Fall 12-16-2016

Spatial Data Mining Analytical Environment for Large Scale Geospatial Data

Zhao Yang
zyang1@uno.edu

Spatial Data Mining Analytical Environment for Large Scale Geospatial Data

A Dissertation

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Engineering and Applied Science
Computer Science

by

Zhao Yang

M.S. University of New Orleans, 2012

December 2016

ACKNOWLEDGEMENT

After an intensive period of four years, today is the day: writing this note of thanks is the finishing touch on my dissertation. It has been a period of intense learning for me, not only in the scientific area, but also on a personal level. Writing this thesis has had a big impact on me. I would like to reflect on the people who have supported and helped me so much throughout this period.

I would like to express my heartfelt gratitude to my major professor Dr. Mahdi Abdelguerfi for his belief in me and the support extended by him throughout my work. It has been a wonderful experience to work under his guidance.

I would like to thank Dr. Shengru Tu who admitted me and taught the “Big Data” course which is the foundation of my dissertation.

I would like to thank Dr. Elias Ioup. We spent five years to do the research project together, which will be grateful memory in my life.

I would like to thank Dr. Christopher M. Summa and Dr. Dimitrios Charalampidis for being my thesis committee members.

Lastly, I would like to thank my cat, my friends and family for their love and support throughout.

I would also like to thank my family for their wise counsel and sympathetic ear. You are always there for me. Finally, there are my friends. We were not only able to support each other by deliberating over our problems and findings, but also happily by talking about

things other than just our papers.

Thank you very much, everyone!

Table of Contents

ABSTRACT

We propose a framework for processing and analyzing large-scale geospatial and environmental data using a “Big Data” infrastructure. Existing Big Data solutions do not include a specific mechanism to analyze large-scale geospatial data. In this work, we extend HBase with Spatial Index(R-Tree) and HDFS to support geospatial data and demonstrate its analytical use with some common geospatial data types and data mining technology provided by the R language. The resulting framework has a robust capability to analyze large-scale geospatial data using spatial data mining and making its outputs available to end users.

Keywords

Big Data,
Hadoop,
Hbase,
MapReduce,
R,
Data Mining,
Geographic Information System

CHAPTER 1 INTRODUCTION

Nowadays, many applications are continuously generating large-scale geospatial data. For example, vehicle GPS tracking data, aerial surveillance drones, LiDAR (Light Detection and Ranging), world-wide spatial networks, and high resolution optical or Synthetic Aperture Radar imagery data all generate a huge amount of geospatial data. For instance, the geospatial image data generated by a 14-hour flight mission of a General Atomics MQ-9 reaper drone with a Gorgon Stare sensor system produces over 70 terabytes [1]. However, as data collection increases our ability to process this large-scale geospatial data in a flexible fashion is still limited.

The ability to analyze large-scale geospatial data is a requirement for many geospatial intelligence [2] users, but current techniques for analyzing this data are overly specialized or ad hoc. These techniques are not designed to allow for user defined analytics methods. Commercial analytical products are incapable of fitting the customer’s special requirements. GIS users are expected to use raw datasets with unknown statistical information—the implicit statistics information is insufficient for analytical purposes. In order to successfully analyze statistics information, users require the various analytical functions on an integrated environment. In the field of ocean and acoustic modeling, there is still limited use of data mining measures on geospatial data. Our framework will provide a new approach for analyzing statistical information and distributions on large-scale geospatial data. The raw geospatial data with unknown error information and uncertainty information [3] can be analyzed over a heterogeneous infrastructure in a well-defined manner.

Existing user-oriented geospatial analytic environments are limited to smaller datasets. Much of the current work in large scale analytics focus on automating analysis tasks, for example detecting suspicious activity in wide area motion imagery [4]. Neither approach provides geospatial analysts with the flexibility to employ creativity and discover new trends in data while still efficiently operating over extremely large data sets. Currently, large-scale user defined analytics must be created by users with expertise in distributed computing frameworks, programming, data processing, and storage in addition to the geospatial and statistical subject matter. A new approach is necessary which hides the details of the distributed computing frameworks behind common geospatial analysis tools while still supporting large-scale analytics.

When undertaking this project, the following factors need be considered. First, raw geospatial data comes in many forms. Therefore, the framework must be able to represent all forms of raw data—in a file or in a database, for instance. Second, the language must support user-defined data mining analysis. Lastly, the data mining functions must be able to run in parallel. The framework must support complete features with reliability, availability, and scalability of processing statistical information from end-to-end.

In this project, we have developed a framework to store and process large-scale geospatial data over a “Big Data” [5] infrastructure while providing users with a common geospatial analysis front-end that hides these infrastructure details. The geospatial data is stored in various data stores including file-based HDFS [6] and Hbase [7] (NoSQL database running on top of HDFS). The spatial data mining function is implemented by the R system. The framework provides high-performance analytical features. Also the framework is flexible and easily extendable.

Compared to the SAS [8] system, our framework supports a variety of user generated geospatial data mining application instead of limited SAS analytical functions. It is convenient to transform current large-scale geospatial data analytical tasks to our framework. The framework is being constructed on an open source software platform, however, use of the framework does not require comprehensive programming skills; the spatial data mining task is easy to share, access, and visualize.

This paper is organized as follows: Section 2 deals with the background knowledge necessary to comprehend and undertake the project; the concepts and definitions of Hadoop, R system, and Hybrid Cloud are presented in this section. Section 3 presents the framework and its architecture and advantages. The designs of the various components that make up the framework are presented in detail in Section 4. The implementation of all the framework components will be discussed in detail in this section. Section 5 presents examples on how to use the framework to analyze large-scale geospatial data and three such applications are presented. Section 6 discusses the conclusions we have drawn and future directions for our work.

CHAPTER 2 BACKGROUND

2.1 Big Data and R

In recent years, Big Data has become a hot topic. Big Data deals with large-scale and complex data more efficiently than traditional database management systems and data storage tools. Big Data provides various methods to extract, transform, load, manipulate, share, and analyze large-scale data with high performance. Hadoop [4] is the most popular solution for distributing and processing large-scale data sets. Hadoop, as data storage infrastructure, can easily scale up from single node to thousands of nodes. To design a reliable, scalable infrastructure with high availability, we have chosen the Apache Hadoop as the data storage layer software in our framework. Here is a sample structure of a Hadoop cluster:

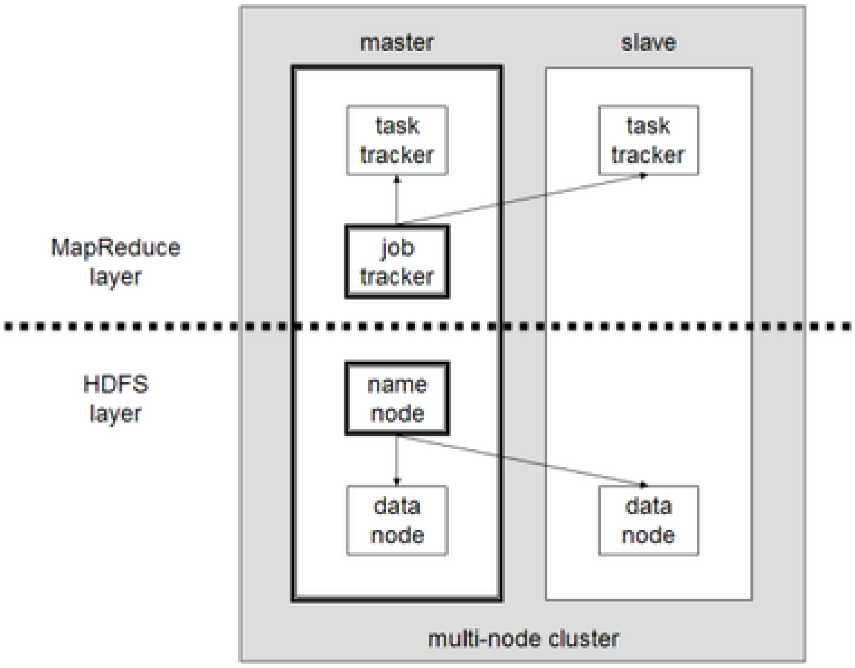


Figure 2-1. Multi-node Hadoop cluster [9]

A Hadoop cluster allows users to process large-scale data on distributed computing environments with simple methods, such as the map-reduce model. The Hadoop clusters may consist of thousands of data nodes. With Hadoop as the data storage infrastructure, the framework can handle petabyte level data. The Hadoop cluster provides high availability features including autonomous handling hardware or software failures. It is proved to be reliable as a high-availability commercial level infrastructure, like Microsoft Azure and Amazon EC2/S3.

A Hadoop cluster consists of Client nodes, Master nodes, and Slave nodes. The Master nodes are the management part of Hadoop and the Name nodes are used to manage the HDFS (Hadoop File System) storage infrastructure. The job tracker, on the other hand, is used to manage MapReduce tasks which are parallel running. Finally, the Slave nodes are used to store the data and processing work. Slave nodes consist of Data nodes and a Task Tracker daemon, and are controlled by Master nodes. The job tracker controls the Task Tracker daemon and the Name node controls the data node daemon.

