacity scalability means that an implementation is able to solve a bigger problem than would otherwise be possible using a single machine. Speedup measures the impact of using more resources as compared to not using them.

Speedup is measured by comparing the processing time on a specific number of processors (n) to the processing time on a single processor:

$$\text{speedup}(n) = t(n)/t(1)$$

To see how scalable an application is, a speedup plot is generated which shows ideal

Figure 3.1: Example Speedup Graph



speedup for the application based on a serial run. Figure 3.3 shows a sample speedup graph with three speedup lines plotted. The Ideal Speedup line shows what the ideal speedup would look like on the graph enabling easy comparison of the other line(s). The 80% efficiency line shows an application that has achieved linear speedup. This application scales very well. The diminishing returns line shows an application that increases speedup to a point around 256 processors, at which time adding more resources actually increases the program runtime. This limitation could occur because of limited data size or some other concern.

Graphs like this will be used throughout this thesis to evaluate scalability.

3.4 Ease of Use

The last requirement is the most ambiguous: ease of use. The main thing to think of here is that most people doing GIS processing are interested in the results of the processing

and would like the method to obtain those results to be as easy as possible. In addition, most GIS processors are not computer scientists, and so if a programmable interface is the point of interaction, it must be as easy to use as possible so that the learning curve is not excessive.

Ease of use is a subjective measure. No user studies will be performed here, but it is a point of discussion that is important in evaluating a GIS processing engine implementation.

CHAPTER 4 DESIGN

This thesis implements and evaluates two approaches to a parallel GIS processing engine: hadoopGIS and clusterGIS. HadoopGIS is based on the Hadoop map-reduce framework. ClusterGIS uses MPI. This chapter discusses how the two implementations were designed to fulfill the requirements in Chapter 3. First design choices common to both implementations will be discussed, then the specifics of each implementation will be detailed.

How fulfill requirements

- Operation set (OGC-SFS)

- Batch Mode Processing

- Scalability Provided by:

– Dataset centric (no need for record level locking, etc.) A good cluster based GIS processing environment needs to be separate from a graphical user interface so that it does not incur additional overhead on the compute nodes and so that it can interact well with the generally batch-scheduled environment of a cluster. In addition, it must be able to distribute data between all the nodes used such that a single node does not become a limiting factor in the environment's scalability. A variety of data distribution models should also be available so that the data is distributed in a manner consistent with the required processing. Of course, the environment should be able to execute all the standard geospatial operations as defined by the Open Geospatial Consortium in their Simple Features[2] standard.

– Data parallel for speed/scaleup, different distribution models possible?

4.1 HadoopGIS

Uses JTS

Map/Reduce

Communication only possible in Reduce

Multiple Map/Reduce phases

How to use more than one dataset

Limited dataset distribution models

4.2 ClusterGIS

Uses GEOS

Split using MPI-IO

Communicate using MPI, possibility for different distributions

How a basic program works

How to use more than one dataset

Datasets could be distributed in any way imaginable.

CHAPTER 5 EXPERIMENTAL SETUP AND RESULTS

Chapter 3 defined the requirements for a good parallel GIS processing engine. Chapter 4 discussed the designs used in hadoopGIS and clusterGIS. Given these requirements and designs, the implementations can now be evaluated and compared against each other. As both these implementations are experimental, PostGIS will be used as a reference point to represent current methods. As such, it will also be evaluated as much as possible in the same manner as hadoopGIS and clusterGIS.

5.1 Standard Geospatial Operations

The first requirement is that an implementation can perform a set of standard geospatial operations. Allowing full access to a compliant geospatial library fulfills this requirement a priori. PostGIS uses, and in fact developed, the GEOS library. HadoopGIS makes use of JTS, from which the GEOS library was ported. ClusterGIS uses the GEOS library.

All three implementations fulfill this requirement, no further experimentation is needed.

5.2 Batch Mode Processing

Batch mode operation is essential to running operations on a cluster.

