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

Geospatial Information Systems were designed to model the world around us. As our abilities to gather data and analyze it in more sophisticated ways have improved we have had larger datasets with more computationally intensive methods for analysis. Even with the increased power of desktop computers, desktop GIS processing applications are unable to perform the desired analysis.

This thesis applies the parallel processing paradigms of map reduce and message passing to GIS processing, creating two implementations of parallel GIS processing environments, one using the open source Hadoop map-reduce framework and the other using MPI. Both implementations are evaluated for scalability and usability.

TABLE OF CONTENTS

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 computers.

Programming for multiple machines is not the easiest of tasks. A common paradigm is Single Program, Multiple Data[1] (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[2] 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[3] 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[4]. 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[5] and the GEOS[6] 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 Cluster-GIS. 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)[7] have been in use since the 1960's with the Canada Geographic Information System (CGIS)[8] and then moving from the mainframes to current desktop applications like ESRI's ArcGIS[9] 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[9], QuantumGIS[10], and GRASS GIS[11] 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[12] or PostGIS[13] 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[13], ArcSDE[12], or Oracle Spatial[14]. 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[15] (USF) designed by the Digital Phoenix[16] 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[17] is a popular simulation tool for growth models.

GeoDa[18] is a spatial analysis tool that includes spatial regressions. PySal[19] 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[3] 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[20] (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[21] (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[22]. 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[4]. While Google's MapReduce environment is proprietary, the concepts and idea have passed into the open source Hadoop[2] 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[23] 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[24] and Oracle[14] use data parallel methods. Paradise[25, 26] 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[22]. 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[3] 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 service. 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[3] (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[20] (JTS). While the

JTS is implemented in Java, it has been ported to other languages. For example PostGIS uses the C/C++ GEOS library[6].

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

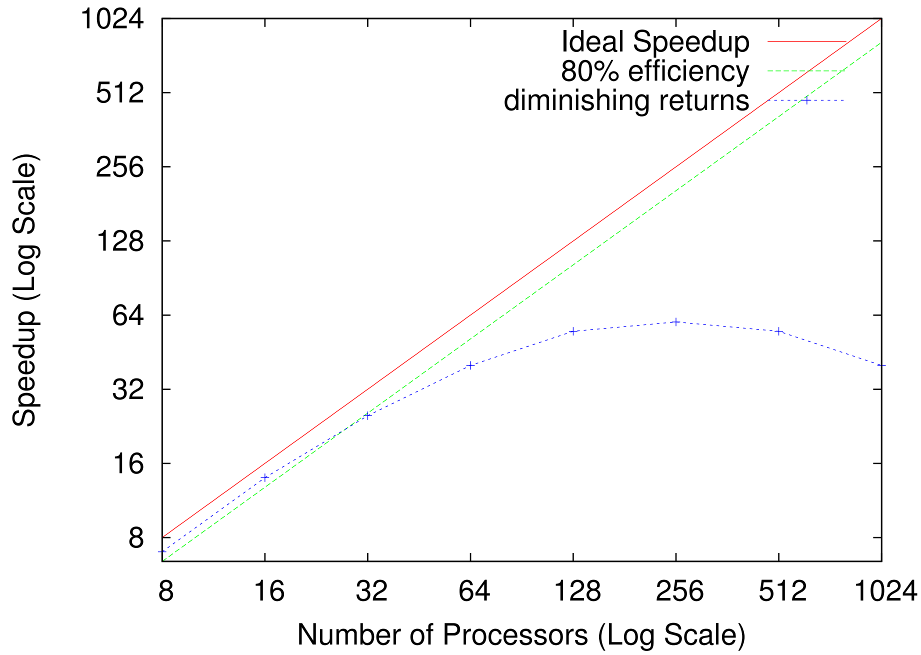
The goal of a scalable application is utilize the resources given it effectively. There are two aspects to scalability: capacity and speedup. Capacity scalability means that an implementation is able to solve a bigger problem than would otherwise be possible using a single machine. Speedup measures the 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

To discover the differences between using message passing and map reduce as methods for parallel GIS processing, this thesis implements two environments for manipulating GIS data. The first approach, hadoopGIS, is based the Hadoop map-reduce framework. The second, clusterGIS, uses MPI. Both implementations build on these existing technologies and utilize standards compliant geospatial libraries. The main work is combining the underlying parallel technology with geospatial processing capabilities in a way that works for that paradigm.

One point of simplification made for both implementations is supporting only one file format. Both hadoopGIS and clusterGIS use CSV formatted data with geospatial data stored in the Well Known Text (WKT)[3] format. HadoopGIS adds an additional file which stores the column names, one on each line. ClusterGIS just uses column numbers.

4.1 HadoopGIS

Hadoop[2] is an open source map-reduce framework developed in Java. The Java Topology Suite (JTS)[20] is used for geospatial processing.

Map reduce is basically a two phase solution for parallel computing. The first phase applies a function, map, to each record in the input dataset. This phase is inherently parallel as each call to map is completely independent, requiring no more data than just the record that only it is responsible for. The second phase, reduce, is inherently not parallel. Some parallelization is given by Hadoop in this phase by limiting a reduce function from seeing all the data produced by the map function to only the data produced with the same key (each output being in the form of a key-value pair). Because of this limitation, reduce has some parallelism, but much much less than in map.

Hadoop itself is comprised of two major parts: a distributed filesystem, and a parallel execution engine. The filesystem, Hadoop Distributed Filesystem (HDFS), splits files into blocks which are then spread among the participating computers. Blocks can be automatically replicated, such that the loss of any participating computer will not cause data loss. The execution engine is friendly with HDFS, in that it knows where file blocks, and there replicas, are located and can place computation on the same computer that contains the data. The idea here is that data is larger than the program and that therefore moving the program to the data will improve efficiency. Hadoop may also start additional computation tasks that replication the computation being done elsewhere on one of the replicated blocks.

Basic Hadoop programs require three parts. The first part is some startup code that tells Hadoop which input files to use, and which map and reduce functions to execute. The Hadoop environment will look at the inputs and decide on the number of mappers to create. Each mapper is responsible for channeling the data from a file block to the map function, and then taking that output and passing it onto the reducers. The number of mappers used is based on two criteria. Each file will get a mapper. If a file consists of more than one block of data, a mapper will be created for each additional block. This process is meant to be done automatically, but has real-world effects on program performance.

The main idea on integrating GIS processing into Hadoop was to create a GIS datatype that could pass through the map and reduce phases like any other Hadoop datatype. In practical terms there are three boundaries where more work was required than just create a plain old java object.

The first gap to overcome is getting data into the mappers. Hadoop uses two classes in addition to the core GIS class to load data from HDFS blocks. The process starts with GISInputFormat which uses GISRecordReader. GISRecordReader is responsible for extracting the records from a file block. In this case, GISRecordReader wraps the

Hadoop supplied LineRecordReader, which manages to solve the problem of connecting lines in a file split between blocks. In this phase the geospatial data is converted from WKT to a Geometry object provided by the Java Topology Suite.

To bridge the gap between mappers and reducers, Hadoop created an interface called Writable. The GIS class implements writable by providing functions that serialize the object state into a byte stream, and reconstitute an object from that byte stream.

The last gap to cross is taking data from the reducers and putting it back on HDFS. The process here is nearly a mirror image of loading data, except that the data is being written. The process starts with GISOutputFormat which utilizes GISRecordWriter. GIS-RecordWrite converts the GIS object back into CSV format and sends the resulting byte stream through the supplied DataOutputStream.

This design builds upon the basic architecture of Hadoop and adds as little as possible to get a GIS datatype to work in the environment. The user of this new functionality has access to the full power of both Hadoop's task and data management which allows for parallel processing, and the capabilities of a JTS geometry object which implements all the standard geospatial operations. All of this is done for the user by them stating that the input data should use the GISInputFormat and outputted data uses the GISOutputFormat.

This enhancement does not solve one of Hadoop's problems. That is the use of a secondary dataset. The map function only is provided with one record. The framework expects most computation that requires access to more than just one record to be accomplished in the reduce phase. There is provision, however, to load extra data when a mapper is created. These limitation will be discussed further in the experimental setup and results section of this thesis.

4.2 ClusterGIS

ClusterGIS is library that handles the basic functionality required to do GIS processing using MPI. ClusterGIS utilized the GEOS library for geospatial functionality.

MPI is based on the idea groups of cooperating tasks can collaborate on some computation by passing messages among themselves. These messages allow for coordination and communication. ClusterGIS builds on these basic capabilities

The first problem clusterGIS solves is distributing the records in a dataset among the participating tasks. The end

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. This section details the experiments and specific implementation details. Chapter 6 presents and discusses the results of these experiments.

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

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

are no operations that require more than two 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[16]. 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 network interconnects, using the one most natural for use.

Saguaro is a CentOS 5.3 based cluster running on Intel Xeon processors. Each node used in these experiments has two harpertown processors each with four cores running at 2.83GHz and has 16GB of RAM. In addition each node has one gigabit ethernet connection and one DDR InfiniBand connection. The shared filesystem for these experiments is Luster 1.6.6 accessed through the InifiniBand connection.

5.5 PostGIS Implementation

The PostGIS experiments were ran using PostgreSQL 8.4.1 and PostGIS 1.3. Both pieces of software were compiled for these tests and used default options.

Each of the following operations was executed on a freshly setup PostGIS instance, running on local disk with default options. Data was loaded, indexed, and vacuumed before the experiment was executed. Runtimes were gathered by using the linux ‘time’ command.

Full SQL statements for each operation are listed below.

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

The hadoopGIS experiments were executed using Hadoop 0.19.0, data was accessed on an HDFS running on the same nodes off local disk. Internode communication was accomplished through Gigabit Ethernet.

Default values were mostly used. Changes were made to increase the number of mappers per node to 8, increasing the memory available to each JVM, and altering the HDFS block size to produce the desired number of mappers.

Core algorithm details are described below. Full listings are available in the appendix.