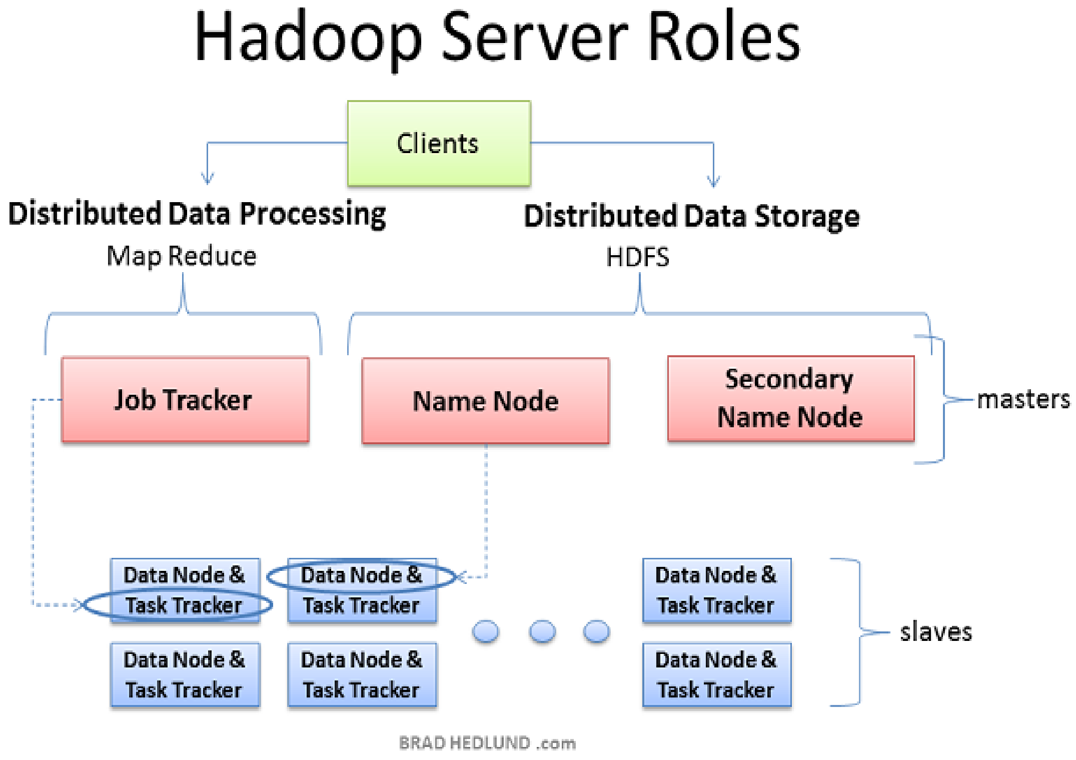


Figure 2-2. Hadoop Server Roles [10]

Users can administrate and access the Hadoop cluster through various interfaces and methods. First, the user can load data into the cluster using shell script commands. Second, the user can submit Hadoop commands to interact with Hadoop data. Hadoop supports Java, C++, and Python. The output of a Hadoop command will be downloaded to the user machine for analytical purposes.

HBase, designed on the basis of Hadoop and ZooKeeper [11], is a centralized service that provides configuration and communication services between different products. The framework uses Apache HBase because the system needs random, real-time read/write access to the geospatial data. This framework will support large-scale geospatial data from the Hadoop infrastructure.

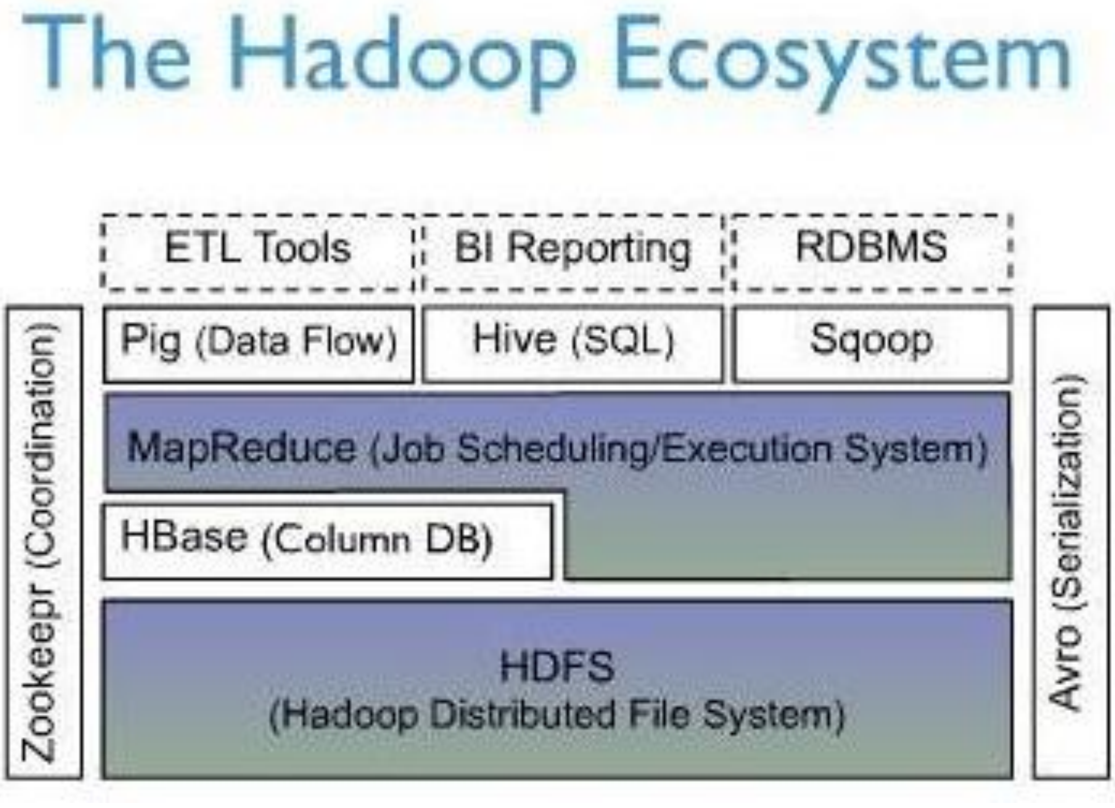


Figure 2-3. The Hadoop Ecosystem [12]

The Hbase's design is based on Google's BigTable [13] model. As the above figure shows, the Hbase uses HDFS as the data storage layer and provides access to the data using data structures like the Big Table format. In our framework, Hbase is the NoSQL DBMS to be used to store and process large-scale geospatial data.

The R software [14] provides functions like SAS and SPSS in the fields of statistical computing and data analysis. R has become very popular among statistician programmers, and its software has been developed from the S programming language. The S language is a statistical programming language designed by John Chambers from Bell Labs[15]. The statistician uses various statistical analysis methods, for instance, linear and nonlinear modeling, classification, clustering, and graphical procedures. R packages can be written using C, C++, FORTRAN, Java, and Python. In the R community, many programmers have developed numerous packages for new functions and extensions.

2.2 Panoply, NetCDF and R

In our framework, we have chosen Panoply [16] as the visualization tool. Panoply can plot geospatial data in multiple platforms such as MAC, Windows, and Linux. It also supports multiple source file formats like NetCDF [17], HDF, and GRIB.

NetCDF [17] is used to share large-scale multidimensional data, which includes bioinformatics data, climate data, and geospatial data. The NetCDF format includes the metadata of the content in the file—for example; both the coordinate value and the attribute's information are stored in the metadata. As a result, a NetCDF file is said to be self-describing [17]. Additionally, these files can be shared between different platforms such as Mac, Windows, and Linux. NetCDF is supported by many platforms and features machine-independence [17], so that data will not be compromised in a shared process.

With the ncdf and ncdf4 packages, R has the capability to read and write the netCDF source file. The netCDF3 and netCDF4 formats are both widely used; however, the netCDF4 format has better support for large-scale data and better compression performance.

2.3 Hybrid Cloud

Hybrid cloud is a mixed concept of public cloud and private cloud. One organization could use both public cloud and private cloud to integrate a universal cloud service. As a cloud computing service, private cloud may focus on a specific purpose for departments while public cloud may focus on efficiency and cost.

The concept of public cloud is not limited to public corporations; it can be corporation level public cloud, which is still private to the outsider. The hybrid cloud may consist of heterogeneous architecture and platform.