PostGIS extends the PostgreSQL database. PostgreSQL can be accessed through many different methods including programming interfaces in C and other languages, or through the provided command line client, “psql”. This client can be used interactively, or a sql script can be passed to it. That leaves the problem of the server. Either a dedicated server can be used, or a script can be written to create a server as needed to do the processing. This thesis sets a PostgreSQL server up as needed. PostGIS can perform batch mode processing.

Hadoop executes user jobs from a command line interface. The only hindrance for Hadoop to batch mode processing is the same as PostGIS: Hadoop is designed to have a persistent server environment setup. Projects like Hadoop on Demand allow a Hadoop environment to be created as needed. This thesis accomplishes a similar solution through a custom designed script. With this setup, HadoopGIS is capable of batch mode processing.

ClusterGIS only operates in a batch mode, there is no interactive mode.

5.3 Scalability

The first two requirements did not require any experimentation to evaluate. Scalability requires experimentation.

Verification of operations will be done by comparing output of each operation for HadoopGIS and ClusterGIS with the output from a PostGIS reference implementation.

As HadoopGIS and ClusterGIS are processing engines, scalability depends on the individual processing that is done. To achieve this, a set of processing operations is defined. After the operations are discussed in general, specific details for each implementation are discussed.

5.3.1 Operation set

One of the main tasks in parallelizing an algorithm is solving the problem of data access. HadoopGIS and clusterGIS split up the data that is being processed between different computers. The OGC SFS standard defines the operations that need to be supported. Table 5.1 shows a count of the number of operations in each section of the standard that defines operations along with the number of geometries required to execute that operation. One geometry is required for 42 of the operations, while 24 operations require two. There are no operations that require more than two geometries.

Section	1 Geometry	2 Geometries
6.1.2 (Geometry)	16	15
6.1.4 (Point)	4	0
6.1.6 (Curve)	5	0
6.1.7 (LineString, Line, LinearRing)	2	0
6.1.8 (MultiCurve)	2	0
6.1.10 (Surface)	3	0
6.1.11 (Polygon, Triangle)	3	0
6.1.12 (PolyhedralSurface)	4	0
6.1.13 (MultiSurface)	3	0
6.1.15 (Relational Operators)	0	9
Total	42	24

Table 5.1: OGC Methods by Number of Required Geometries

The problem faced by HadoopGIS and ClusterGIS then is to get the data required to perform the geospatial operations required by what ever processing is required. The operations defined here are meant to be representative of what actual processing requirements would demand. The first four operations deal with individual record access. The following three operations deal with getting the data needed for a geospatial operation where it is needed.

Specific details on the datasets follow the operation descriptions. For now all that is needed to know is that there are two datasets, one of 34 thousand employers and one of 1.2 million parcels of land.

5.3.1.1 Create

The create operation adds a new record to an existing dataset. Adding a record is the basic operation required to build a new dataset. Record-centric systems merely add a record to the record store, while dataset-centric systems output a new dataset that contains the data of the input dataset with the new record included in it. Expected output is the original dataset with the addition of a single record.

The create operation implementations add a parcel of land to the parcels dataset.

5.3.1.2 Read

The read operation extracts a single record from a dataset based on a unique identifier. Record-centric systems are able to employ indexes and other methods to quickly locate a record whereas dataset-centric systems must scan an entire dataset to produce the single record. Expected output is a single record.

The read operation extracts a parcel from the parcels dataset.

5.3.1.3 Update

The update operation finds a single record from a dataset and changes some attribute of the record. Record-centric systems are able to employ indexes to locate and modify the record. Dataset-centric systems generate a new dataset with the changed record. Expected output is a dataset that contains all the records from the input dataset with exception that the specified record has been changed in the specified manner.

The update operation changes the land use code for a parcel in the parcels dataset.

5.3.1.4 Destroy

The destroy operation finds and removes a record from a dataset. Record-centric systems are able to find the record to be removed and then removing it from the record store. Dataset-centric systems generate a new dataset without the specified record. Expected output is a dataset with all the records from the input dataset except for the one that was removed.

The destroy operation removes a parcel from the parcels dataset.