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 Implementation

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:

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

5.7.5 Filter

emit only records matching the requested criteria

5.7.6 Nearest

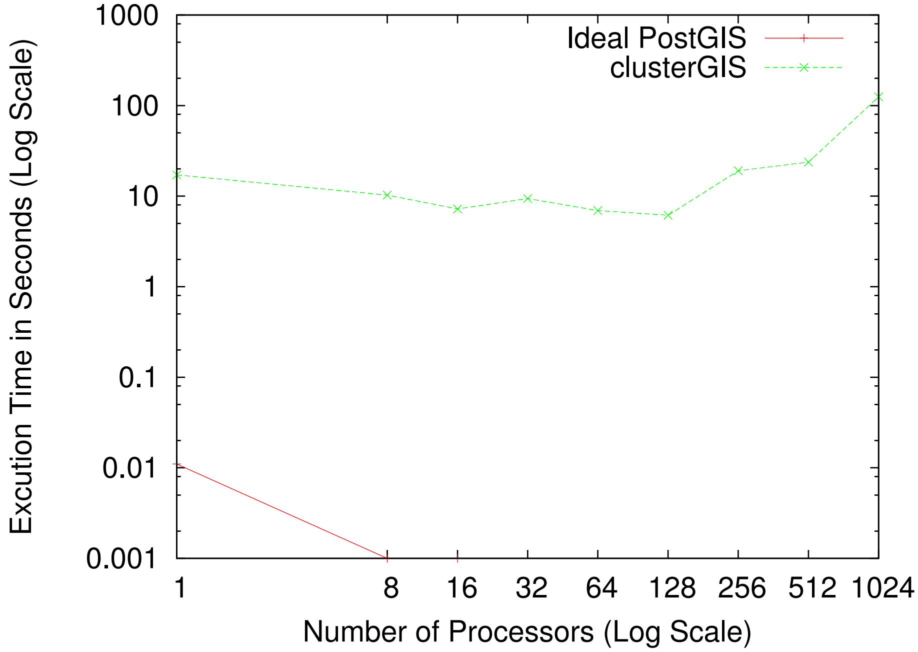
loop through primary and secondary dataset, emitting the id of primary and secondary which match

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 6.1: Create Execution Time



PostGIS data will be expanded as though it had ideal speedup

Figure 6.2: Create Speedup

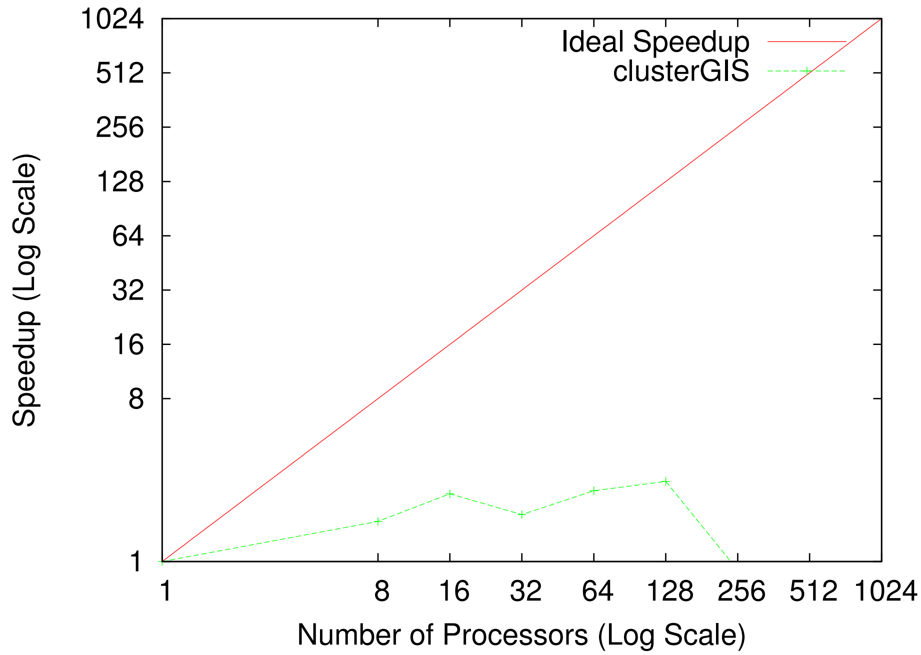


Figure 6.3: Read Speedup

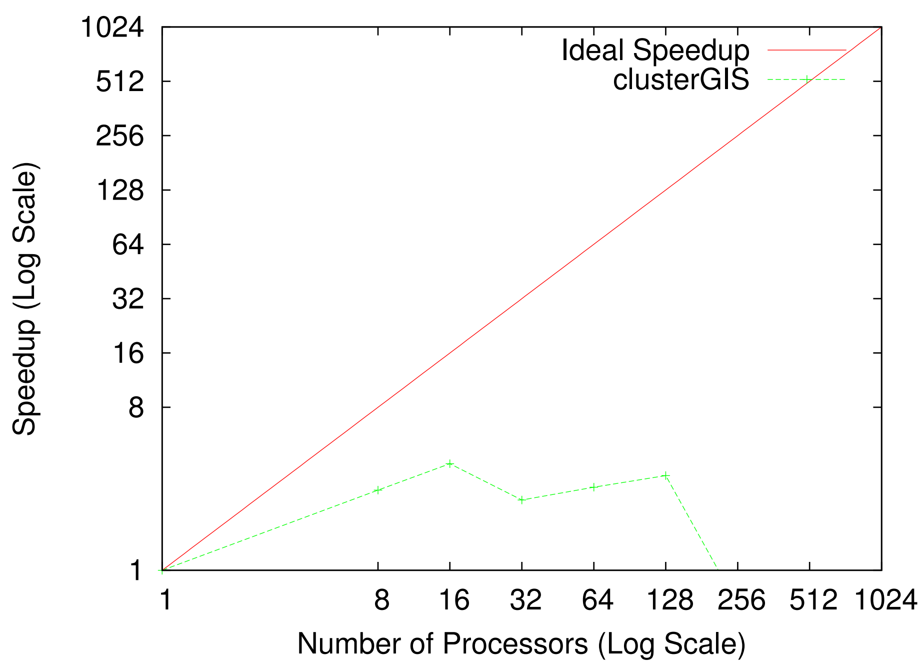


Figure 6.4: Update Speedup

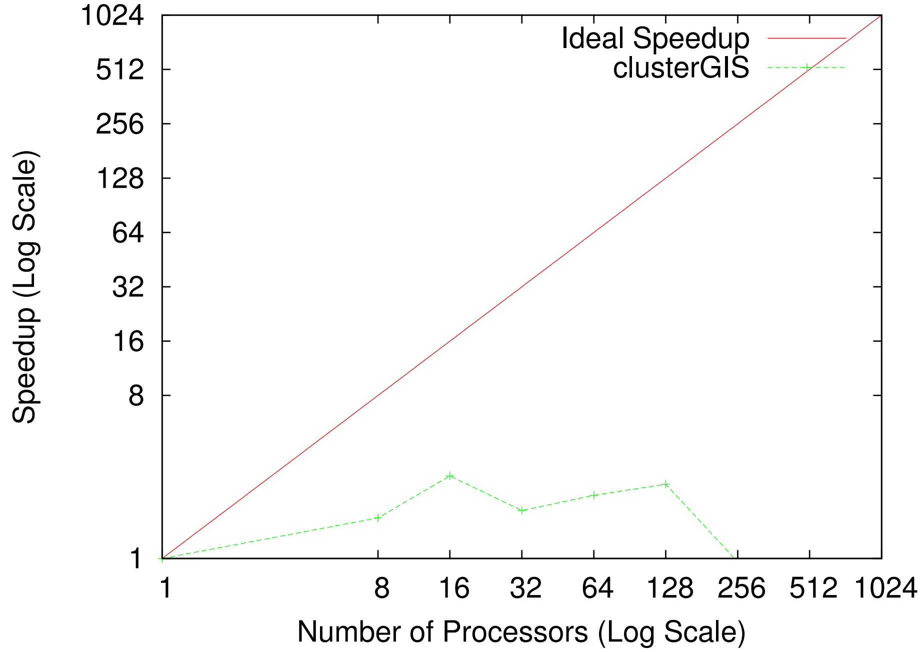


Figure 6.5: Delete Speedup

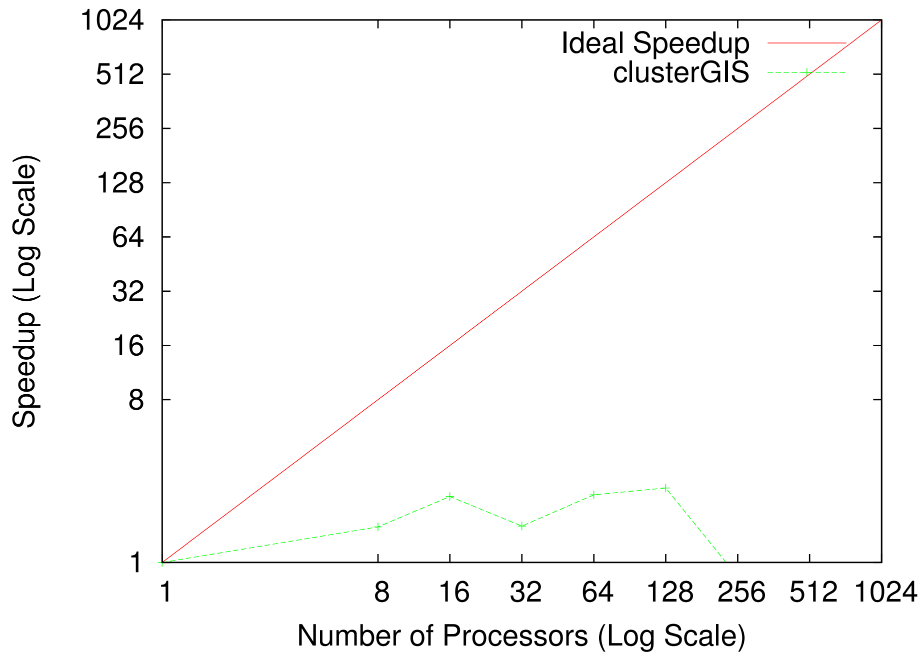


Figure 6.6: Filter Speedup

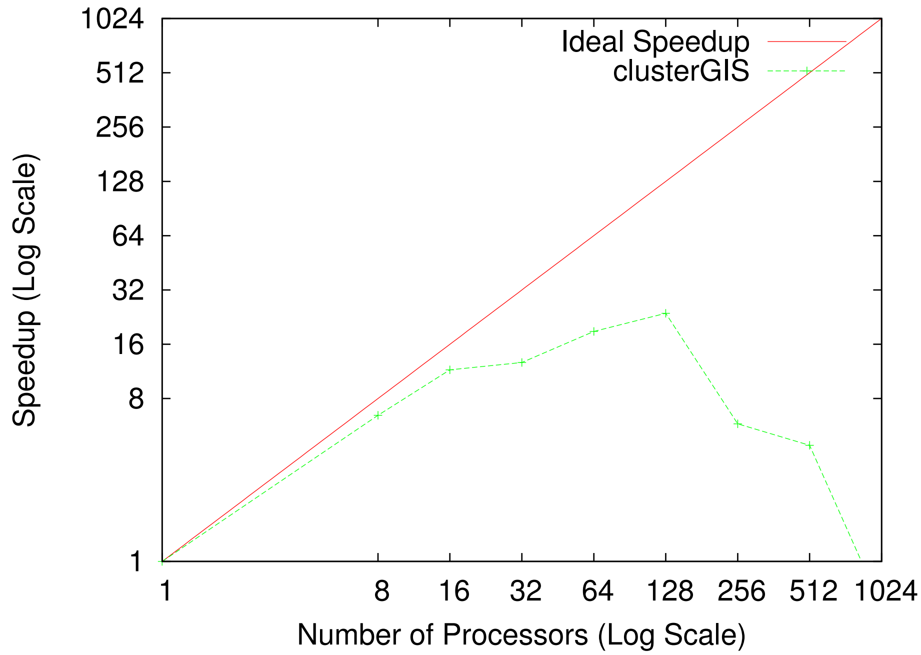
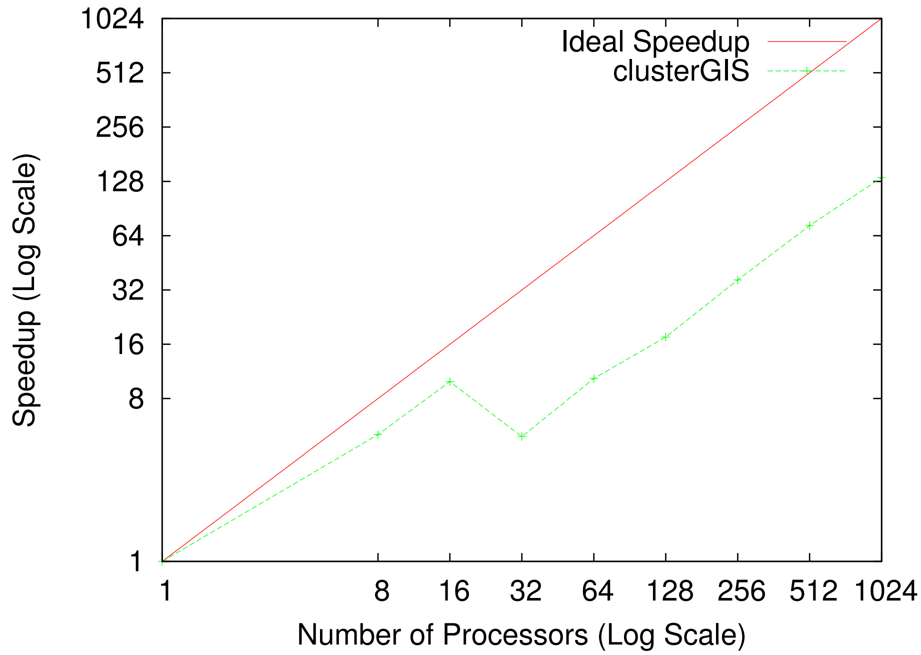