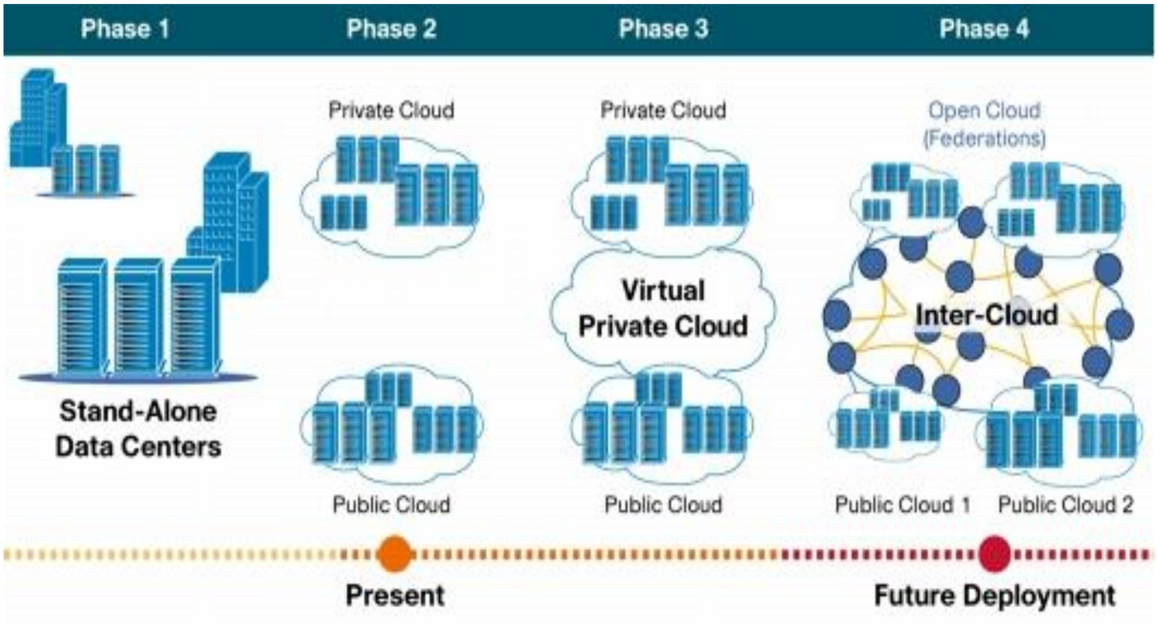
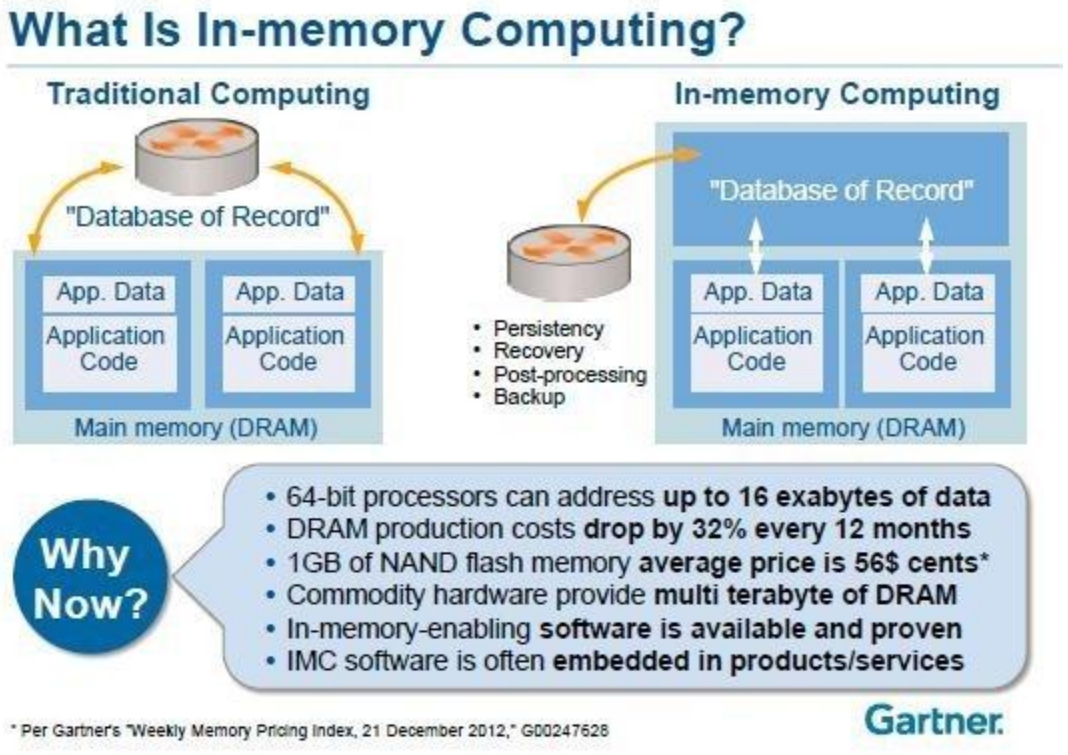


Figure 2-4. Hybrid Cloud [18]

Figure 2-3 shows a hybrid architecture which is presently being used as commercial solutions. This hybrid cloud provides a platform as a service (PaaS) ability which allows a user to run cloud base applications. For large-scale geospatial data, public cloud can be used as data warehouse while private cloud can be used as department cloud.

2.4 In-Memory Computing

In the era of Big Data, In-Memory Computing has become a popular solution for processing large amounts of data in a server’s RAM. The purpose of in-memory computing is to get the fastest speed possible.



What Is In-memory Computing?

Traditional Computing

"Database of Record"

App. Data

Application Code

App. Data

Application Code

Main memory (DRAM)

In-memory Computing

"Database of Record"

- Persistency
- Recovery
- Post-processing
- Backup

App. Data

Application Code

App. Data

Application Code

Main memory (DRAM)

Why Now?

- 64-bit processors can address **up to 16 exabytes of data**
- DRAM production costs **drop by 32% every 12 months**
- 1GB of NAND flash memory **average price is 56$ cents***
- Commodity hardware provide **multi terabyte of DRAM**
- In-memory-enabling **software is available and proven**
- IMC software is often **embedded in products/services**

* Per Gartner's "Weekly Memory Pricing Index, 21 December 2012," G00247628

Gartner.

Figure 2-5. In-Memory Computing [19]

The traditional way of computing reads and writes database records on the disk, which means lots of I/O cost. Another approach is to let application programs load the entire data set into memory. This avoids high I/O cost database access. Multiple concurrent users and programs can access the data and enhance the performance of query. The application code and the application data are all in memory. This makes the computation task run at a faster speed.

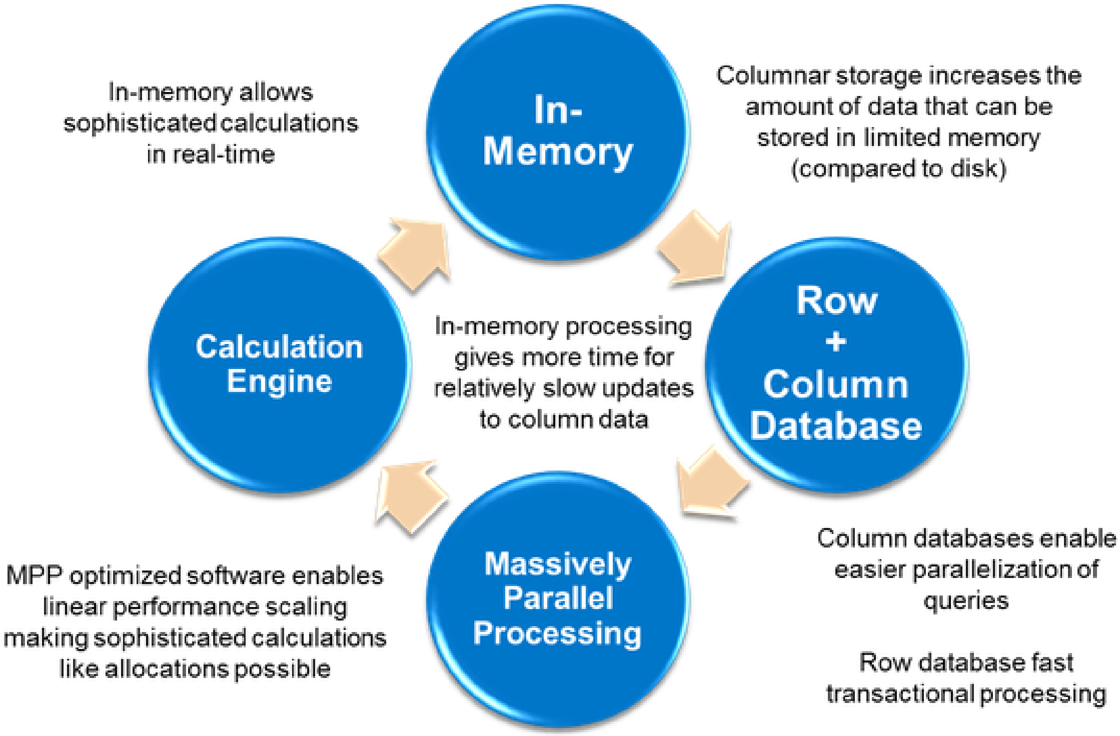

In-memory allows sophisticated calculations in real-time
In-Memory
Columnar storage increases the amount of data that can be stored in limited memory (compared to disk)
Calculation Engine
In-memory processing gives more time for relatively slow updates to column data
Row + Column Database
MPP optimized software enables linear performance scaling making sophisticated calculations like allocations possible
Massively Parallel Processing
Column databases enable easier parallelization of queries
Row database fast transactional processing

Figure 2-6. IBM: What is In-Memory Computing? [20]

In-memory computing provides a real-time response for Big Data applications including data analytics, business intelligence reporting tasks, etc. [21].

2.5 Spatial Data Warehouse and Spatial Data Mining

Central to a spatial data warehouse system is the effective handling of large-scale geospatial data. Spatial data warehouse stores large amount of information about the coordinates of individual spatial objects in space. For OLTP spatial databases, the spatial computation is expensive. The cost of rendering online processing is not acceptable [22] for most users. Driven by the Internet Company's paradigm such as Google, spatial data warehouses can be a solution to accelerate spatial data mining operations.