5.3.1.5 Filter

The filter operation removes all records that don't fulfill a certain requirement, in this case all the records that don't intersect with a defined geometry. Record-centric systems do not have much of an advantage here as indexing the results of some geospatial operation is not usually worth the indexing cost, so both the record-centric and dataset-centric systems must scan the entire dataset. Expected output is a dataset containing all of the records from the input dataset that fulfill the filtering requirements.

The filter operation removes all parcels of land that don't overlap a defined region in the parcels dataset.

5.3.1.6 Nearest

The nearest operation is derived from an actual processing operation required for the Digital Phoenix Project[11]. This operation uses two datasets. The first is a set of points representing employers in the Phoenix metro-area. The second is a set of polygons representing parcels of land in the same area. Digital Phoenix needed to match the employer with the parcel of land where the business should be. Thus the operation is for each employer, find the nearest parcel of land with a compatible zoning code. The datasets used in this thesis have been simplified to make this matching appear more straight forward. Expected output is a list of employer ids with associated parcel ids.

The nearest operation uses both the employers and parcels datasets.

5.3.1.7 Chaining

The chaining operation combines the filter and nearest operations with the intent of showing how multiple operations can be performed one after the other. This operation is an

obvious optimization of the nearest operation in that it first removes all residential parcels before running the nearest operation on the remaining parcels. As employers cannot (in this simplified world) exist on residential parcels, and since residential parcels make up a large portion of the entire parcel dataset, the number of distances calculated for each employer will be significantly decreased thus decreasing processing time. Expected output is the same as for the nearest operation.

The chaining operation uses both the employers and parcels datasets.

5.3.2 Dataset Description

Two datasets are used in evaluating these operations. The first is a set of employers where each record contains an employer id, the place where the employer is located represented as a point, and the business classification: commercial, industrial, or governmental. The employer dataset contains 34,302 records.

The second dataset is a set of parcels where each record contains a parcel id, a multipolygon representing the parcel coverage, and a land use code: residential, commercial, industrial, or governmental. The parcel dataset contains 1,218,130 records.

Both datasets use R, C, I, and G to represent residential, commercial, industrial, and governmental use codes. These datasets are simplified versions of real datasets for Maricopa County, Arizona. The datasets were simplified by adding the simplified land use code attribute and removing the other attributes not used in these operations.

5.4 Execution Environment

All operations will be executed on ASU's Saguaro cluster. The Saguaro cluster is comprised of several generations of hardware. To simplify comparisons, all operations will be executed on similar hardware with similar interconnects.

Description of Saguaro (common execute environment)

CentOS 5.3 based

Moab/Torque scheduler

GigE/Inifiband network

Lustre version, options

5.5 PostGIS Implementation

PostgreSQL 8.4

PostGIS 1.3....

Full or chunk of code per operation, description of how the operation was accomplished

All operations must be performed as if they were part of a series of operations.

5.5.1 Create

insert...

5.5.2 Read

select ... where id =

5.5.3 Update

update....

5.5.4 Destroy

delete ...

5.5.5 Filter

select...where

5.5.6 Nearest

Crazy set of queries (nested would have been nice, but too slow)

5.5.7 Chaining

Modify crazy set of queries to use the filter

5.6 HadoopGIS Implementation

Hadoop 0.20.0

HOD on Saguaro

JTS ...

JDK ...

Full or chunk of code per operation, description of how the operation was accomplished

All operations must be performed as if they were part of a series of operations.

5.6.1 Create

map: emit all record giving them the same key (forcing one reduce); reduce emits all received records with the addition of one

5.6.2 Read

map: emit only the record with the correct id; reduce: identity

5.6.3 Update

map: emit all records, updating the one with the correct id; reduce: identity

5.6.4 Destroy

map: emit all records except for the one with the correct id; reduce: identity

5.6.5 Filter

map: emit only records matching the requested criteria; reduce: identity

5.6.6 Nearest

map: loop through secondary dataset emitting id of primary and secondary; reduce: identity

5.6.7 Chaining

(Filter the datasets, then find the nearest in the filtered data)

filter secondary dataset on load; map: only search for nearest for filtered data, skip others; reduce: identity

5.7 ClusterGIS

ClusterGIS was compiled with the Intel C Compiler (icc) 10.1 20080312 against MVAPICH 1.0.1. MVAPICH makes use of the DDR InfiniBand network connection for communication. Dataset storage is done on Lustre, which is connected through the same InifiBand connections.