Figure 6.7: Nearest Speedup



6.1 Record Operations

6.1.1 Create

6.1.2 Read

6.1.3 Update

6.1.4 Delete

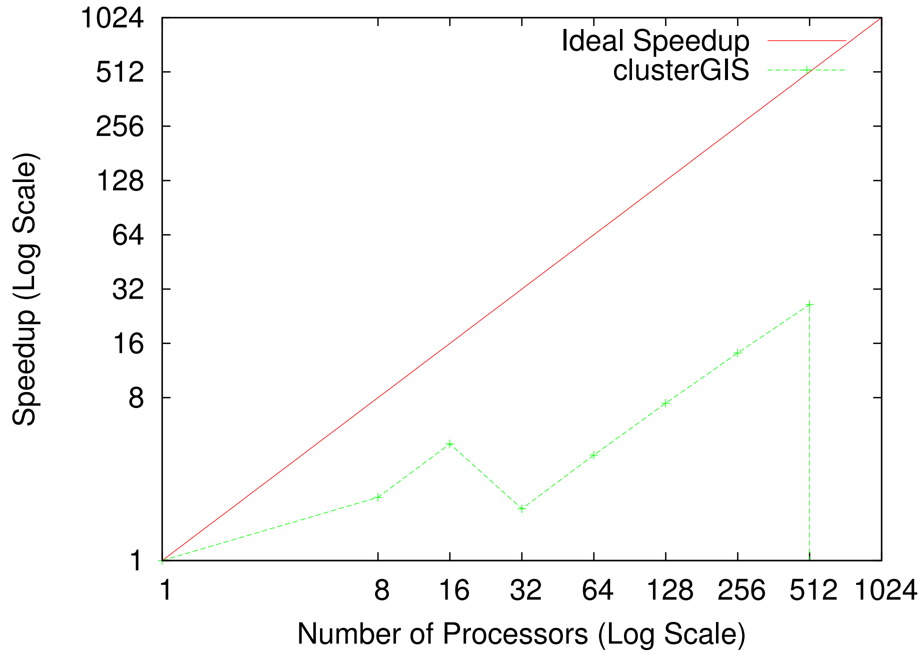
6.2 Dataset Operations

6.2.1 Filter

6.2.2 Nearest

6.2.3 Chained

Figure 6.8: Chained Speedup



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] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

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

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

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

[5] M. P. I. Forum, "Mpi: A message-passing interface standard," *International Journal of Supercomputer Applications*, 1994.

[6] O. S. G. Foundataion, "Geos - geometry engine, open source," *http://trac.osgeo.org/geos/*, 2009.

[7] M. D. Michael Worboys, *GIS: a computing perspective*. CRC PRess, 2004.

[8] P. Schut, "Back from the brink: the story of the remarkable resurrection of the canada land inventory data," *http://www.igs.net/ schut/cli.html*, 2000.

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

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

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

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Y. K. Luc Anselin, Ibnu Syabri, "Geoda: An introduction to spatial data," *Geographical Analysis*, vol. 38, pp. 5–22, 2006.

[19] L. A. Sergio J. Rey, "Pysal: A python library of spatial analytical methods," *The Review of Regional Studies*, vol. 37, no. 1, pp. 5–27, 2007.

[20] V. Solutions, "Java topology suite," *http://www.vividsolutions.com/jts/jtshome.htm*, 2009.

[21] D. Guidi, "Nettopologysuite," *http://code.google.com/p/nettopologysuite/*, 2009.

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

[23] 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.

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

[25] 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.

[26] 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.

APPENDIX A HADOOPGIS SOURCE

A.1 Core

A.1.1 GIS.java

```
package hadoopGIS;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

import java.util.ArrayList;
import org.apache.hadoop.io.BinaryComparable;
import com.vividsolutions.jts.geom.Coordinate;
import java.io.DataInput;
import java.io.DataOutput;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.GeometryFactory;
import java.util.HashMap;
import java.io.IOException;
import java.util.List;
import java.util.Map;
import java.util.Iterator;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;

public class GIS extends BinaryComparable implements
   Writable
{

        public Geometry geometry;
        public HashMap<String, String> attributes;
        private List<String> columns;

        public GIS() {
                geometry = new GeometryFactory().
                   createPoint(new Coordinate(0,0));
                attributes = new HashMap<String, String
```

```
            >(32);
        columns = (List<String >) new ArrayList<
            String >();
}

public Text toText()
{
        return new Text(hashToString(attributes) +
            "\n");
}

// For BinaryComparable
public byte[] getBytes()
{
        return toText().getBytes();
}

public int getLength()
{
        return toText().getLength();
}

public boolean update(Text value, List<String>
    columnList)
{
        columns.clear ();
        attributes.clear ();
        String[] splits = value.toString().split
            ("\",\"");

        columns.addAll (columnList);
        for (int i=0; i < splits.length; i++)
        {
                // Erase begining and ending quotes
                    /commas
                splits[i] = splits[i].replaceAll
                    ("^\"","");
                splits[i] = splits[i].replaceAll
                    ("\"$","");

                attributes.put(columns.get(i),
                    splits[i]);
```

```
                if (columnList.size () == 0)
                        columns.add (String.valueOf
                            (i));
        }

        try {
                geometry = new WKTReader().read(
                   new String ((String) attributes.
                   get("the_geom")));
        }
        catch (com.vividsolutions.jts.io.
           ParseException e) { }

        return true;
}

public String toString()
{
        StringBuilder finalString = new
           StringBuilder ();
        for (int i=0; i<columns.size (); i++)
        {
                finalString.append ("\"");
                finalString.append (attributes.get
                   (columns.get (i)));
                finalString.append ("\"");
                if (i != columns.size ()-1)
                        finalString.append(",");
        }

        //finalString.append ("\n");
        return finalString.toString ();
}

public void write(DataOutput out) throws
   IOException {
        Text value = toText();
        value.write(out);
}

public void readFields(DataInput in) throws
```

```
    IOException {
        Text value = new Text();
        value.readFields(in);
        attributes.putAll (stringToHash (value.
            toString()));
        try {
                geometry = new WKTReader().read(
                    new String ((String) attributes.
                    get("the_geom")));
        }
        catch (com.vividsolutions.jts.io.
            ParseException e) { }
}

// Outputs "key1"="value1","key2"="value2" after
    escaping \",= characters
private String hashToString(HashMap<String,String>
    h)
{
        h.put ("the_geom", geometry.toText ());
        String key, value, finalString = "";

        for (int i=0; i<columns.size (); i++)
        {
                // Escape string
                key = columns.get (i);
                value = attributes.get(key);

                key = key.replace("^\"","");
                value = value.replace("\"$","");

                key = key.replace("\\", "\\\\");
                key = key.replace("\"", "\\\"");
                key = key.replace("=", "\\=");
                key = key.replace(",", "\\,");

                value = value.replace("\\", "\\\\")
                    ;
                value = value.replace("\"", "\\\"")
                    ;
                value = value.replace("=", "\\=");
                value = value.replace(",", "\\,");
```

```
                finalString += "\"" + key + "\"=\""
                    + value + "\"";
                if(i != columns.size ()-1)
                        finalString += ",";
        }

        return finalString;
}

private HashMap<String,String> stringToHash(String
    str)
{
        if(str == null)
                return null;

        columns.clear ();
        str = str.substring (0, str.length ()-1);
        HashMap<String,String> h = new HashMap<
            String,String>();
        String [] splits = str.split("\",\"");
        for(int i=0; i<splits.length; i++)
        {
                String [] pair = splits [i].split
                    ("\"=\"");
                if(pair.length != 2)
                {
                        h.put (pair [0], "");
                        columns.add (pair [0]);
                        continue;
                }

                // Unescape string
                pair[0] = pair[0].replace("\\\\",
                    "\\");
                pair[0] = pair[0].replace("\\\"",
                    "\"");
                pair[0] = pair[0].replace("\\=",
                    "=");
                pair[0] = pair[0].replace("\\,",
                    ",");
```

```
                    pair[1] = pair[1].replace("\\\\",
                       "\\");
                    pair[1] = pair[1].replace("\\\"",
                       "\"");
                    pair[1] = pair[1].replace("\\=",
                       "=");
                    pair[1] = pair[1].replace("\\,",
                       ",");

                    pair[1] = pair[1].replace("\"","");
                    pair[0] = pair[0].replace("\"","");

                    h.put(pair [0], pair [1]);
                    columns.add (pair [0]);
            }

            return h;
        }
}
```

A.1.2 GISInputFormat.java

```
package hadoopGIS;

import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileSplit;
import hadoopGIS.GISRecordReader;
import org.apache.hadoop.mapred.InputSplit;
import java.io.IOException;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reporter;

public class GISInputFormat extends FileInputFormat<
   LongWritable, GIS> {

        // From FileInputFormat
        public RecordReader<LongWritable, GIS>
           getRecordReader(InputSplit split, JobConf job,
           Reporter reporter) throws IOException
        {
```

```
                return new GISRecordReader(job, (FileSplit)
                    split);
        }

}
```

A.1.3 GISRecordReader.java

```
package hadoopGIS;

import java.util.List;
import java.util.ArrayList;
import java.io.BufferedReader;
import java.io.FileReader;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileSplit;
import hadoopGIS.GIS;
import java.io.IOException;
import org.apache.hadoop.mapred.LineRecordReader;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.io.Text;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.filecache.DistributedCache;

class GISRecordReader implements RecordReader<LongWritable,
    GIS>
{
        private List<String> columnList;
        private LineRecordReader reader;

        public GISRecordReader(Configuration job, FileSplit
            split) throws IOException
        {
                columnList = new ArrayList<String> ();
                reader = new LineRecordReader(job, (
                    FileSplit) split);

                String columnFilename = job.get ("
                    columnNames");
```

```
        Path[] distCacheFiles = new Path[0];
        try { distCacheFiles = DistributedCache.
           getLocalCacheFiles(job); }
        catch (IOException e)   { return; }

        String line = new String ();
        for (int i=0; i<distCacheFiles.length; i++)
        {
                if (distCacheFiles [i].getName ().
                   equals (columnFilename))
                {
                        BufferedReader reader = new
                            BufferedReader(new
                           FileReader(
                           distCacheFiles [i].
                           toString ()));
                        while ((line = reader.
                           readLine()) != null)
                                columnList.add (
                                   line);

                        break;
                }
        }
}

public float getProgress()
{
        return reader.getProgress();
}

public synchronized void close() throws IOException
{
        reader.close();
}

public  synchronized long getPos() throws
   IOException
{
        return reader.getPos();
}
```

```
        public GIS createValue()
        {
                return new GIS();
        }

        public LongWritable createKey() {
                return new LongWritable();
        }

        public synchronized boolean next(LongWritable key,
           GIS value) throws IOException
        {
                Text textValue = new Text();
                if (reader.next(key, textValue)) {
                        value.update(textValue, columnList)
                           ;
                        key.set(new Long(value.attributes.
                           get("id")));
                        return true;
                }
                return false;
        }
}
```