Spatial Data Warehouse is so crucial to the enablement of an enterprise system that its effective usage will be the technical centerpiece of this work. On-Line Transaction Processing (OLTP) is the traditional model for enterprise data processing. OLTP databases focus on transactions involving the input, update, and retrieval of data. On-Line Analytical Processing (OLAP) data warehouses focus on queries that collate, summarize, and analyze its contents. Sample data mining techniques in OLAP process include applying statistics, artificial intelligence, and machine learning techniques to find previously unknown or undiscovered relationships in the data [23]. These methods provide different perspectives from analytical techniques, in which the goal is to prove or disprove an existing hypothesis.

Spatial data mining is the process of discovering potentially useful patterns from large-scale geospatial datasets [24]. Discovering geospatial patterns from geospatial datasets is more difficult than recognizing the statistical patterns from traditional analysis object such as numeric and categorical data. The complexity of spatial data types, spatial relationships, and spatial autocorrelation is still an open problem to be solved.

This work seeks to optimize large-scale geospatial data handling by addressing the remaining open research problems regarding spatial data warehouse and spatial data mining that will most likely impact a future, large-scale enterprise system implementation.

2.6 Previous work

In the paper “From Databases to Dataspaces: A New Abstraction for Information Management [25],” the author highlights the demands of accessing data from anywhere and in any format. The author proposes the concept of “data space” to provide universal API and interface for any kind of data. Their work has inspired the industry’s successive Big Data concepts.

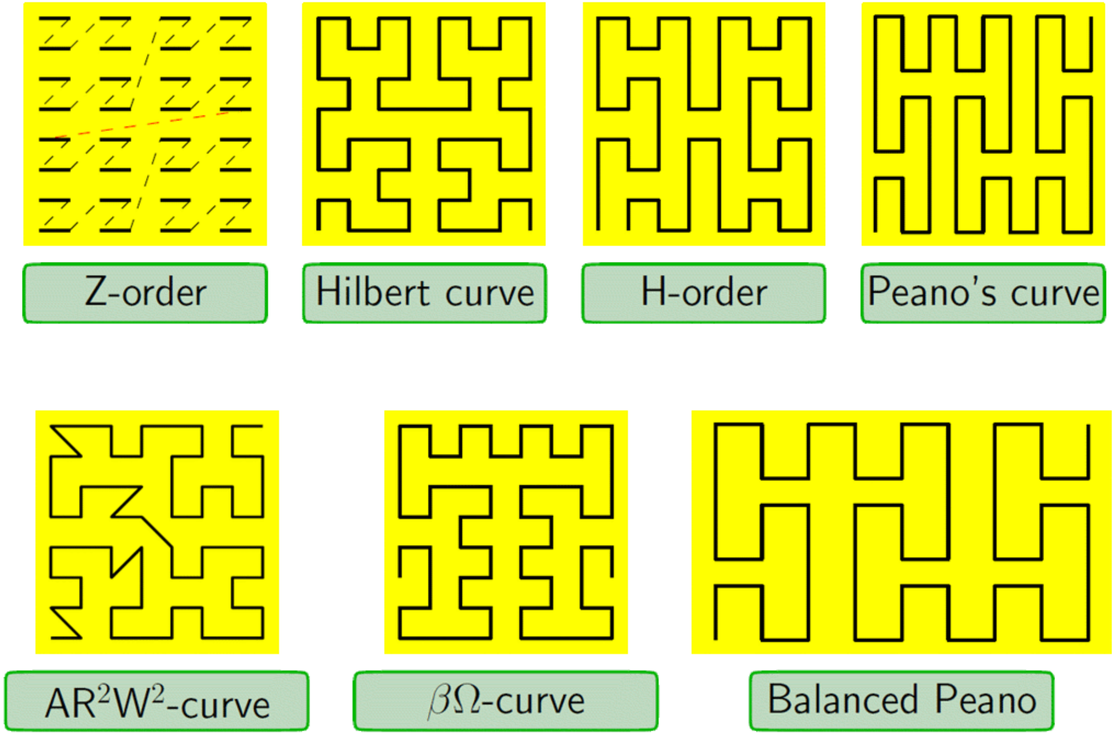


Figure 2-7. Space Filling Curves [26]

In “Digital halftoning with space filling curves [27],” the author presents a method to access POIs with space filling curves. However, this method focuses on algorithms in computing the average intensities of regions and determining the aperiodic dot patterns. These algorithms provide a solution for handling dispersed dot error. Still, the author has only scratched the surface of the problem. For example, we cannot yet say definitively whether space filling curves are better for querying geospatial data, or under what conditions they are better; nor do we understand the impact of space filling curves on the process of data mining or decision making.

In the paper “Efficient Spatial Query Processing for Big Data [28]”, the author defines a lightweight and scalable spatial index on Big Data. The result of this experiment shows that the index is both effective and efficient. The author defines some spatial Operators like containing, containedIn, intersects, and withinDistance. Such a system would provide a possible solution for improving the performance of spatial query on Big Data. In our framework, we use a similar design of spatial index which is built on Hbase to improve the performance of spatial query.

In “MAD Skills: New Analysis Practices for Big Data [29],” the authors propose the Deep data analytic method, referred to as Magnetic, Agile, and Deep (MAD). Their work is based on the Greenplum parallel database [29]. They provide algorithms that perform both SQL and MapReduce analysis on Big Data. The paper gives a general direction for analysis of Big Data. However, Greenplum is built on PostgreSQL which is on the Relational DBMS model. NoSQL is now the mainstream solution for Big Data analytics.

Their solution is not based on a NoSQL platform, like our solution.

The company Revolution Analytics [30] provides RHadoop project, which is the mainstream solution for Big Data analytics. The prototypes of current Big Data solutions like Google, Yahoo, etc. are based on a web system; they use Hadoop to store web pages and R to analyze customer behavior. Our research object, which is not based on web data but on geospatial data, is totally different from theirs. To use the R-Hadoop solution, a lot of customization and optimization work for geospatial data would need to be done. The RHbase package is also limited by the memory capability of the R client; therefore an improvement of spatial query for R is definitely needed for geospatial Big Data analytics to work. The RHbase package is the example of connectivity tools to access HBase through R. The RHbase is designed for the general type of Big Data not for the best performance of spatial query. Our work provides optimization for spatial query in R with HBase connectivity.

In “SpatialHadoop: towards flexible and scalable spatial processing using mapreduce [31]”, the authors propose the framework combined MapReduce and spatial data. Their work provides a new language called Pigeon [32] for spatial query and data mining support on Hadoop. The SpatialHadoop solution requires that the user be a professional programmer, who ought to know the technical details of MapReduce. The SpatialHadoop solution defined their own data mining function but missed the well-established R analysis functions. Our work comparing with SpatialHadoop is different; we chose R, which is popular for statisticians, as the solution for data mining and as a visualization tool. The mainstream data analytics solution, like Teradata Corporation Aster Analytics [33] which is a popular Big Data Analytics solution in the industry, chose a similar R-Hadoop platform to the one we have used, which has been proven to be robust in a real industry environment. Moreover, our in-memory spatial query is also designed and optimized for the R system. In conclusion, our work is designed as a solution with simplicity, robust and user-friendly for R analyst, business analysts and data scientists.

There is a commercial solution, called the SAS Intelligence Platform architecture [34], to perform the analytical processing of large-scale data. It also supports concurrent access by multiple users. However, this commercial solution is based on multi-tier machines and lacks the flexibility to achieve high-performance as Hadoop solution.

There are four tiers of machines in the SAS approach:

- Data sources -- The storage layer that supports various types of data sources.
- SAS servers -- The SAS server layer that performs the analytical processing. Each type of workload must map to a different SAS server.
- The middle tier -- The web interface for an analytical job
- The client tier -- The desktop client to generate analysis reports in a web browser environment.