The specific implementation of each operation is discussed below. The main block of code is included and discussed. Full code listings are available in the appendix.

5.7.1 Create

The create operation is quite simple. First the dataset is loaded in a distributed manner across all the tasks included in the operation. One task then adds a record after which the dataset is written to disk.

The record addition code is as follows:

```
1 record = clusterGIS_Create_record_from_csv("97123897,POINT(0 0),C\n",
2           &start);
3 record->next = dataset->data;
4 dataset->data = record;
```

Lines 1-2 create the record using the clusterGIS_Create_record_from_csv

5.7.2 Read

emit only the record with the correct id;

5.7.3 Update

emit all records updating the correct one

5.7.4 Destroy

emit all record except the correct one

Figure 5.1: Speedup for clusterGIS create

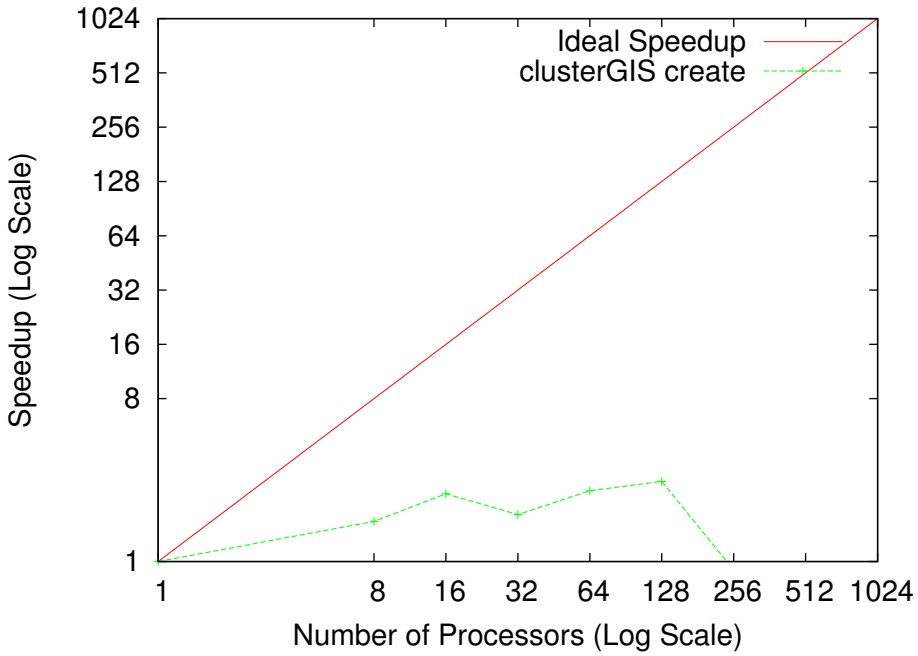


Figure 5.2: Speedup for clusterGIS read