A.1.4 GISOutputFormat.java

```
package hadoopGIS;

import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FSDataOutputStream;
import hadoopGIS.GIS;
import java.io.IOException;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.Progressable;
import org.apache.hadoop.mapred.RecordWriter;

public class GISOutputFormat extends FileOutputFormat<
   LongWritable, GIS>
{
```

```
        public RecordWriter<LongWritable, GIS>
           getRecordWriter(FileSystem ignored, JobConf job,
            String name, Progressable progress) throws
           IOException
        {
                Path file = getTaskOutputPath(job, name);
                FileSystem fs = file.getFileSystem(job);
                FSDataOutputStream fileOut = fs.create(file
                   , progress);
                return new GISRecordWriter<LongWritable,
                   GIS>(fileOut);
        }

}
```

A.1.5 GISRecordWriter.java

```
package hadoopGIS;

import java.io.DataOutputStream;
import hadoopGIS.GIS;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapred.RecordWriter;
import org.apache.hadoop.mapred.Reporter;
import java.io.UnsupportedEncodingException;

public class GISRecordWriter<LongWritable, GIS> implements
   RecordWriter<LongWritable, GIS>
{
        DataOutputStream out;

        public GISRecordWriter(DataOutputStream out) {
                this.out = out;
        }

        public synchronized void write(LongWritable key,
           GIS value) throws IOException {
                out.writeBytes(value.toString());
                out.writeBytes("\n");
```

```
        }

        public synchronized void close(Reporter reporter)
          throws IOException {
                out.close();
        }
}
```

A.2 Operations

A.2.1 Create

```
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.io.Text;

import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;

import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
```

```
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;

public class create extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, GIS>, Reducer<
   LongWritable, GIS, LongWritable, GIS>
{
        // For Mapper interface
        public void map(LongWritable key, GIS value,
           OutputCollector<LongWritable, GIS> output,
           Reporter reporter) throws IOException
        {
                output.collect(key, value);
        }

        // For Reducer interface
        public void reduce(LongWritable key, Iterator<GIS>
           values, OutputCollector<LongWritable, GIS>
           output, Reporter reporter)
        {
                while(values.hasNext()) {
                        try {
                                output.collect(key, values.
                                   next());
                        } catch (IOException e) {}
                }
        }

        // For Mapper (via JobConfigurable) interface
        public void configure(JobConf job)
        {
        }

        // For Mapper (via Closeable) interface
        public void close() {}
```

```
// For Tool interface
public int run(String[] args) throws Exception
{
        JobConf job = new JobConf(new Configuration
           (), this.getClass());

        GISInputFormat.setInputPaths(job, new Path
           ("/user/alaster/gis/parcels.gis"));
        Path p = new Path ("/user/alaster/gis/
           parcels.names");
        DistributedCache.addCacheFile (p.toUri (),
           job);
        job.set ("columnNames", p.getName ());

        GISOutputFormat.setOutputPath(job, new Path
           ("output"));

        job.setJobName("hadoopGIS.examples.create")
           ;

        job.setMapperClass(this.getClass());
        //job.setCombinerClass(this.getClass());
        job.setReducerClass(this.getClass());

        job.setInputFormat(GISInputFormat.class);
        //job.setOutputFormat(TextOutputFormat.
           class);
        job.setOutputValueClass(GIS.class);
        job.setOutputFormat(GISOutputFormat.class);

        return JobClient.runJob(job).getJobState();
}

// Hadoop runner requires this to be a static void!
// Thus must use exit instead of return
// Also must directly use the class name instead of
    figuring it out
public static void main(String[] args) throws
   Exception {
        System.exit(ToolRunner.run(new
           Configuration(), new hadoopGIS.examples.
           create(), args));
```

```
        }
}
```

A.2.2 Read

```
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;

import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;

import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;
```

```
public class read extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, GIS>, Reducer<
   LongWritable, GIS, LongWritable, GIS>
{
        // For Mapper interface
        public void map(LongWritable key, GIS value,
           OutputCollector<LongWritable, GIS> output,
           Reporter reporter) throws IOException
        {
                // emit only the record with the correct
                   key
                if(key.equals(new LongWritable(1008130))) {
                        output.collect(key, value);
                }
        }

        // For Reducer interface
        public void reduce(LongWritable key, Iterator<GIS>
           values, OutputCollector<LongWritable, GIS>
           output, Reporter reporter)
        {
                while(values.hasNext()) {
                        try {
                                output.collect(key, values.
                                   next());
                        } catch (IOException e) {}
                }
        }

        // For Mapper (via JobConfigurable) interface
        public void configure(JobConf job)
        {
        }

        // For Mapper (via Closeable) interface
        public void close() {}

        // For Tool interface
        public int run(String[] args) throws Exception
        {
                JobConf job = new JobConf(new Configuration
                   (), this.getClass());
```

```
            GISInputFormat.setInputPaths(job, new Path
               ("/user/alaster/gis/parcels.gis"));
            Path p = new Path ("/user/alaster/gis/
               parcels.names");
            DistributedCache.addCacheFile (p.toUri (),
               job);
            job.set ("columnNames", p.getName ());

            GISOutputFormat.setOutputPath(job, new Path
               ("output"));

            job.setJobName("hadoopGIS.examples.read");

            job.setMapperClass(this.getClass());
            //job.setCombinerClass(this.getClass());
            job.setReducerClass(this.getClass());

            job.setInputFormat(GISInputFormat.class);
            //job.setOutputFormat(TextOutputFormat.
               class);
            job.setOutputValueClass(GIS.class);
            job.setOutputFormat(GISOutputFormat.class);

            return JobClient.runJob(job).getJobState();

      }

      // Hadoop runner requires this to be a static void!
      // Thus must use exit instead of return
      // Also must directly use the class name instead of
          figuring it out
      public static void main(String[] args) throws
         Exception {
            System.exit(ToolRunner.run(new
               Configuration(), new hadoopGIS.examples.
               read(), args));
      }
}
```

A.2.3 Update

```java
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;

import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;

import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;

public class update extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, GIS>, Reducer<
   LongWritable, GIS, LongWritable, GIS>
{
        // For Mapper interface
```

```
public void map(LongWritable key, GIS value,
   OutputCollector<LongWritable, GIS> output,
   Reporter reporter) throws IOException
{
        // Change record 1008130 to a Commercial
           parcel
        if(key.equals(new LongWritable(1008130))) {
                value.attributes.put("devtype", "C
                   ");
        }
        output.collect(key, value);
}

// For Reducer interface
public void reduce(LongWritable key, Iterator<GIS>
   values, OutputCollector<LongWritable, GIS>
   output, Reporter reporter)
{
        while(values.hasNext()) {
                try {
                        output.collect(key, values.
                           next());
                } catch (IOException e) {}
        }
}

// For Mapper (via JobConfigurable) interface
public void configure(JobConf job)
{
}

// For Mapper (via Closeable) interface
public void close() {}

// For Tool interface
public int run(String[] args) throws Exception
{
        JobConf job = new JobConf(new Configuration
           (), this.getClass());

        GISInputFormat.setInputPaths(job, new Path
           ("/user/alaster/gis/parcels.gis"));
```

```
                Path p = new Path ("/user/alaster/gis/
                   parcels.names");
                DistributedCache.addCacheFile (p.toUri (),
                   job);
                job.set ("columnNames", p.getName ());

                GISOutputFormat.setOutputPath(job, new Path
                   ("output"));

                job.setJobName("hadoopGIS.examples.update")
                   ;

                job.setMapperClass(this.getClass());
                //job.setCombinerClass(this.getClass());
                job.setReducerClass(this.getClass());

                job.setInputFormat(GISInputFormat.class);
                //job.setOutputFormat(TextOutputFormat.
                   class);
                job.setOutputValueClass(GIS.class);
                job.setOutputFormat(GISOutputFormat.class);

                return JobClient.runJob(job).getJobState();
        }

        // Hadoop runner requires this to be a static void!
        // Thus must use exit instead of return
        // Also must directly use the class name instead of
            figuring it out
        public static void main(String[] args) throws
           Exception {
                System.exit(ToolRunner.run(new
                   Configuration(), new hadoopGIS.examples.
                   update(), args));
        }
}
```

A.2.4 Delete

```
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;

import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;

import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;

public class delete extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, GIS>, Reducer<
   LongWritable, GIS, LongWritable, GIS>
{
        // For Mapper interface
        public void map(LongWritable key, GIS value,
           OutputCollector<LongWritable, GIS> output,
           Reporter reporter) throws IOException
```

```
{
        // emit only the record with the correct
           key
        if(!key.equals(new LongWritable(1008130)))
           {
                output.collect(key, value);
        }
}

// For Reducer interface
public void reduce(LongWritable key, Iterator<GIS>
   values, OutputCollector<LongWritable, GIS>
   output, Reporter reporter)
{
        while(values.hasNext()) {
                try {
                        output.collect(key, values.
                           next());
                } catch (IOException e) {}
        }
}

// For Mapper (via JobConfigurable) interface
public void configure(JobConf job)
{
}

// For Mapper (via Closeable) interface
public void close() {}

// For Tool interface
public int run(String[] args) throws Exception
{
        JobConf job = new JobConf(new Configuration
           (), this.getClass());

        GISInputFormat.setInputPaths(job, new Path
           ("/user/alaster/gis/parcels.gis"));
        Path p = new Path ("/user/alaster/gis/
           parcels.names");
        DistributedCache.addCacheFile (p.toUri (),
           job);
```

```
            job.set ("columnNames", p.getName ());

            GISOutputFormat.setOutputPath(job, new Path
                ("output"));

            job.setJobName("hadoopGIS.examples.delete")
                ;

            job.setMapperClass(this.getClass());
            //job.setCombinerClass(this.getClass());
            job.setReducerClass(this.getClass());

            job.setInputFormat(GISInputFormat.class);
            //job.setOutputFormat(TextOutputFormat.
                class);
            job.setOutputValueClass(GIS.class);
            job.setOutputFormat(GISOutputFormat.class);

            return JobClient.runJob(job).getJobState();
        }

        // Hadoop runner requires this to be a static void!
        // Thus must use exit instead of return
        // Also must directly use the class name instead of
             figuring it out
        public static void main(String[] args) throws
            Exception {
                System.exit(ToolRunner.run(new
                    Configuration(), new hadoopGIS.examples.
                    delete(), args));
        }
}
```