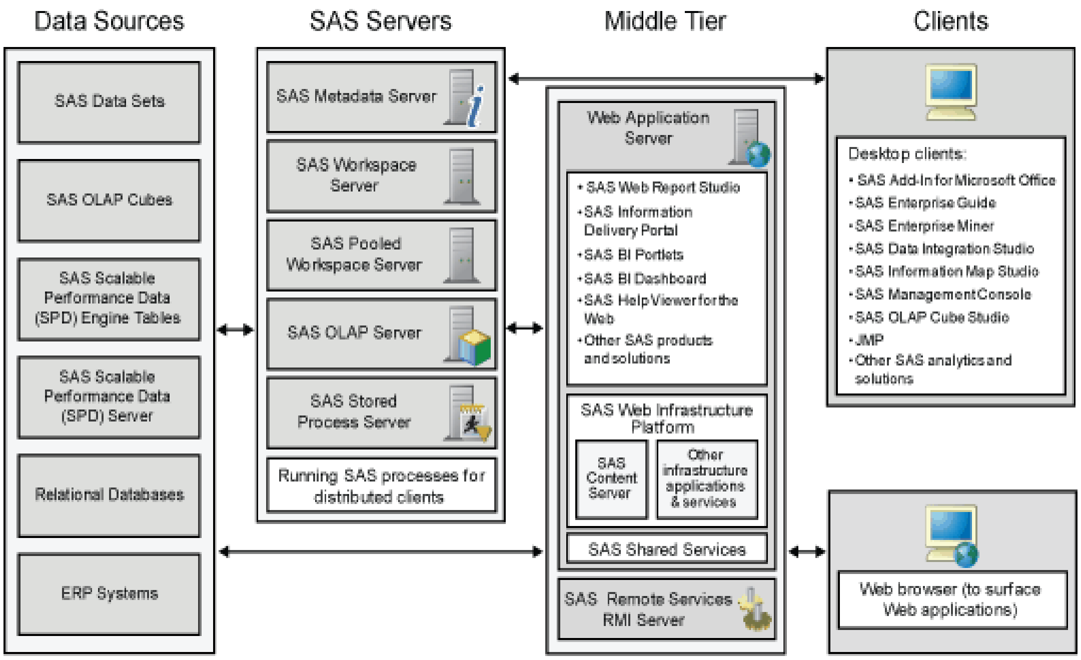

Data Sources
SAS Servers
Middle Tier
Clients
SAS Data Sets
SAS OLAP Cubes
SAS Scalable Performance Data (SPD) Engine Tables
SAS Scalable Performance Data (SPD) Server
Relational Databases
ERP Systems
SAS Metadata Server
SAS Workspace Server
SAS Pooled Workspace Server
SAS OLAP Server
SAS Stored Process Server
Running SAS processes for distributed clients
Web Application Server
• SAS Web Report Studio
• SAS Information Delivery Portal
• SAS BI Portlets
• SAS BI Dashboard
• SAS Help Viewer for the Web
• Other SAS products and solutions
SAS Web Infrastructure Platform
SAS Content Server
Other infrastructure applications & services
SAS Shared Services
SAS Remote Services RMI Server
Desktop clients:
• SAS Add-In for Microsoft Office
• SAS Enterprise Guide
• SAS Enterprise Miner
• SAS Data Integration Studio
• SAS Information Map Studio
• SAS Management Console
• SAS OLAP Cube Studio
• JMP
• Other SAS analytics and solutions
Web browser (to surface Web applications)

Figure 2-8. Architecture of the SAS Intelligence Platform [34]

In the SAS approach, the machines perform specific roles at different layers. Each machine must be installed and configured by the administrator. For large-scale data analysis, the user should add enough machines to handle the task. In contrast, Hadoop allows the autonomous management of its nodes. Moreover, a Hadoop cluster will automatically handle the workload management task. Therefore, the storage support of SAS approach is not as scalable as the Hadoop cluster.

The SAS solution provides several analytical methods such as the following procedures: KRIGE2D, SIM2D SPP, and VARIOGRAM for spatial analysis. User defined analytical packages are hard to add and implement in this commercial platform.

Currently, there are no commercial systems to support spatial data mining over a Big Data infrastructure. Data providers are mandated to use existing geospatial data systems such as traditional DBMS. All current systems support sophisticated geospatial and environmental constructs, such as Datums, Projections, Topology, and Grids by default. However, none of these systems support a state-of-the-art Big Data framework.

The RHadoop package, developed by Revolution Analytics, will provide a means to undertake this kind of analytical task. First, RHadoop provides various data sources such as file system, HDFS, and traditional DBMS. Second, R is not originally designed for processing Big Data analytical work. The analytical task of R runs in memory. However, there is a limit to the amount of memory R can access. RHadoop provides a workaround that allows R to support parallelism with acceptable performance [35].

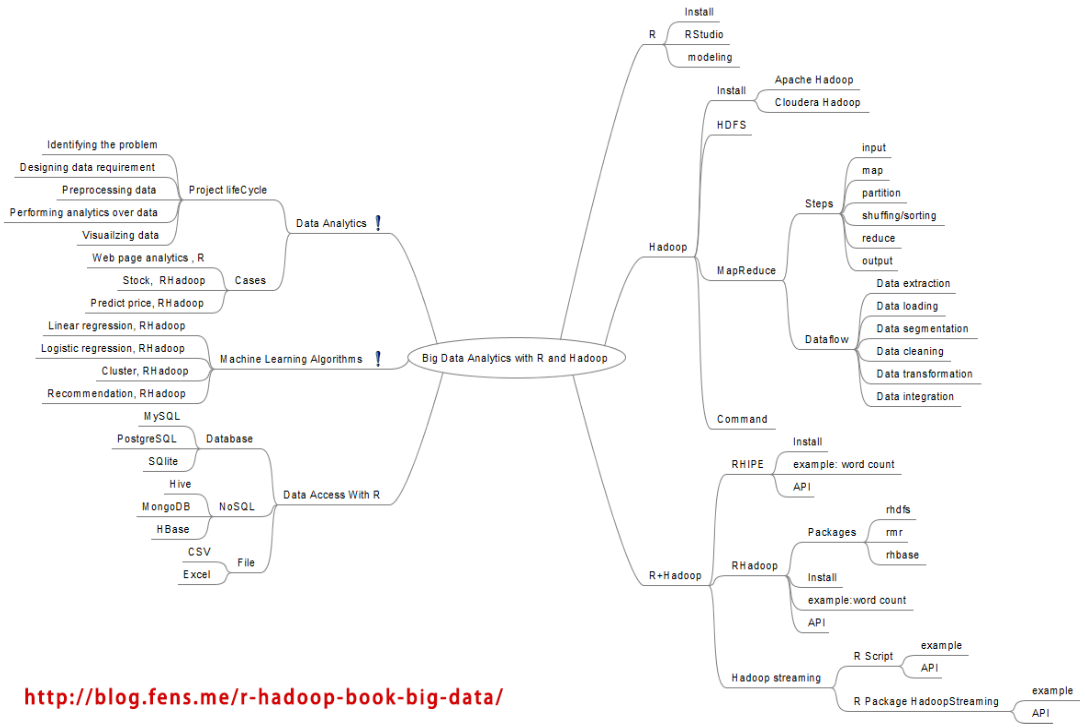


Figure 2-9. R and Hadoop [36]

The solution of Big Data Analytics has two advantages other than its analytical framework. First, they define a simple analytical model on Big Data, which does not fit the current statistical analysis software. The model also maintains high performance and accuracy with large-scale data. Second, the solution supports user defined analytical packages to avoid expensive commercial solutions.

Revolution Analytics provides several R packages for the Big Data analytics solution. For example, the RHipe package merges the environment of R and Hadoop, which allows users to divide data into subsets and form multiple divisions. The Rmr package is used to perform MapReduce [37] tasks between R and the Hadoop environment. The RProtoBuf package is used to load serialized data from other MapReduce [37] environments and communicate among different MapReduce jobs.

R is a good choice for Big Data analytical tasks, and Hadoop is a popular solution for Big Data infrastructure. As a result, in this framework we have chosen R as the spatial data mining tool to analyze geospatial Big Data.

CHAPTER 3 THE FRAMEWORK

3.1 Framework Overview

Hadoop has gained significant acceptance in the Big Data user community. However, none of the existing Big Data frameworks supports spatial data mining over large-scale geospatial data. In this project, we have developed a method which makes it possible to integrate spatial data mining technology with geospatial data. We outline the process of analyzing large-scale geospatial data using the R language. The resulting data is compatible with existing Big Data solutions and compliant software.

The Hadoop software provides the data storage and access capabilities for the geospatial data used in this framework. Our primary goal is to support retrieving large-scale geospatial data. Environmental data is represented as both gridded data and vector data. The R language provides spatial data mining capabilities for geospatial data as well as commercial products. As such, it is a perfect solution for the large-scale geospatial data analytical environment. However, as mentioned above, the current R environment does not provide any analytical processing method for Big Data due to the limit of R. As a result, a vital performance issue occurred when using R to load large-scale raw data: adding large-scale data into R usually results in a system halt or low system performance.

Some of the current open research problems noted by the spatial data warehouse with spatial data mining tools that are especially relevant to an enterprise system for large-scale geospatial data are:

- R analytical function runs in memory and has limits for data frame size;
- No spatial index definition on HBase to improve query performance;
- No standard mechanisms to access geospatial grid files stored on the Hadoop Distributed File System (HDFS);
- No clear understanding of how to run a spatial data mining analytical package through R to access geospatial data stored on Hadoop;
- No standard representations that support ETL process of homogeneous and heterogeneous geospatial data across different sources;
- No broadly applicable approaches for dealing with spatial data mining packages.

Substantial work has been done by this work in the large-scale spatial data mining area that was detailed in the existing work section, as well as developed to show the methodology selected and technical approach of this work, that builds upon substantial progress in recent years regarding large-scale spatial data mining.

Our aim is a new approach to the spatial data mining analytic method for large-scale geospatial data. As discussed above, Hadoop provides a data infrastructure to store and process geospatial data. Combining Hadoop with the existing R system provides a comprehensive capability to analyze and share environmental data without removing its associated variable information. The following objectives are set for our framework:

1. Provide a distributed method to run statistical computation among large-scale test data;
2. Improve large-scale data querying performance;
3. Extend R application to large-scale data;
4. Provide various data sources and methods for geospatial data analysis；
5. Support a wide variety of data formats from different data sources;
6. Easy to define, control, manipulate, and manage the asymmetrical data; sources through unique system interface;
7. Provide various analytical methods for a geospatial data analyst, with the ability to return a visualized analytical result;
8. Efficient data storage management;
9. Provide a reliable, available, and scalable service.