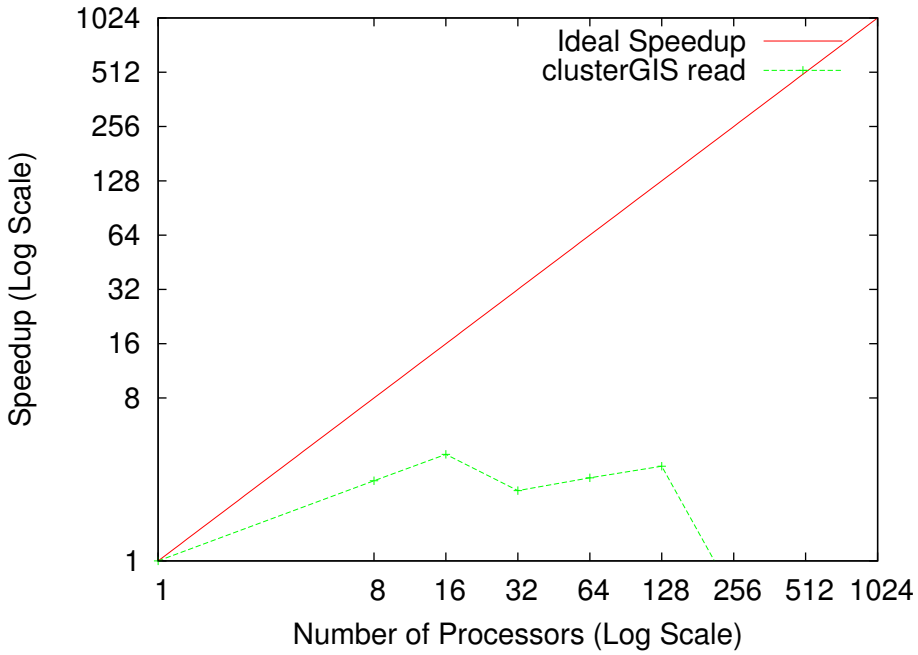


Figure 5.3: Speedup for clusterGIS update

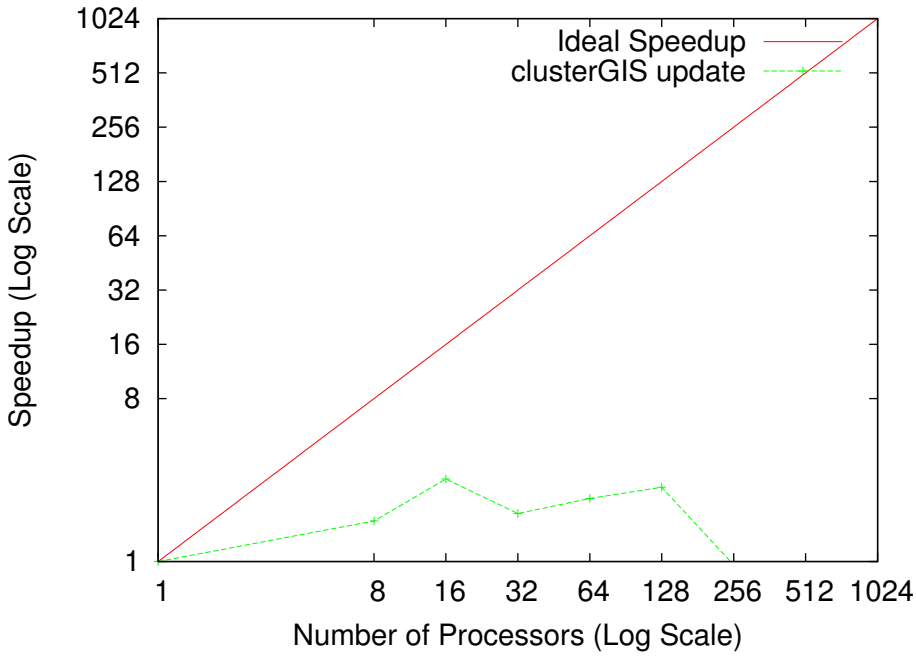


Figure 5.4: Speedup for clusterGIS delete

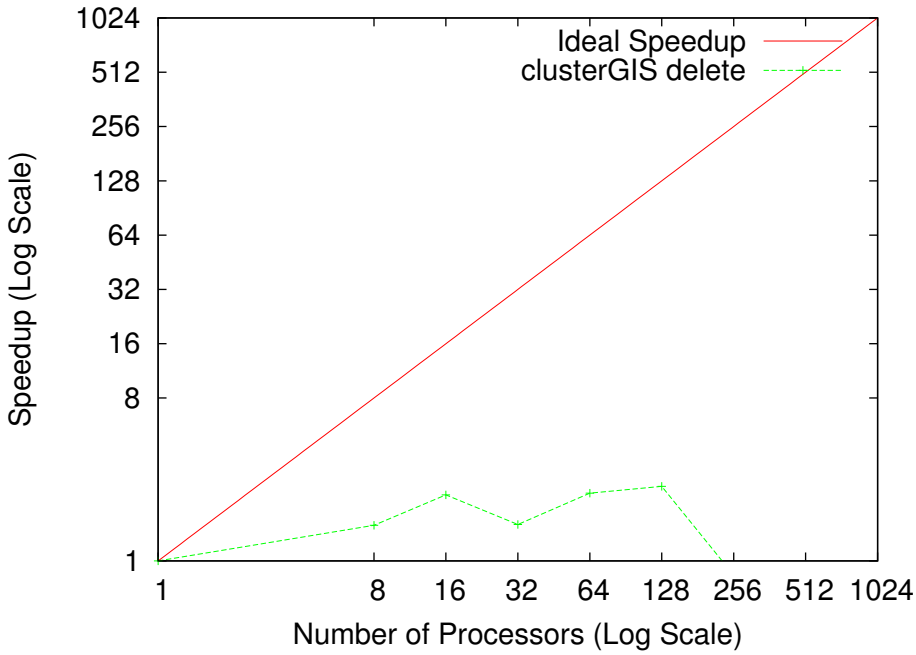
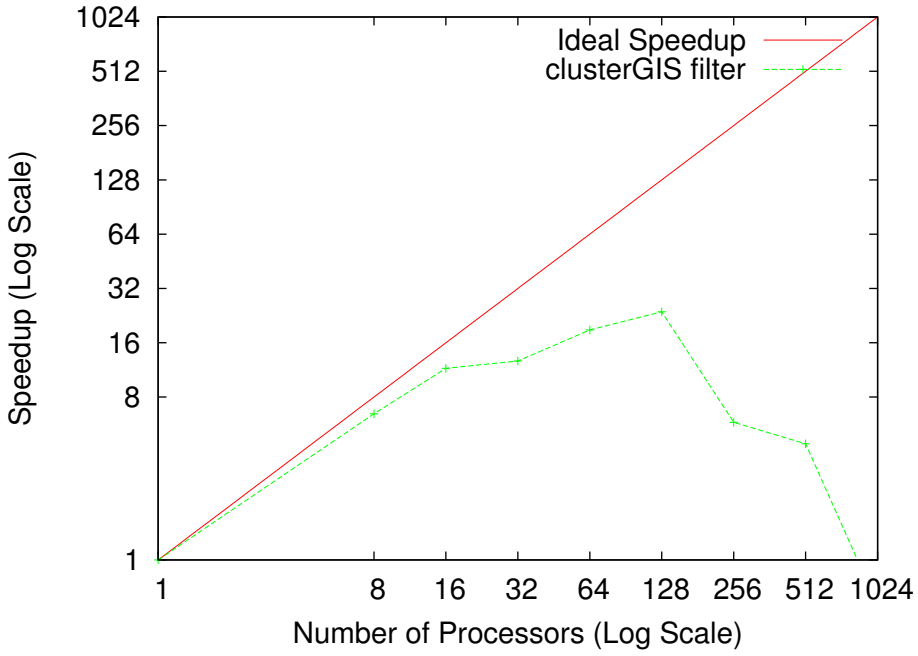


Figure 5.5: Speedup for clusterGIS filter



5.7.5 Filter

emit only records matching the requested criteria

5.7.6 Nearest

loop through primary and secondary dataset, emitting the id of primary and secondary which match