A.2.5 Filter

```
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
```

```
import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;

import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;

public class filter extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, GIS>, Reducer<
   LongWritable, GIS, LongWritable, GIS>
{
        private Geometry box;

        // For Mapper interface
        public void map(LongWritable key, GIS value,
           OutputCollector<LongWritable, GIS> output,
           Reporter reporter) throws IOException
        {
              // Keep records that intersect with the box
```

```
        if(value.geometry.intersects(box)) {
                output.collect(key, value);
        }
}

// For Reducer interface
public void reduce(LongWritable key, Iterator<GIS>
   values, OutputCollector<LongWritable, GIS>
   output, Reporter reporter)
{
        while(values.hasNext()) {
                try {
                        output.collect(key, values.
                           next());
                } catch (IOException e) {}
        }
}

// For Mapper (via JobConfigurable) interface
public void configure(JobConf job)
{
        ArrayList<String> columns = new ArrayList<
           String>();

        // Create the column list that is used
        columns.add("id");
        columns.add("the_geom");
        columns.add("devtype");

        // Create the box
        try {
                box = new WKTReader().read("POLYGON
                   ((-112.0859375
                   33.4349975585938,-112.0859375
                   33.4675445556641,-112.059799194336

                   33.4675445556641,-112.059799194336
                    33.4349975585938,-112.0859375
                   33.4349975585938))");
        } catch(com.vividsolutions.jts.io.
           ParseException e) {}
}
```

```
// For Mapper (via Closeable) interface
public void close() {}

// For Tool interface
public int run(String[] args) throws Exception
{
        JobConf job = new JobConf(new Configuration
           (), this.getClass());

        GISInputFormat.setInputPaths(job, new Path
           ("/user/alaster/gis/parcels.gis"));
        Path p = new Path ("/user/alaster/gis/
           parcels.names");
        DistributedCache.addCacheFile (p.toUri (),
           job);
        job.set ("columnNames", p.getName ());

        GISOutputFormat.setOutputPath(job, new Path
           ("output"));

        job.setJobName("hadoopGIS.examples.filter")
           ;

        job.setMapperClass(this.getClass());
        //job.setCombinerClass(this.getClass());
        job.setReducerClass(this.getClass());

        job.setInputFormat(GISInputFormat.class);
        //job.setOutputFormat(TextOutputFormat.
           class);
        job.setOutputValueClass(GIS.class);
        job.setOutputFormat(GISOutputFormat.class);

        return JobClient.runJob(job).getJobState();
}

// Hadoop runner requires this to be a static void!
// Thus must use exit instead of return
// Also must directly use the class name instead of
    figuring it out
public static void main(String[] args) throws
```

```
            Exception {
                System.exit(ToolRunner.run(new
                    Configuration(), new hadoopGIS.examples.
                    filter(), args));
        }
}
```

A.2.6 Nearest

```
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;

import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;

import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;
```

```
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;

public class chained extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, LongWritable>,
   Reducer<LongWritable, LongWritable, LongWritable,
   LongWritable>
{
        ArrayList<GIS> parcels;

        // For Mapper interface
        public void map(LongWritable key, GIS value,
           OutputCollector<LongWritable, LongWritable>
           output, Reporter reporter) throws IOException
        {
                double minDistance = Double.MAX_VALUE,
                   currDistance;
                int closestParcel = -1;

                Iterator it = parcels.iterator();
                while (it.hasNext())
                {
                        GIS parcel = (GIS) it.next();

                        currDistance = value.geometry.
                           distance(parcel.geometry);
                        if (currDistance < minDistance)
                        {
                                minDistance = currDistance;
                                closestParcel = new Integer
                                   (parcel.attributes.get("
                                   id"));
                        }
                }

                LongWritable lngClosestParcel = new
                   LongWritable (closestParcel);
                output.collect(key, lngClosestParcel);
        }

        // For Reducer interface
```

```
public void reduce(LongWritable key, Iterator<
   LongWritable> values, OutputCollector<
   LongWritable, LongWritable> output, Reporter
   reporter)
{
        while(values.hasNext()) {
                try {
                        output.collect(key, values.
                           next());
                } catch (IOException e) {}
        }
}

// For Mapper (via JobConfigurable) interface
public void configure(JobConf job)
{
        String columnFilename = job.get ("
           parcelColumnNames");
        String dataFilename = job.get ("parcelData
           ");
        Path[] distCacheFiles = new Path[0];
        try { distCacheFiles = DistributedCache.
           getLocalCacheFiles(job); }
        catch (IOException e)   { return; }

        ArrayList<String> parcelColumnList = new
           ArrayList<String>();
        parcels = new ArrayList<GIS>();

        BufferedReader reader = null;
        String line;
        for (int i=0; i<distCacheFiles.length; i++)
        {
                if (distCacheFiles [i].getName ().
                   equals (columnFilename))
                {
                        try
                        {
                                reader = new
                                   BufferedReader(
                                   new FileReader(
                                   distCacheFiles [
```

```
                                        i].toString()))
                                        ;
                                    while ((line =
                                        reader.readLine
                                        ()) != null)
                                            parcelColumnList
                                                .add(
                                                line);
                        }
                        catch (Exception e) { }
                        break;
                }
        }

        GIS myGIS;
        for (int i=0; i<distCacheFiles.length; i++)
        {
                if (distCacheFiles[i].getName().
                    equals(dataFilename))
                {
                        try
                        {
                                reader = new
                                    BufferedReader(
                                    new FileReader(
                                    distCacheFiles[
                                    i].toString()))
                                    ;
                                while ((line =
                                    reader.readLine
                                    ()) != null)
                                {
                                        myGIS = new
                                            GIS();
                                        myGIS.
                                            update(
                                            new Text
                                             (line),

                                            parcelColumnList
                                            );
                                        parcels.add
```

```
                                        (myGIS)
                                    ;
                              }
                        }
                        catch (Exception e) { }

                        break;
                  }
            }
      }

      // For Mapper (via Closeable) interface
      public void close() {}

      // For Tool interface
      public int run(String[] args) throws Exception
      {
            JobConf job = new JobConf(new Configuration
               (), this.getClass());

            GISInputFormat.setInputPaths(job, new Path
               ("/user/alaster/gis/jobs.gis"));
            GISOutputFormat.setOutputPath(job, new Path
               ("output"));

            job.setJobName("hadoopGIS.examples.nearest
               ");

            job.setMapperClass(this.getClass());
            //job.setCombinerClass(this.getClass());
            job.setReducerClass(this.getClass());

            job.setInputFormat(GISInputFormat.class);
            //job.setOutputFormat(TextOutputFormat.
               class);
            job.setOutputValueClass(LongWritable.class)
               ;

            Path p = new Path ("/user/alaster/gis/jobs.
               names");
            DistributedCache.addCacheFile (p.toUri (),
               job);
```

```
                job.set ("columnNames", p.getName ());

                p = new Path ("/user/alaster/gis/parcels.
                   names");
                DistributedCache.addCacheFile (p.toUri (),
                   job);
                job.set ("parcelColumnNames", p.getName ())
                   ;

                p = new Path ("/user/alaster/gis/parcels.
                   gis");
                DistributedCache.addCacheFile (p.toUri (),
                   job);
                job.set ("parcelData", p.getName ());

                return JobClient.runJob(job).getJobState();
        }

        // Hadoop runner requires this to be a static void!
        // Thus must use exit instead of return
        // Also must directly use the class name instead of
            figuring it out
        public static void main(String[] args) throws
           Exception {
                System.exit(ToolRunner.run(new
                   Configuration(), new hadoopGIS.examples.
                   nearest(), args));
        }
}
```

A.2.7 Chained

```
package hadoopGIS.examples;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;

import hadoopGIS.GIS;
import hadoopGIS.GISInputFormat;
import hadoopGIS.GISOutputFormat;
```

```
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Iterator;
import java.util.HashMap;
import java.util.Map;
import java.util.ArrayList;

import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.filecache.DistributedCache;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.io.WKTReader;
import com.vividsolutions.jts.io.ParseException;

public class chained extends Configured implements Tool,
   Mapper<LongWritable, GIS, LongWritable, LongWritable>,
   Reducer<LongWritable, LongWritable, LongWritable,
   LongWritable>
{
        ArrayList<GIS> parcels;

        // For Mapper interface
        public void map(LongWritable key, GIS value,
           OutputCollector<LongWritable, LongWritable>
           output, Reporter reporter) throws IOException
        {
                double minDistance = Double.MAX_VALUE,
                   currDistance;
                int closestParcel = -1;
```

```
        Iterator it = parcels.iterator();
        while (it.hasNext())
        {
                GIS parcel = (GIS) it.next();

                currDistance = value.geometry.
                   distance(parcel.geometry);
                if (currDistance < minDistance)
                {
                        minDistance = currDistance;
                        closestParcel = new Integer
                           (parcel.attributes.get("
                           id"));
                }
        }

        LongWritable lngClosestParcel = new
           LongWritable (closestParcel);
        output.collect(key, lngClosestParcel);
}

// For Reducer interface
public void reduce(LongWritable key, Iterator<
   LongWritable> values, OutputCollector<
   LongWritable, LongWritable> output, Reporter
   reporter)
{
        while(values.hasNext()) {
                try {
                        output.collect(key, values.
                           next());
                } catch (IOException e) {}
        }
}

// For Mapper (via JobConfigurable) interface
public void configure(JobConf job)
{
        String columnFilename = job.get ("
           parcelColumnNames");
        String dataFilename = job.get ("parcelData
```

```
   ");
Path[] distCacheFiles = new Path[0];
try { distCacheFiles = DistributedCache.
   getLocalCacheFiles(job); }
catch (IOException e)   { return; }

ArrayList<String> parcelColumnList = new
   ArrayList<String>();
parcels = new ArrayList<GIS>();

BufferedReader reader = null;
String line;
for (int i=0; i<distCacheFiles.length; i++)
{
        if (distCacheFiles[i].getName().
           equals(columnFilename))
        {
                try
                {
                        reader = new
                           BufferedReader(
                           new FileReader(
                           distCacheFiles[
                           i].toString()))
                           ;
                        while ((line =
                           reader.readLine
                           ()) != null)
                                parcelColumnList
                                   .add(
                                   line);
                }
                catch (Exception e) { }
                break;
        }
}

GIS myGIS;
for (int i=0; i<distCacheFiles.length; i++)
{
        if (distCacheFiles[i].getName().
           equals(dataFilename))
```

```
{
        try
        {
                reader = new
                   BufferedReader(
                   new FileReader(
                   distCacheFiles [
                   i ].toString ()))
                   ;
                while ((line =
                   reader.readLine
                   ()) != null)
                {
                        myGIS = new
                            GIS();
                        myGIS.
                           update (
                           new Text
                            (line),

                           parcelColumnList
                           );
                        if (!myGIS.
                           attributes
                           .get ("
                           devtype
                           ").
                           equals("
                           R"))
                        {
                                parcels
                                   .
                                   add

                                   (
                                   myGIS
                                   )
                                   ;

                        }
                }
        }
```

```
                    catch (Exception e) { }

                    break;
                }
            }
        }

        // For Mapper (via Closeable) interface
        public void close() {}

        // For Tool interface
        public int run(String[] args) throws Exception
        {
            JobConf job = new JobConf(new Configuration
                (), this.getClass());

            GISInputFormat.setInputPaths(job, new Path
                ("/user/alaster/gis/jobs.gis"));
            GISOutputFormat.setOutputPath(job, new Path
                ("output"));

            job.setJobName("hadoopGIS.examples.chained
                ");

            job.setMapperClass(this.getClass());
            //job.setCombinerClass(this.getClass());
            job.setReducerClass(this.getClass());

            job.setInputFormat(GISInputFormat.class);
            //job.setOutputFormat(TextOutputFormat.
                class);
            job.setOutputValueClass(LongWritable.class)
                ;

            Path p = new Path ("/user/alaster/gis/jobs.
                names");
            DistributedCache.addCacheFile (p.toUri (),
                job);
            job.set ("columnNames", p.getName ());

            p = new Path ("/user/alaster/gis/parcels.
                names");
```

```
            DistributedCache.addCacheFile (p.toUri (),
               job);
            job.set ("parcelColumnNames", p.getName ())
               ;

            p = new Path ("/user/alaster/gis/parcels.
               gis");
            DistributedCache.addCacheFile (p.toUri (),
               job);
            job.set ("parcelData", p.getName ());

            return JobClient.runJob(job).getJobState();
        }

        // Hadoop runner requires this to be a static void!
        // Thus must use exit instead of return
        // Also must directly use the class name instead of
            figuring it out
        public static void main(String[] args) throws
           Exception {
            System.exit(ToolRunner.run(new
               Configuration(), new hadoopGIS.examples.
               chained(), args));
        }
}
```