Here is the design of our framework:

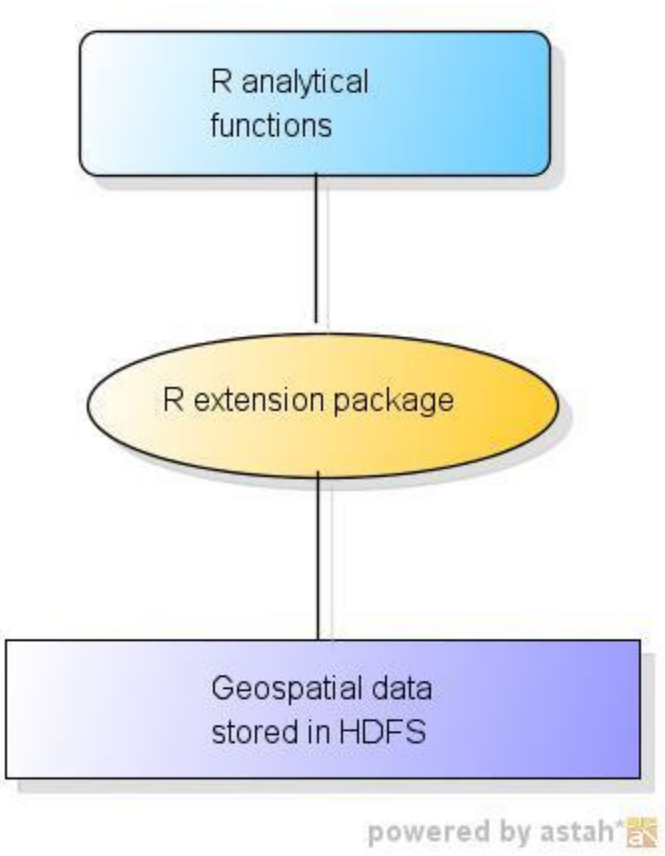


Figure 3-1. Geospatial Big Data analytical framework

In this project, we have developed a framework to implement spatial data mining over Big Data infrastructures.

The next chapter describes the implementation of these capabilities using an analytical model. We then expand to three use cases and describe our overall environmental data analytical architecture implementation.

3.2 Bi-Directional Spatial ETL Server

The purpose of this framework is to store and analyze large volumes of geospatial data. The first analytical step is to store the geospatial data in the spatial data warehouse. The second analytical step is to run ETL (Extract, Transform and Load) over the raw data. The third analytical step is to perform analytical functions by spatial online analytical processing (SOLAP) environment. To design the spatial data warehouse, we built conceptual physical and logical models, effective spatial indexes, SOLAP, and other analytical functions.

We used Hadoop as the platform for the data warehouse. The spatial query HPC works as one component of the ETL server. The basic function of spatial ETL is:

- Use HBase as Spatial Data Warehouse
- Extract geospatial data from different geospatial data sources, especially grid files; The data maybe in homogeneous or heterogeneous format;
- Transform the geospatial data to a universal format which fits HBase;
- Load the geospatial data into the HBase.

The traditional ETL concept is one-way, which extracts, transforms, and loads data from the source database to the data warehouse. We designed an infrastructure called "Bi-Directional ETL" which can perform ETL work between a public cloud and a private cloud. For instance, The Figure below shows a public cloud for a large corporation and a private cloud for a specific department within the corporation.

Public Cloud:	Corporation Cloud
Private Cloud:	Ocean Investigation Vessel Cloud

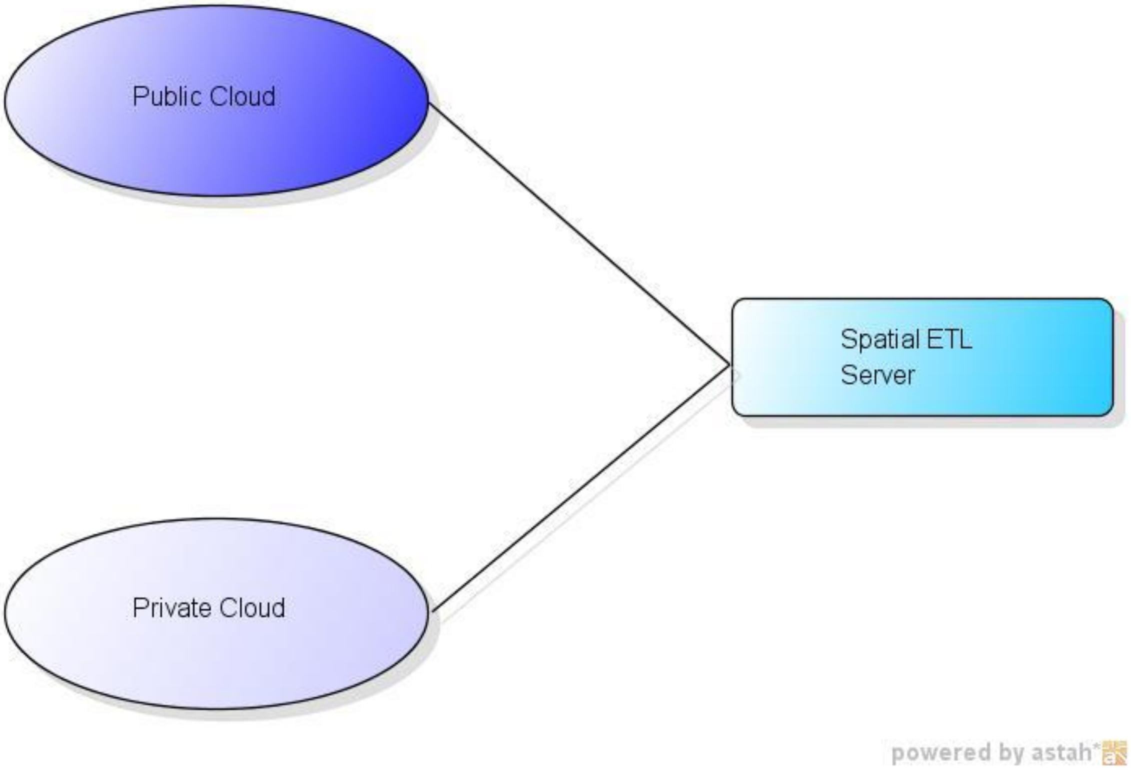


Figure 3-2. Bi-Directional ETL

The ELT process can be Bi-Directional.

- Public to Private: Vessel downloads (ETL) the navigation data for a specific area to the private cloud
- Private to Public: Vessel uploads (ETL) the ocean investigation data to the public cloud

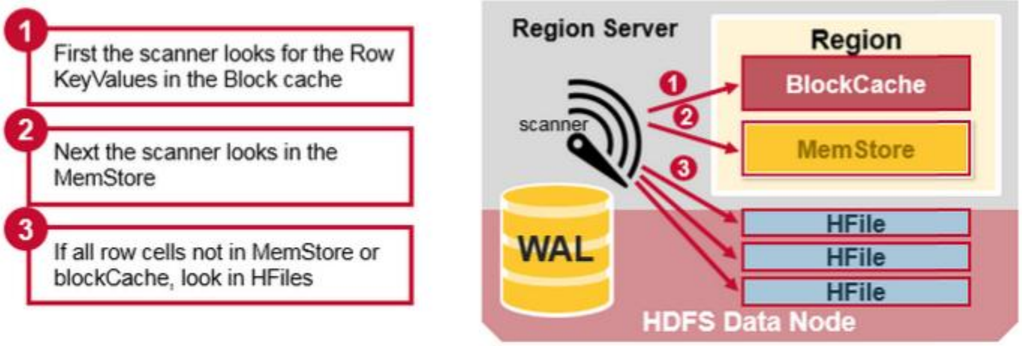
Both the Public and Private clouds can utilize the ETL server to process the geospatial data, which means the ETL process, works bi-directionally.

3.3 In-Memory Spatial Index and Spatial Query

We choose HBase as the platform for the Spatial Data Warehouse. The spatial index is an efficient method for indexing geospatial data. To ensure the performance of spatial query we chose to build the spatial index in memory.

Comparing random read versus sequential read is one way of assessing database query efficiency. Sequential read allows DBMS to access large amounts of data from adjacent locations on the physical disk. Random read allows DBMS to access data which can be accessed in any sequence.

The read path of HBase is shown in the following figures.