5.7.7 Chaining

(Filter the datasets, then find the nearest in the filtered data)

remove nodes in the datasets not matching the filter, find nearest as before

Figure 5.6: Speedup for clusterGIS nearest

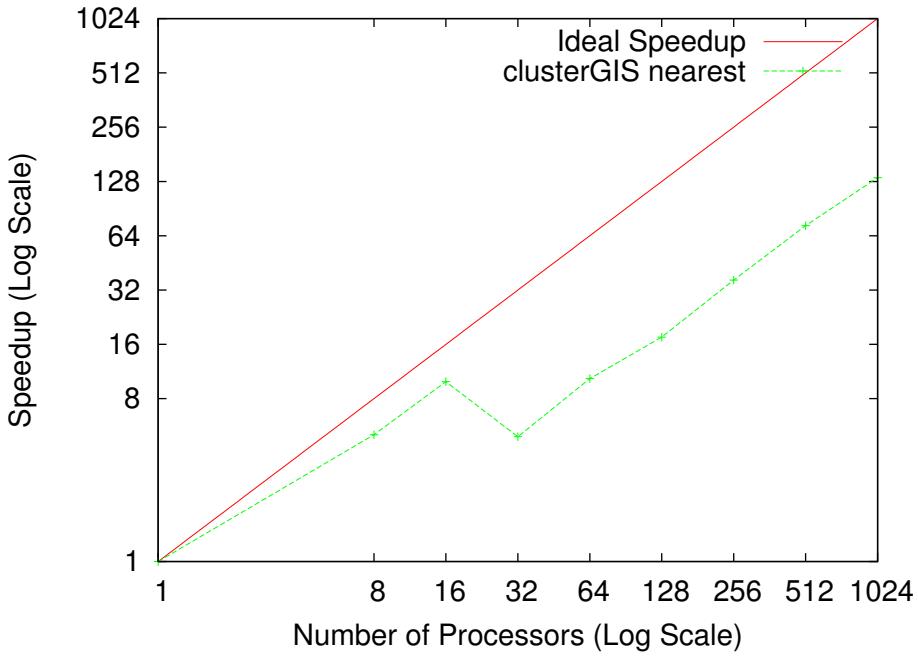
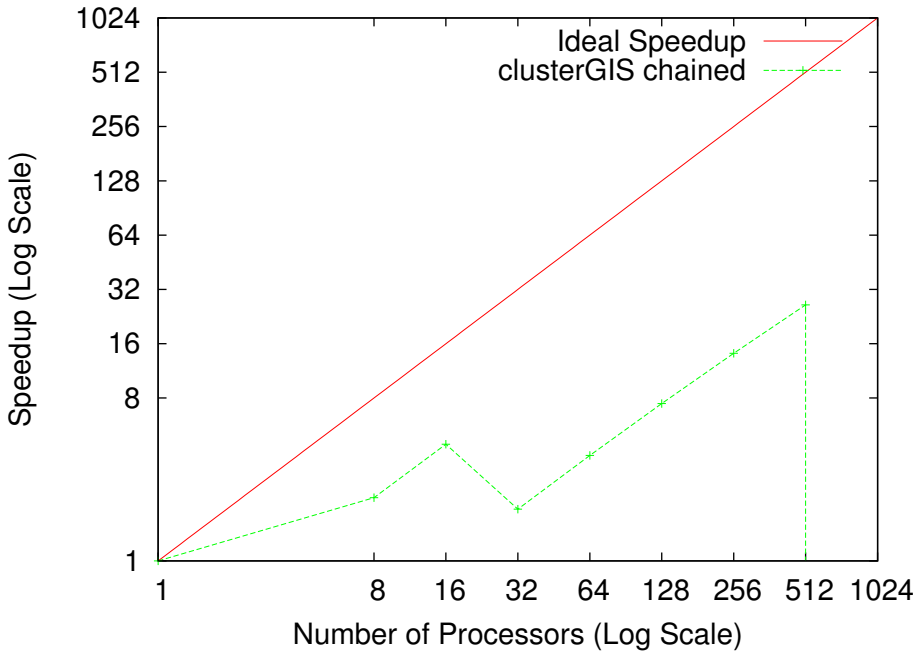


Figure 5.7: Speedup for clusterGIS chained



CHAPTER 6 RESULTS COMPARISON

6.1 Performance Analysis

vary procs, maintain data

vary data, maintain procs

vary data, vary procs

6.2 Comparisons

PostGIS as baseline

PostGIS to Hadoop

Hadoop to ClusterGIS

PostGIS to ClusterGIS

CHAPTER 7 CONCLUSION

By simplifying the requirements to handle process GIS operations in parallel, the cost to implementing parallel solutions will decrease, enabling more parallel processing methods to be created. This parallel methods will be able to take advantage of the increased processing powers made available through compute clusters.