CHAPTER B CLUSTERGIS SOURCE

B.1 Library

B.1.1 clustergis.h

```
#ifndef CLUSTERGIS_H
#define CLUSTERGIS_H

#define CLUSTERGIS_BUFFERSIZE 2*1024*1024

#include "stdio.h"
#include "stdlib.h"
#include "mpi.h"
#include "geos_c.h"

/* variables */
int clusterGIS_started;

/* datatypes */
struct clusterGIS_record_el {
        char** data;
        int columns;
        GEOSGeometry* geometry;
        struct clusterGIS_record_el * next;
};
typedef struct clusterGIS_record_el clusterGIS_record;
struct clusterGIS_dataset {
        clusterGIS_record* data;
};
typedef struct clusterGIS_dataset clusterGIS_dataset;

/* startup and shutdown */
void clusterGIS_Init(int* argc, char*** argv);
void clusterGIS_Finalize(void);

/* dataset operations */
clusterGIS_dataset* clusterGIS_Create_dataset(void);
clusterGIS_dataset* clusterGIS_Load_csv_distributed(
   MPI_Comm comm, char* filename);
clusterGIS_dataset* clusterGIS_Load_csv_replicated(MPI_Comm
    comm, char* filename);
```

```
void clusterGIS_Write_csv(char* filename,
   clusterGIS_dataset* dataset);
void clusterGIS_Write_csv_distributed(MPI_Comm comm, char*
   filename, clusterGIS_dataset* dataset);
void clusterGIS_Free_dataset(clusterGIS_dataset* dataset);

/* record operations */
clusterGIS_record* clusterGIS_Create_record_from_csv(char*
   csv, int* size);
void clusterGIS_Free_record(clusterGIS_record* record);

/* MPI operations */
MPI_Comm clusterGIS_Create_chunked_communicator(MPI_Comm
   comm, int size);
MPI_Comm clusterGIS_Create_strided_communicator(MPI_Comm
   comm, int stride);

/* Geometry operations */
void clusterGIS_Create_wkt_geometries(clusterGIS_dataset*
   dataset, int geometry_column);
void clusterGIS_Create_wkt_geometry(clusterGIS_record*
   record, int geometry_column);

#endif
```

B.1.2 clustergis.c

```
#include "clustergis.h"
#include "string.h"
#include "sys/stat.h"
#include "assert.h"

/* clusterGIS_Init
 *
 * Sets up the clusterGIS environment
 *
 * argc - count of arguments in argv
 * argv - char** of arguments
 */
void clusterGIS_Init(int* argc, char*** argv) {
        MPI_Init(argc, argv);
        initGEOS(NULL, NULL);
```

```
        clusterGIS_started = 1;
}

/* clusterGIS_Finalize
 *
 * Closes out the clusterGIS environment
 */
void clusterGIS_Finalize(void) {
        MPI_Finalize();
        finishGEOS();
}

/* dataset operations */

/* clusterGIS_Create_dataset
 *
 * Creates a new clusterGIS_dataset
 */
clusterGIS_dataset* clusterGIS_Create_dataset(void) {
        clusterGIS_dataset* dataset = malloc(sizeof(
           clusterGIS_dataset));
        dataset->data = NULL;

        return dataset;
}

/* clusterGIS_Load_csv_distributed
 *
 * Loads a portion of a dataset on each task
 *
 * comm - MPI communicator to use
 * filename - path to the dataset
 * dataset - the dataset which will be created
 */
clusterGIS_dataset* clusterGIS_Load_csv_distributed(
   MPI_Comm comm, char* filename) {
        MPI_File file;
        int err;
        char* buffer;
        MPI_Status status;
        clusterGIS_record** record;
```

```
MPI_Offset offset;
MPI_Offset chunkstart;
MPI_Offset chunkend;
int count;
MPI_Offset filesize;
int i;
int start;
int end;
clusterGIS_dataset* dataset;
int comm_rank;
int comm_size;

MPI_Comm_rank(comm, &comm_rank);
MPI_Comm_size(comm, &comm_size);

err = MPI_File_open(MPI_COMM_WORLD, filename,
   MPI_MODE_RDONLY, MPI_INFO_NULL, &file);
assert(err == MPI_SUCCESS);

buffer = (char*) malloc(CLUSTERGIS_BUFFERSIZE);
MPI_File_get_size(file, &filesize);
offset = 0;
dataset = (clusterGIS_dataset*) malloc(sizeof(
   clusterGIS_dataset));
record = &dataset->data;

/* determine chunksizes, last task picks up the
   slack */
chunkstart = comm_rank * (filesize / comm_size);
if (comm_rank == comm_size - 1) {
        chunkend = filesize;
} else {
        chunkend = chunkstart + (filesize /
           comm_size) - 1;
}

offset = chunkstart;
while(offset < chunkend - 1) {
        MPI_File_read_at(file, offset, buffer,
           CLUSTERGIS_BUFFERSIZE, MPI_CHAR, &status
           );
        MPI_Get_count(&status, MPI_CHAR, &count);
```

```
/* determine the start of this record set
   */
start = 0;
if (offset == chunkstart && comm_rank != 0)
    {
        while(buffer[start] != '\n' &&
           start < count) {
                start++;
        }
        if(start == count) {
                fprintf(stderr, "%d: buffer
                     is too small\n",
                   comm_rank);
                MPI_Abort(MPI_COMM_WORLD,
                   1);
        }
        start++;
}

/* determine the end of this record set */
end = count - 1;
while(end > 0 && buffer[end] != '\n') {
        end--;
}
if(end == 0) {
        fprintf(stderr, "%d: \\n not found
           from end\n", comm_rank);
        MPI_Abort(MPI_COMM_WORLD, 1);
}
end++;

if(chunkend - offset < count) {
        /* Buffer overruns the
           responsibility of this task */
        end = chunkend - offset + 1;
        while(end < count && buffer[end] !=
             '\n') {
                end++;
        }
        end++;
        if (end > count - 1) {
```

```
                    end = chunkend - offset +
                       1;
                    while(end > 0 && buffer[end
                       ] != '\n') {
                            end--;
                    }
                    if(end == 0) {
                            fprintf(stderr, "%d
                               : \\n not found
                               from end\n",
                               comm_rank);
                            MPI_Abort(
                               MPI_COMM_WORLD,
                               1);
                    }
                    end++;
            }
        }

        /* Put the records into the dataset */
        i = start;
        while (i < end) {
                (*record) =
                   clusterGIS_Create_record_from_csv
                   (buffer, &i);
                (*record)->next = NULL;
                record = &(*record)->next;
                i++;
        }

        offset += end;
    }

    free(buffer);
    MPI_File_close(&file);

    return dataset;
}

/* clusterGIS_Load_csv_replicated
 *
 * Loads an entire copy of a csv data source on each task
```

```
   included in comm
 *
 * comm - MPI communicator of which all members will get a
   copy of this dataset
 * filename - path to the csv formatted dataset
 *
 * returns a pointer to the dataset
 */
clusterGIS_dataset* clusterGIS_Load_csv_replicated(MPI_Comm
   comm, char* filename) {
        MPI_File file;
        int err;
        char* buffer;
        int buffersize = 2*1024*1024;
        MPI_Status status;
        clusterGIS_record** record;
        MPI_Offset offset;
        int count;
        int last_full_record_end;
        MPI_Offset filesize;
        int i;
        clusterGIS_dataset* dataset;
        int comm_rank;

        MPI_Comm_rank(comm, &comm_rank);

        err = MPI_File_open(comm, filename, MPI_MODE_RDONLY
           , MPI_INFO_NULL, &file);
        if(err != MPI_SUCCESS) {
                fprintf(stderr, "%d: Error opening file %s\
                   n", comm_rank, filename);
                MPI_Abort(comm, err);
        }

        buffer = (char*) malloc(buffersize);
        MPI_File_get_size(file, &filesize);
        offset = 0;
        dataset = (clusterGIS_dataset*) malloc(sizeof(
           clusterGIS_dataset));
        record = &dataset->data;

        while(offset < filesize) {
```

```
                MPI_File_read_at_all(file, offset, buffer,
                   buffersize, MPI_CHAR, &status);
                MPI_Get_count(&status, MPI_CHAR, &count);

                /* find where the last full record ends */
                last_full_record_end = count - 1;
                while(buffer[last_full_record_end] != '\n'
                  && last_full_record_end >= 0) {
                        last_full_record_end--;
                }
                if(last_full_record_end < 0) {
                        fprintf(stderr, "%d: Error in
                           clusterGIS_Load_data_replicated:
                            buffersize is too small, %d\n",
                            comm_rank, count);
                        MPI_Abort(comm, 1);
                }

                /* Put the records into the dataset */
                i = 0;
                while (i < last_full_record_end) {
                        (*record) =
                           clusterGIS_Create_record_from_csv
                           (buffer, &i);
                        (*record)->next = NULL;
                        record = &(*record)->next;
                        i++;
                }

                offset = offset + last_full_record_end + 1;
        }

        free(buffer);
        MPI_File_close(&file);

        return dataset;
}

/* clusterGIS_Write_csv
 *
 * Writes a dataset out to a file as csv
 *
```

```
 * filename - file to write to
 * dataset - dataset to write
 */
void clusterGIS_Write_csv(char* filename,
   clusterGIS_dataset* dataset) {
        FILE* file;
        clusterGIS_record* record;
        int i;

        remove(filename);
        file = fopen(filename, "w");

        record = dataset->data;
        while(record != NULL) {
                fprintf(file, "\"%s\"", record->data[0]);
                for(i = 1; i < record->columns; i++) {
                        fprintf(file, ",\"%s\"", record->
                           data[i]);
                }
                fprintf(file, "\n");
                record = record->next;
        }

        fclose(file);
}

/* clusterGIS_Write_csv_distributed
 *
 * Writes a distributed dataset to disk using MPI-IO
 *
 * comm - MPI communicator of the participants of the
    distributed dataset
 * filename - path of the file to write to
 * dataset - dataset to write
 */
void clusterGIS_Write_csv_distributed(MPI_Comm comm, char*
   filename, clusterGIS_dataset* dataset) {
        char* filename_part;
        MPI_Offset filesize;
        MPI_Offset offset;
        int i;
        int tmp;
```

```
FILE* file_part;
char* buffer;
int buffersize=1024*1024;
MPI_File file;
struct stat *file_part_stat;
int size;
MPI_Status status;
int comm_rank;
int comm_size;

MPI_Comm_rank(comm, &comm_rank);
MPI_Comm_size(comm, &comm_size);

/* Write local part of dataset to individual files
   */
filename_part = (char*) malloc(strlen(filename) +
   5);
sprintf(filename_part, "%s.%d", filename, comm_rank
   );
clusterGIS_Write_csv(filename_part, dataset);

/* Figure out offset by talking with other tasks */
file_part_stat = malloc(sizeof(struct stat));
stat(filename_part, file_part_stat);
filesize = (int) file_part_stat->st_size;
free(file_part_stat);

offset = 0;
for(i = 0; i < comm_size; i++) {
        tmp = filesize;
        MPI_Bcast(&tmp, 1, MPI_INT, i, comm);
        if (i < comm_rank) {
                offset += tmp;
        }
}

/* Write local parts back together into a single
   large file */
file_part = fopen(filename_part, "r");
remove(filename);
MPI_File_open(comm, filename, MPI_MODE_WRONLY |
   MPI_MODE_CREATE, MPI_INFO_NULL, &file);
```

```
        buffer = (char*) malloc(buffersize);

        size = fread(buffer, 1, buffersize, file_part);
        while(size != 0) {
                MPI_File_write_at(file, offset, buffer,
                   size, MPI_CHAR, &status);
                offset += size;
                size = fread(buffer, 1, buffersize,
                   file_part);
        }
        free(buffer);
        fclose(file_part);
        remove(filename_part);
        MPI_File_close(&file);

        free(filename_part);
}

/* clusterGIS_Free_dataset
 *
 * Frees all memory associated with a dataset (all
   associated records, etc)
 *
 * dataset - the dataset to be freed
 */
void clusterGIS_Free_dataset(clusterGIS_dataset* dataset) {
        clusterGIS_record* head;
        clusterGIS_record* current;

        head = dataset->data;
        current = head;
        while(current != NULL) {
                head = current->next;
                clusterGIS_Free_record(current);
                current = head;
        }

        free(dataset);
}

/* clusterGIS_Create_record_from_csv
 *
```

```
 * Creates a record from the given csv formatted char*
 *
 * csv - csv formatted representation of record
 * start - index of the start of the record in csv,
   returned with the end index
 *
 * Returns generated record
 */
clusterGIS_record* clusterGIS_Create_record_from_csv(char*
  csv, int* start) {
        int end = *start;
        int field_end;
        int field_count;
        int field_start;
        int i;
        int j;
        clusterGIS_record* record;

        struct item {
                char* data;
                int len;
                struct item* next;
        };
        struct item* head;
        struct item* current;

        /* find end of record */
        while(csv[end] != '\n') {
                end++;
        }

        /* Pull the fields out of the record. Assumes
           fields are comma delimited. Quotes surround
           fields with commas or quotes (escaped in the
           field) */
        i = *start;
        field_count = 0;
        head = (struct item*) malloc(sizeof(struct item));
        current = head;
        current->next = NULL;
        while(i < end) {
                /* get the start and stop indexes of the
```

```
            field */
        if(csv[i] == '"') {
                /* escaped field */
                i++;
                field_start = i;
                while (csv[i] != '"' && csv[i-1] !=
                    '\\') {
                        i++;
                }
                field_end = i;
                i++; /* moves to the , or \n that
                   follows the " */
        } else {
                /* non escaped field */
                field_start = i;
                while(csv[i] != ',' && csv[i] != '\
                   n') {
                        i++;
                }
                field_end = i;
        }

        /* add field to the linked list and move to
            the next field */
        current->data = (char*) malloc(field_end -
           field_start + 1);
        for(j = 0; j < field_end - field_start; j
           ++) {
                current->data[j] = csv[field_start
                   + j];
        }
        current->data[j] = '\0';

        current->next = (struct item*) malloc(
           sizeof(struct item));
        current = current->next;
        current->next = NULL;
        field_count++;
        i++;
}

/* Convert the linked list to an array of strings
```

```
        */
        current = head;
        i = 0;
        record = (clusterGIS_record*) malloc(sizeof(
            clusterGIS_record));
        record->data = malloc(field_count * sizeof(char*));
        record->columns = field_count;
        record->next = NULL;
        record->geometry = NULL;
        for(i = 0; i < field_count; i++) {
                record->data[i] = current->data;
                head = current->next;
                free(current);
                current = head;
        }

        (*start) = end;
        return record;
}

/* clusterGIS_Free_record
 *
 * Frees all memory associated with a record
 *
 * record - the record to free
 */
void clusterGIS_Free_record(clusterGIS_record* record) {
        if(record != NULL) {
                free(record->data);
                free(record);
        }
}

/* MPI operations */
/* clusterGIS_Create_sub_communicator
 *
 * Creates a new MPI_Comm of size contiguous tasks
 *
 * comm - The communicator which to create a subset from
 * size - The size of the sub comminicator
 * new_comm - The address of the new communicator
 */
```

```
MPI_Comm clusterGIS_Create_chunked_communicator(MPI_Comm
   comm, int size) {
        int* members;
        int start;
        int end;
        int rank;
        int tasks;
        MPI_Group old_group;
        MPI_Group new_group;
        int i;
        MPI_Comm new_comm;

        MPI_Comm_rank(comm, &rank);
        MPI_Comm_size(comm, &tasks);
        MPI_Comm_group(comm, &old_group);

        /* Determine start and end of this group */
        start = rank - (rank % size);
        end = start + size - 1;
        if(end >= tasks) end = tasks - 1;
        size = end - start + 1;

        /* List the members in this group */
        members = malloc(sizeof(int) * size);
        for(i = 0; i < size; i++) {
                members[i] = start + i;
        }

        /* Create the new group, and then from it the new
           communicator */
        MPI_Group_incl(old_group, size, members, &new_group
           );
        MPI_Comm_create(comm, new_group, &new_comm);

        free(members);
        return new_comm;
}

MPI_Comm clusterGIS_Create_strided_communicator(MPI_Comm
   comm, int stride) {
        int* members;
        int comm_rank;
```

```
        int comm_size;
        MPI_Group old_group;
        MPI_Group new_group;
        int i;
        MPI_Comm new_comm;
        int rank;
        int position;

        MPI_Comm_rank(comm, &comm_rank);
        MPI_Comm_size(comm, &comm_size);
        MPI_Comm_group(comm, &old_group);

        /* Determine which stride group we are in */
        position = comm_rank % stride;

        /* List the members in this group */
        i = 0;
        members = malloc(sizeof(int) * comm_size);
        for(rank = 0; rank < comm_size; rank++) {
                if(rank % stride == position) {
                        members[i] = rank;
                        i++;
                }
        }

        /* Create the new group, and then from it the new
           communicator */
        MPI_Group_incl(old_group, i, members, &new_group);
        MPI_Comm_create(comm, new_group, &new_comm);

        free(members);
        return new_comm;
}

/* clusterGIS_Create_wkt_geometries
 *
 * Creates geometries in the dataset from the WKT formatted
     data in geometry_column
 *
 * dataset - dataset to be modified
 * geometry_column - column of the dataset the WKT
    formatted geometry is located in
```

```
 */
void clusterGIS_Create_wkt_geometries(clusterGIS_dataset*
   dataset, int geometry_column) {
        clusterGIS_record* record;

        record = dataset->data;
        while(record != NULL) {
                clusterGIS_Create_wkt_geometry(record,
                   geometry_column);
                record = record->next;
        }
}

/* clusterGIS_Create_wkt_geometry
 *
 * Creates a geometry in the record from the WKT formatted
    datas in geometry_column
 *
 * record - the record to be modified
 * geometry_column - the column in record->data containing
    the WKT formatted geometry data
 */
void clusterGIS_Create_wkt_geometry(clusterGIS_record*
   record, int geometry_column) {
        GEOSWKTReader* reader = GEOSWKTReader_create();
        record->geometry = GEOSWKTReader_read(reader,
           record->data[geometry_column]);
        GEOSWKTReader_destroy(reader);
}
```