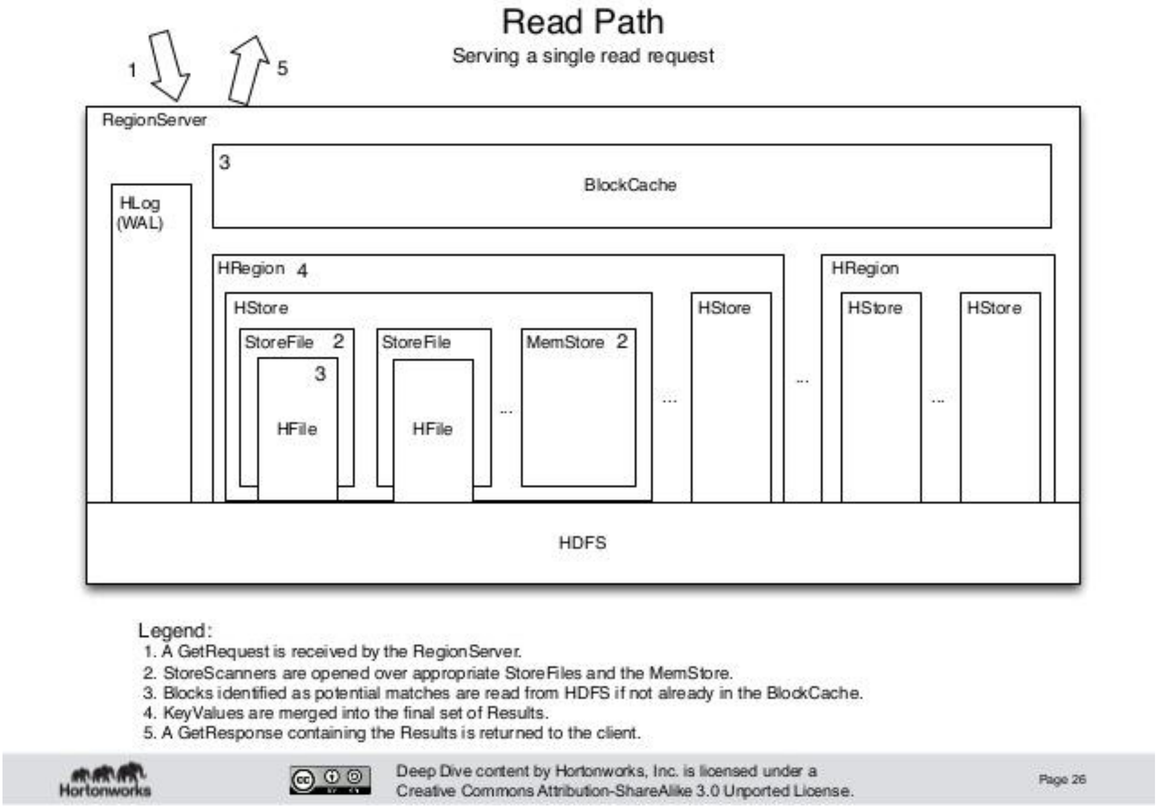


1 First the scanner looks for the Row KeyValues in the Block cache

2 Next the scanner looks in the MemStore

3 If all row cells not in MemStore or blockCache, look in HFiles

Figure 3-3. HBase Read Path [38]


Read Path

Serving a single read request

Legend:

1. A GetRequest is received by the RegionServer.
2. StoreScanners are opened over appropriate StoreFiles and the MemStore.
3. Blocks identified as potential matches are read from HDFS if not already in the BlockCache.
4. KeyValues are merged into the final set of Results.
5. A GetResponse containing the Results is returned to the client.

Hortonworks

Deep Dive content by Hortonworks, Inc. is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.

Page 26

Figure 3-4. HBase Read Path Detail [39]

HBase has two kinds of caches: memory store and block cache. The performance of sequential read relies on cache hits ratio. When HBase runs a query it will firstly look at the block cache or memory store, then HFiles. The block cache or memory stores are all in the memory cache. During the HBase performance evaluation experiment [40] sequential read performed better than random read.

We used in-memory spatial index for the spatial data warehouse (HBase). The spatial data warehouse is designed to store historical data and run read-only analytical functions. Most administrators of a data warehouse perform “regular” incremental load every one to three months. Therefore, even if ETL process takes about two or three days which is still considered acceptable.

Our design has no physical index stored on HBase. The geospatial raw data is stored as a grid file, and each grid file contains POIs in a specific area. This means the cluster ratio of the geospatial raw data is very high. The best query performance to access the HBase record is to use rowkey. The distribution of HBase data is determined by the rowkey generation function. The hashes as rowkey will ensure the best spread of the data. The sequential keys as rowkey will ensure the locality of the data. When HBase locates the first row of geospatial data, the following read has a high chance of being a sequential read. To get a sequential read in range scan mode, the rowkey of the data must be sequential. In our use case in chapter 5.1, we chose pre-sorting grid files as data sources. The generated sequential keys ensure that the range scan of HBase performs as sequential read. This experiment [40] shows that the performance of sequential read is better than random read in HBase. So we have chosen to build the spatial index in memory. The process of spatial query is depicted in the following figure.

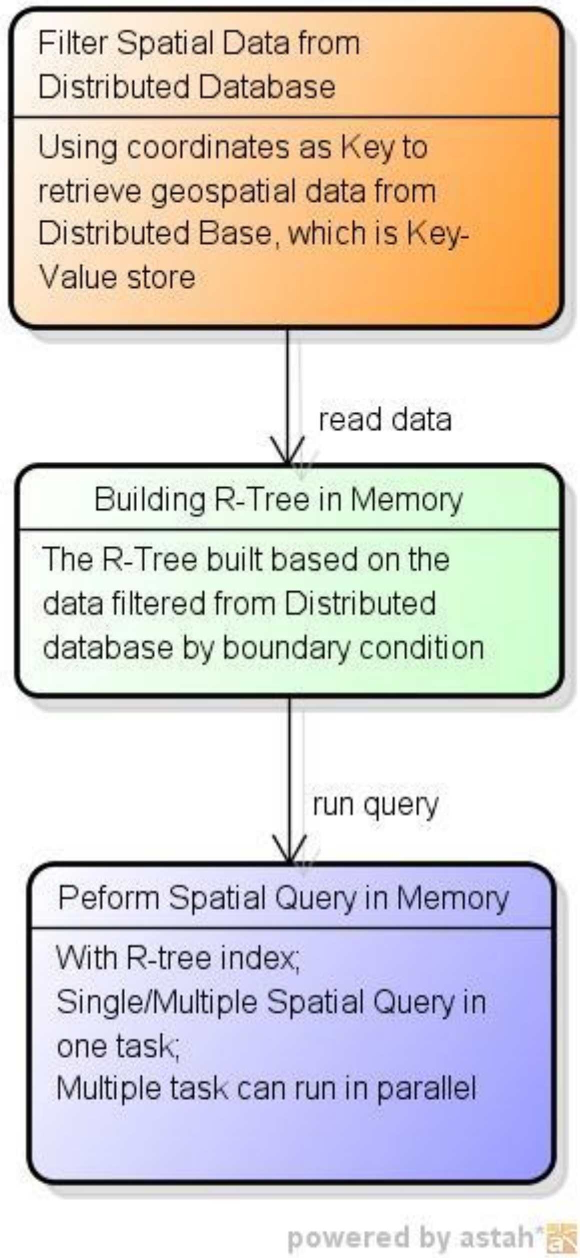

Filter Spatial Data from Distributed Database
Using coordinates as Key to retrieve geospatial data from Distributed Base, which is Key-Value store
read data
Building R-Tree in Memory
The R-Tree built based on the data filtered from Distributed database by boundary condition
run query
Peform Spatial Query in Memory
With R-tree index;
Single/Multiple Spatial Query in one task;
Multiple task can run in parallel
powered by astah*

Figure 3-5. In-Memory R-Tree & Spatial Query

The first step of spatial query is to filter the spatial data from HBase. Users should assign four coordinates such as upper left, upper right, lower left, and lower right as boundary conditions of the spatial query. HBase will retrieve this area of geospatial data as a data window. The second step is building spatial index in memory. The R-tree is built based on the data filter from HBase by the boundary conditions at the first step. The last step is to run spatial query in memory. The spatial query, which can access both the data and spatial index, can be single or multiple in one process.

The design methods of this framework are a combination of scale-out (horizontal) and scale-up (vertical). Hadoop, in our solution, works as a scale out storage system: it expands the scalability of geospatial data storage. Spatial query HPC works as a scale up computation system: it accelerates the performance of the ETL process. Designing computer architecture is an art; one must find the balance between cost, speed, and performance. The basic rule is to save money on the Hadoop data node, which is used as the storage system. On the other hand, users should invest more on spatial query HPC because the spatial query task is memory intensive and data intensive.

CHAPTER 4 FRAMEWORK IMPLEMENTATION

4.1 Framework Structure

The Hadoop software provides the core data accessing capabilities for the data used in this work. Our primary goal is to support accessing large-scale geospatial data. Geospatial data is represented as both gridded and vector data. The R software product provides spatial data analytical capabilities for geospatial data as well as the solutions provided by SAS. As such, it is a perfect solution for large-scale geospatial data in an analytical environment. However, as mentioned above, it does not provide an analytical method for Big Data. As a result, a severe performance bottleneck problem occurs when the R software is used to load large-scale raw data. Adding that large-scale data back into R usually results into low system performance or even system halt.

We chose various methods to access the data stored in HDFS. The first method is the spatial data mining package SPATSTAT [41], which can analyze raw geospatial data. The second method is the package RHbase, which provides a path to access data stored in the Hadoop database. The third method is R-MapReduce packages. The programmer can perform MapReduce tasks to HDFS objects directly through the packages Plyrmr or Rmr2.