Another benefit of this research is a classification of geospatial operations based on data access requirements and a sample set of queries to evaluate the efficiency of a parallel GIS processing implementation.

7.1 Future Work

additional decomposition methods, combined with alternative MPI communicators

addition of preprocessing methods (indexing, etc)

chunking of replicated dataset

CHAPTER Bibliography

[1] G. Mackey, S. Sehrish, J. Bent, J. Lopez, S. Habib, and J. Wang, "Introducing map-reduce to high end computing," in *Petascale Data Storage Workshop, 2008. PDSW '08. 3rd*, pp. 1–6, Nov. 2008.

[2] "Opengis implementation specification for geographic information - simple feature access - part 1: Common architecture," standard, Open Geospatial Consortium, Inc., 2006.

[3] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[4] T. Ormsby and R. Burke, *Getting to Know ArcGIS Desktop: Basics of ArcView, ArcEditor, and ArcInfo*. ESRI, Inc., 2nd ed., 2004.

[5] J. Gray, "Quantum gis: the open-source geographic information system," *Linux J.*, vol. 2008, no. 172, p. 8, 2008.

[6] M. Neteler and H. Mitasova, *Open source GIS: a grass GIS approach*. Springer, 3rd ed., 2002.

[7] R. West, *Understanding ArcSDE*. ESRI, Inc., 2001.

[8] D. Blasby, "Building a spatial database in postgresql," in *Open Source Database Summit*, 2001.

[9] S. Ravada and J. Sharma, "Oracle8i spatial: Experiences with extensible databases," in *SSD '99: Proceedings of the 6th International Symposium on Advances in Spatial Databases*, (London, UK), pp. 355–359, Springer-Verlag, 1999.

[10] R. Pahle and N. Kerr, "A datacentric framework for research in planning," in *UPE8, The 8th International Symposium (UPE 8) of the International Urban Planning and Environment Association*, 2009.

[11] S. Guhathakurta, Y. Kobayashi, M. Patel, J. Holston, T. Lant, J. Crittenden, K. Li, G. Konjevod, and K. Date, "Digital phoenix project: A multidimensional journey through time." Not published, 2006.

[12] P. Waddell, A. Borning, M. Noth, N. Freier, M. Becke, and G. Ulfarsson, "Microsimulation of urban development and location choices: Design and implementation of urbansim," *Networks and Spatial Economics*, vol. 3, pp. 43–67, 2003.

[13] P. J. Braam, "Lustre file system," white paper, Cluster File Systems, Inc., 2007.

[14] F. Huang, D. Liu, P. Liu, S. Wang, Y. Zeng, G. Li, W. Yu, J. Wang, L. Zhao, and L. Pang, "Research on cluster-based parallel gis with the example of parallelization on grass gis," in *Grid and Cooperative Computing, 2007. GCC 2007. Sixth International Conference on*, pp. 642–649, Aug. 2007.

[15] "Dbc/1012 data base computer concepts & facilities," tech. rep., Teradata Corp Document No C02-0001-00, 1983.

[16] J. Patel, J. Yu, N. Kabra, K. Tufte, B. Nag, J. Burger, N. Hall, K. Ramasamy, R. Lueder, C. Ellmann, J. Kupsch, S. Guo, J. Larson, D. De Witt, and J. Naughton, "Building a scaleable geo-spatial dbms: technology, implementation, and evaluation," in *SIGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 336–347, ACM, 1997.

[17] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu, "Client-server paradise," in *VLDB '94: Proceedings of the 20th International Conference on Very Large Data Bases*, (San Francisco, CA, USA), pp. 558–569, Morgan Kaufmann Publishers Inc., 1994.