B.2 Operations

B.2.1 Create

```
/* File: create.c
 * Author: Nathan Kerr
 *
 * Copies a dataset, adding a new record in the process.
 */

#include "clustergis.h"
```

```
int main(int argc, char** argv) {
        clusterGIS_dataset* dataset;
        clusterGIS_record* record;
        int rank;

        /* Process local arguments */
        if (argc != 3) {
                fprintf(stderr, "Usage: %s input output\n",
                    argv[0]);
                exit(1);
        }

        clusterGIS_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        dataset = clusterGIS_Load_csv_distributed(
           MPI_COMM_WORLD, argv[1]);

        /* adds a new record to the beginning of the
           dataset */
        if(rank == 0) {
                int start = 0;
                record = clusterGIS_Create_record_from_csv
                   ("97123897,POINT(0 0),C\n", &start);
                record->next = dataset->data;
                dataset->data = record;
        }

        /* writes the records to disk */
        clusterGIS_Write_csv_distributed(MPI_COMM_WORLD,
           argv[2], dataset);

        /* Finalize */
        clusterGIS_Finalize();
        return 0;
}
```

B.2.2 Read

```
/* File: read.c
 * Author: Nathan Kerr
 *
 * Outputs the specified record (by id)
```

```
 */

#include "clustergis.h"
#include "string.h"

int main(int argc, char** argv) {
        clusterGIS_dataset* dataset;
        clusterGIS_record* record;
        clusterGIS_record** head;

        /* Process local arguments */
        if (argc != 3) {
                fprintf(stderr, "Usage: %s input output\n",
                    argv[0]);
                exit(1);
        }

        /* Init */
        clusterGIS_Init(&argc, &argv);
        dataset = clusterGIS_Load_csv_distributed(
           MPI_COMM_WORLD, argv[1]);

        record = dataset->data;
        head = &(dataset->data);
        /* keep records that match the criteria, otherwise
           delete them */
        while(record != NULL) {
                if(atoi(record->data[0]) == 1008130) {
                        head = &(record->next);
                        record = record->next;
                } else {
                        *head = record->next;
                        clusterGIS_Free_record(record);
                        record = *head;
                }
        }

        clusterGIS_Write_csv_distributed(MPI_COMM_WORLD,
           argv[2], dataset);

        /* Finalize */
        clusterGIS_Finalize();
```

```
        return 0;
}
```

B.2.3 Update

```
/* File: read.c
 * Author: Nathan Kerr
 *
 * Changes record 1008130 from R to C
 */

#include "clustergis.h"
#include "string.h"

int main(int argc, char** argv) {
        clusterGIS_dataset* dataset;
        clusterGIS_record* record;

        /* Process local arguments */
        if (argc != 3) {
                fprintf(stderr, "Usage: %s input output\n",
                    argv[0]);
                exit(1);
        }

        /* Init */
        clusterGIS_Init(&argc, &argv);
        dataset = clusterGIS_Load_csv_distributed(
           MPI_COMM_WORLD, argv[1]);

        record = dataset->data;
        /* keep records that match the criteria, otherwise
           delete them */
        while(record != NULL) {
                if(atoi(record->data[0]) == 1008130) {
                        record->data[2] = "C";
                }
                record = record->next;
        }

        clusterGIS_Write_csv_distributed(MPI_COMM_WORLD,
           argv[2], dataset);
```

```
        /* Finalize */
        clusterGIS_Finalize();
        return 0;
}
```

B.2.4 Delete

```
/* File: read.c
 * Author: Nathan Kerr
 *
 * Deletes the specified record (by id)
 */

#include "clustergis.h"
#include "string.h"

int main(int argc, char** argv) {
        clusterGIS_dataset* dataset;
        clusterGIS_record* record;
        clusterGIS_record** head;

        /* Process local arguments */
        if (argc != 3) {
                fprintf(stderr, "Usage: %s input output\n",
                    argv[0]);
                exit(1);
        }

        /* Init */
        clusterGIS_Init(&argc, &argv);
        dataset = clusterGIS_Load_csv_distributed(
           MPI_COMM_WORLD, argv[1]);

        record = dataset->data;
        head = &(dataset->data);
        /* keep records that match the criteria, otherwise
           delete them */
        while(record != NULL) {
                if(atoi(record->data[0]) == 1008130) {
                        *head = record->next;
                        clusterGIS_Free_record(record);
```

```
                record = *head;
            } else {
                head = &(record->next);
                record = record->next;
            }
        }

        clusterGIS_Write_csv_distributed(MPI_COMM_WORLD,
           argv[2], dataset);

        /* Finalize */
        clusterGIS_Finalize();
        return 0;
}
```

B.2.5 Filter

```
#include "clustergis.h"
#include "geos_c.h"

int main(int argc, char** argv) {
        GEOSGeometry* box;
        GEOSWKTReader* reader;
        clusterGIS_dataset* dataset;
        clusterGIS_record* record;
        clusterGIS_record** head;
        GEOSGeometry* record_geometry;
        int rank;
        double startprocessing;

        if(argc != 3) {
                fprintf(stderr, "Usage %s input output",
                   argv[0]);
                exit(1);
        }

        clusterGIS_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        reader = GEOSWKTReader_create();

        box = GEOSWKTReader_read(reader, "POLYGON
           ((-112.0859375 33.4349975585938,-112.0859375
```

```
   33.4675445556641,-112.059799194336
   33.4675445556641,-112.059799194336
   33.4349975585938,-112.0859375 33.4349975585938))
   ");
dataset = clusterGIS_Load_csv_distributed(
   MPI_COMM_WORLD, argv[1]);
clusterGIS_Create_wkt_geometries(dataset, 1);

startprocessing = MPI_Wtime();
record = dataset->data;
head = &(dataset->data);
/* keep records that match the criteria, otherwise
   delete them */
while(record != NULL) {
        char intersects;

        /*record_geometry = GEOSWKTReader_read(
           reader, record->data[1]);
        if(record_geometry == NULL) {
                fprintf(stderr, "%d: parse error\n
                   ", rank);
                MPI_Abort(MPI_COMM_WORLD, 2);
        }*/

        intersects = GEOSIntersects(record->
           geometry, box);
        if(intersects == 2) {
                fprintf(stderr, "%d: error with
                   overlap function\n", rank);
                MPI_Abort(MPI_COMM_WORLD, 2);
        }

        if(intersects == 1) { /* record overlaps
           with box */
                head = &(record->next);
                record = record->next;
        } else if(intersects == 0) { /* no overlap
           */
                *head = record->next;
                clusterGIS_Free_record(record);
                record = *head;
        } else {
```

```
                    /* should never get here */
                    MPI_Abort(MPI_COMM_WORLD, 2);
                }
        }
        printf("%d: processing time %5.2fs\n", rank,
           MPI_Wtime() - startprocessing);

        clusterGIS_Write_csv_distributed(MPI_COMM_WORLD,
           argv[2], dataset);

        clusterGIS_Finalize();
        return 0;
}
```

B.2.6 Nearest

```
#include "clustergis.h"
#include "string.h"
#include "nearest.h"

#define BLOCK_SIZE 8
#define EMPLOYERS_GEOMETRY_COLUMN 1
#define PARCELS_GEOMETRY_COLUMN 1

/* reduce function for min distances */
void min_distance_function (double *invec, double* outvec,
   int *len, MPI_Datatype *datatype) {
        if(len == NULL || datatype == NULL) {
                MPI_Abort(MPI_COMM_WORLD, 2);
        }

        /* outvec[i] = invec[i] op outvec[i] */
        if(invec[1] < outvec[1]) {
                outvec[0] = invec[0];
                outvec[1] = invec[1];
        }
}

int main(int argc, char** argv) {
        char* employers_filename;
        char* parcels_filename;
        MPI_Comm employers_comm;
```

```
MPI_Comm parcels_comm;
clusterGIS_dataset* employers;
clusterGIS_dataset* parcels;
clusterGIS_record* employer;
clusterGIS_record* parcel;
double distance;
double min_distance;
clusterGIS_record* min_distance_parcel;
double *min;
double *global_min;
int world_rank;
MPI_Op min_distance_op;
char* output_csv;
clusterGIS_dataset* output = NULL;
clusterGIS_record* output_record = NULL;
int start = 0;
char* output_filename;

clusterGIS_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Op_create((MPI_User_function*)
   min_distance_function, 1, &min_distance_op);

if(argc != 4 && world_rank == 0) {
        printf("Usage: %s employers parcels output\
           n", argv[0]);
        MPI_Abort(MPI_COMM_WORLD, 1);
}
employers_filename = argv[1];
parcels_filename = argv[2];
output_filename = argv[3];


/* Load data into appropriate communicators and
   create their geometries */
employers_comm =
   clusterGIS_Create_strided_communicator(
   MPI_COMM_WORLD, BLOCK_SIZE);
employers = clusterGIS_Load_csv_distributed(
   employers_comm, employers_filename);
clusterGIS_Create_wkt_geometries(employers,
   EMPLOYERS_GEOMETRY_COLUMN);
```

```
parcels_comm =
   clusterGIS_Create_chunked_communicator(
   MPI_COMM_WORLD, BLOCK_SIZE);
parcels = clusterGIS_Load_csv_distributed(
   parcels_comm, parcels_filename);
clusterGIS_Create_wkt_geometries(parcels,
   PARCELS_GEOMETRY_COLUMN);

/* Find the min distance */
employer = employers->data;
min = malloc(sizeof(double)*2);
global_min = malloc(sizeof(double)*2);
output_csv = malloc(sizeof(char)*128);
output = clusterGIS_Create_dataset();
while(employer != NULL) {

        /* find the local min */
        parcel = parcels->data;
        GEOSDistance(employer->geometry, parcel->
           geometry, &min_distance);
        min_distance_parcel = parcel;
        while(parcel != NULL) {
                if(strncmp(employer->data[2],
                   parcel->data[2], 1) == 0) {
                        GEOSDistance(employer->
                           geometry, parcel->
                           geometry, &distance);
                        if(distance < min_distance)
                            {
                                min_distance =
                                   distance;
                                min_distance_parcel
                                   = parcel;
                        }
                }
                parcel = parcel->next;
        }

        /* find the global min */
        min[0] = atoi(min_distance_parcel->data[0])
           ;
        min[1] = min_distance;
```

```
            MPI_Allreduce(min, global_min, 2,
               MPI_DOUBLE, min_distance_op,
               parcels_comm);

            /* Add the min to the output dataset */
            sprintf(output_csv, "\"%s\",\"%d\"\n",
               employer->data[0], (int) global_min[0]);
            start = 0;
            output_record =
               clusterGIS_Create_record_from_csv(
               output_csv, &start);
            output_record->next = output->data;
            output->data = output_record;

            employer = employer->next;
    }

    if(world_rank % BLOCK_SIZE == 0) {
            clusterGIS_Write_csv_distributed(
               employers_comm, output_filename, output)
               ;
    }

    MPI_Op_free(&min_distance_op);
    clusterGIS_Finalize();
    return 0;
}
```

B.2.7 Chained

```
#include "clustergis.h"
#include "string.h"
#include "chained.h"

#define BLOCK_SIZE 8
#define EMPLOYERS_GEOMETRY_COLUMN 1
#define PARCELS_GEOMETRY_COLUMN 1

/* reduce function for min distances */
void min_distance_function (double *invec, double* outvec,
   int *len, MPI_Datatype *datatype) {
    if(len == NULL || datatype == NULL) {
```

```
                MPI_Abort(MPI_COMM_WORLD, 2);
        }
        /* outvec[i] = invec[i] op outvec[i] */
        if(invec[1] < outvec[1]) {
                outvec[0] = invec[0];
                outvec[1] = invec[1];
        }
}

int main(int argc, char** argv) {
        char* employers_filename;
        char* parcels_filename;
        MPI_Comm employers_comm;
        MPI_Comm parcels_comm;
        clusterGIS_dataset* employers;
        clusterGIS_dataset* parcels;
        clusterGIS_record* employer;
        clusterGIS_record* parcel;
        double distance;
        double min_distance;
        clusterGIS_record* min_distance_parcel;
        double *min;
        double *global_min;
        int world_rank;
        MPI_Op min_distance_op;
        char* output_csv;
        clusterGIS_dataset* output = NULL;
        clusterGIS_record* output_record = NULL;
        int start = 0;
        char* output_filename;
        clusterGIS_record** head;

        clusterGIS_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
        MPI_Op_create((MPI_User_function*)
           min_distance_function, 1, &min_distance_op);

        if(argc != 4 && world_rank == 0) {
                printf("Usage: %s employers parcels output\
                   n", argv[0]);
                MPI_Abort(MPI_COMM_WORLD, 1);
        }
```

```
employers_filename = argv[1];
parcels_filename = argv[2];
output_filename = argv[3];


/* Load data into appropriate communicators and
   create their geometries */
employers_comm =
   clusterGIS_Create_strided_communicator(
   MPI_COMM_WORLD, BLOCK_SIZE);
employers = clusterGIS_Load_csv_distributed(
   employers_comm, employers_filename);
clusterGIS_Create_wkt_geometries(employers,
   EMPLOYERS_GEOMETRY_COLUMN);
parcels_comm =
   clusterGIS_Create_chunked_communicator(
   MPI_COMM_WORLD, BLOCK_SIZE);
parcels = clusterGIS_Load_csv_distributed(
   parcels_comm, parcels_filename);
clusterGIS_Create_wkt_geometries(parcels,
   PARCELS_GEOMETRY_COLUMN);

/* remove all residential parcels */
parcel = parcels->data;
head = &(parcels->data);
while(parcel != NULL) {
        if(strncmp(parcel->data[2], "R", 1) == 0) {
                head = &(parcel->next);
                parcel = parcel->next;
        } else {
                *head = parcel->next;
                clusterGIS_Free_record(parcel);
                parcel = *head;
        }
}

/* Find the min distance */
employer = employers->data;
min = malloc(sizeof(double)*2);
global_min = malloc(sizeof(double)*2);
output_csv = malloc(sizeof(char)*128);
output = clusterGIS_Create_dataset();
```

```
while(employer != NULL) {

        /* find the local min */
        parcel = parcels->data;
        GEOSDistance(employer->geometry, parcel->
           geometry, &min_distance);
        min_distance_parcel = parcel;
        while(parcel != NULL) {
                if(strncmp(employer->data[2],
                   parcel->data[2], 1) == 0) {
                        GEOSDistance(employer->
                           geometry, parcel->
                           geometry, &distance);
                        if(distance < min_distance)
                            {
                                min_distance =
                                   distance;
                                min_distance_parcel
                                   = parcel;
                        }
                }
                parcel = parcel->next;
        }

        /* find the global min */
        min[0] = atoi(min_distance_parcel->data[0])
           ;
        min[1] = min_distance;
        MPI_Allreduce(min, global_min, 2,
           MPI_DOUBLE, min_distance_op,
           parcels_comm);

        /* Add the min to the output dataset using
           front insertion*/
        sprintf(output_csv, "\"%s\",\"%d\"\n",
           employer->data[0], (int) global_min[0]);
        start = 0;
        output_record =
           clusterGIS_Create_record_from_csv(
           output_csv, &start);
        output_record->next = output->data;
        output->data = output_record;
```

```
                employer = employer->next;
        }

        /* Write one copy of the result dataset out */
        if(world_rank % BLOCK_SIZE == 0) {
                clusterGIS_Write_csv_distributed(
                   employers_comm, output_filename, output)
                   ;
        }

        MPI_Op_free(&min_distance_op);
        clusterGIS_Finalize();
        return 0;
}
```