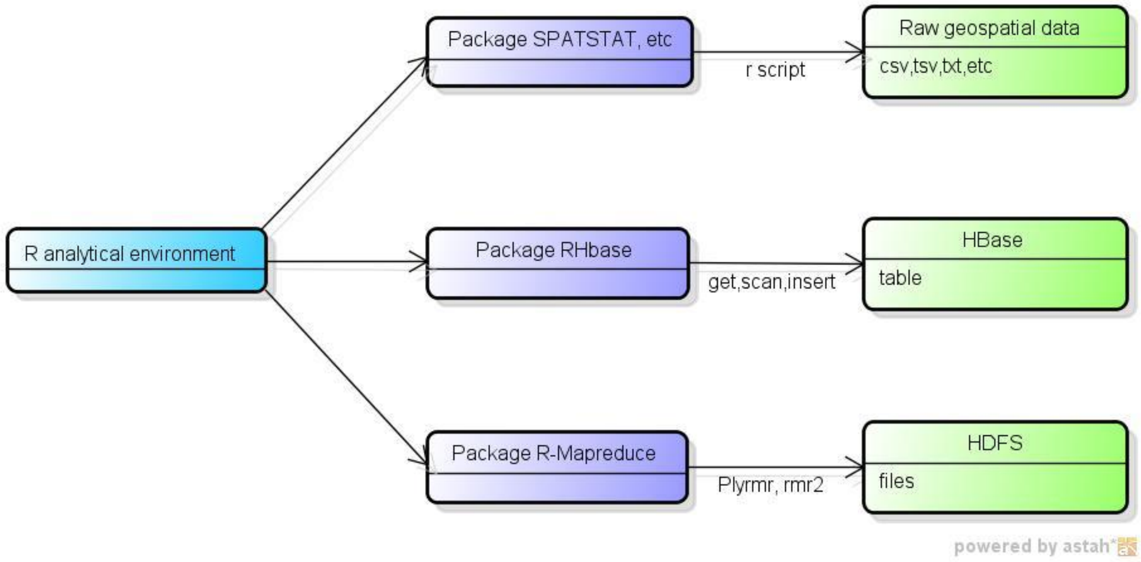


Figure 4-1. Analytical environment structure

4.2 R with package Spatstat

We chose Spatstat as the R package to perform statistical analysis, especially for spatial point patterns. Spatial point patterns can be stored as two-dimensional data formats. Spatstat packages use various analytical methods to discover useful data patterns in large-scale spatial data. Compared to common large data, geospatial data is hard to extract exact patterns from, because of the nature of specific geospatial data sources and its associated data structures.

The package Spatstat uses real numbers (e.g. geospatial POIs with uncertainty information), categorical values (e.g. fishery production by species) and logical values (e.g. saline water/freshwater) to mark the point patterns [41]. These point patterns are capable of analyzing a huge number of points. The region of spatial data can be a complicated shape, such as an arbitrary polygon or a binary pixel image mask. The package Spatstat is capable of analyzing three or more dimensional point pattern datasets. The following are the data formats which Spatstat can analyze:

- Two dimensional space data regions;
- Pixel images in two-dimensional space;
- Spatial patterns of line segments in two-dimensional space;
- Tessellations in two-dimensional space [41].

The package Spatstat supports a variety of statistical analysis methods such as model fitting, spatial data sampling, and statistical formulation. The package provides methods to perform Gibbs point process models, spatial inhomogeneity, and dependence analysis and Cluster process models.

We can use maximum likelihood or approximations as frequentist statistical methods to run a point process fit model such as Poisson point process models, Gibbs point process models, and random cluster process models. Spatial randomness is a Poisson process, which is measured by the point's intensity. Random and uniform distribution of points is called a homogeneous Poisson point process. Unevenly distributed intensity is called an inhomogeneous Poisson point process. Thus, the spatial model can be spatially homogeneous or inhomogeneous. The spatial trend is modeled as functions of the Cartesian coordinates and spatial covariates. Clustering or repulsion interaction can be included and marked as Gibbs models.

Here is the sample code. The following example covers loading geospatial raw data into R. The example provides a variety of ways to create a point pattern as the object of package Spatstat. Chapter 5.1 shows the detail of an analytics function on point patterns.

```
> library(spatstat)
# Here is a simple recipe to create a point pattern(ppp) object from raw
data in R.

#geo_data$"poi_cf:latitude"
#geo_data$"poi_cf:longitude"
#geo_data$"poi_cf:mean"
#geo_data$"poi_cf:standarddeviation"

> east <- geo_data$"poi_cf:latitude"
> north <- geo_data$"poi_cf:longitude"
#create a point pattern from (x, y) and window information
> X <- ppp(east, north, c(-130.0000, -126.0000), c(49.0000, 51.0000))
#print basic description of point pattern X
> X
planar point pattern: 1000 points
window: rectangle = [-130, -126] x [49, 51] units
>                 df=                data.frame(geo_data$"poi_cf:mean",
geo_data$"poi_cf:standarddeviation")
> colnames(df) <- c("mean", "standarddeviation")
Get/> X <- ppp(east, north, c(-130.0000, -126.0000), c(49.0000, 51.0000),
marks=df)
…
```

4.3 R with Hadoop database (RHbase solution)

We used a method called RHbase to access the Hbase data. The package RHbase accesses the Hadoop database via the Thrift server. The table stored in HBASE can be browsed, read, written into, and modified through the RHbase package. Package RHBase, which works like ODBC and JDBC, allows R to access HBase table. However, the current RHBase version does not support parallel programming. The ability of an analytical task is limited by the system resources of R client. In our framework, for RHbase we put R in a large-memory server to ensure the capability of analytical tasks. We also provide coding examples in the appendix chapter to show the methods which permit the writing of multiple spatial queries in one program.

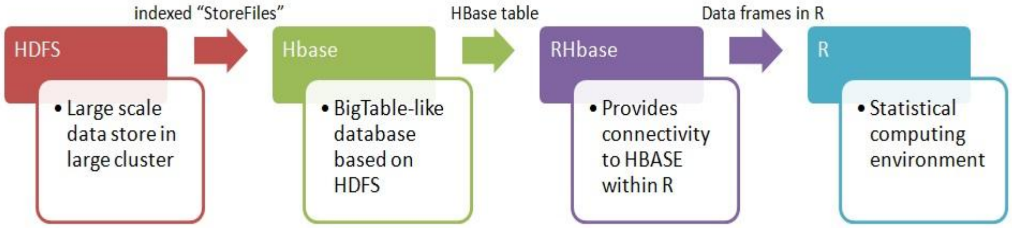

indexed "StoreFiles"
HBase table
Data frames in R
HDFS
• Large scale data store in large cluster
Hbase
• BigTable-like database based on HDFS
RHbase
• Provides connectivity to HBASE within R
R
• Statistical computing environment

Figure 4-2. R and HBase

The following functions are part of the Hbase package:

Table Manipulation	hb.new.table, hb.delete.table, hb.describe.table, hb.set.table.mode, hb.regions.table
Read/Write	hb.insert, hb.get, hb.delete, hb.insert.data.frame, hb.get.data.frame, hb.scan, hb.scan.ex
Utility	hb.list.tables
Initialization	hb.defaults, hb.init

In the following example, there are statements that define the connection to HBase through the RHbase package. So, R can perform spatial analytical functions over data stored on Hbase.

The approach also contains an example in chapter 5.1 about full table scan and specific row retrieval. Here is the example to create, insert, delete, and query geospatial data through the RHbase function.

```
#rhbase function
hb.new.table  hb.insert hb.scan.ex hb.scan
~ R
> library(rhbase)
#initialize hbase connection
> hb.init()
   <pointer: 0x8e548e8>
   attr(,"class")
   [1] "hb.client.connection"
#create new table in hbase
 #>
hb.new.table("geos_rhbase","poi_cf",opts=list(maxversions=5,x=list(maxv
ersions=1L,compression='GZ',inmemory=TRUE)))
  [1] TRUE
> hb.list.tables()
   $student_rhbase
     maxversions compression inmemory bloomfiltertype bloomfiltervecsize
```

```
    info:          5        NONE    FALSE           NONE                0
          bloomfilternbhashes blockcache timetolive
    info:                 0      FALSE        -1
 > hb.describe.table("geos_test")
        maxversions       compression       inmemory       bloomfiltertype
bloomfiltervecsize
poi_cf:          3        NONE    FALSE           NONE                0
        bloomfilternbhashes blockcache timetolive
poi_cf:                 0      TRUE        -1
#insert POI to hbase table
>
hb.insert("geos_test",list(list("row2",c("poi_cf:latitude","poi_cf:long
itude","poi_cf:mean","poi_cf:standarddeviation"),
list(49.0000,130.0000, 2698,26.98))))
    [1] TRUE
> hb.get('geos_test','row2')
[[1]]
[[1]][[1]]
[1] "row2"
[[1]][[2]]
[1] "poi_cf:latitude"         "poi_cf:longitude"
[3] "poi_cf:mean"             "poi_cf:standarddeviation"
[[1]][[3]]
[[1]][[3]][[1]]
[1] 49
[[1]][[3]][[2]]
[1] 130
[[1]][[3]][[3]]
[1] 2698
[[1]][[3]][[4]]
[1] 26.98
#ONLY If you want to clean the table
> hb.delete.table('geos_test')
    [1] TRUE
# scan from beginning
>                                 iter                                  <-
hb.scan("geos_test",startrow="row2",end="row2",colspec="poi_cf")
> while( length(row <- iter$get(1))>0){ print(row)}
> iter$close()

> geos_data <- c()
> iter <- hb.scan("geos_test", startrow="row2",
end="row2",colspec="poi_cf")
> while(length(row <- iter$get(1))>0){data <- c(data, row)}
> iter$close()


# get data frame, note the columns have ":" appended
iter <- hb.get.data.frame("geos_test",start="1")
iter()
# Uptil penultimate row
iter <- hb.get.data.frame("geos_test",start="1",end="87001")
iter()



# get data frame, note the columns have ":" appended
iter <- hb.get.data.frame("geos_test",start="1")
iter()
# Uptil penultimate row
iter <- hb.get.data.frame("geos_test",start="1",end="87001")
iter()



# scan all data (1-87001)
> iter <- hb.scan("geos_test",startrow="1",end="87001",colspec="poi_cf")
> while( length(row <- iter$get(1))>0){ print(row)}
> iter$close()

# load data through RHBASE
source("/home/zyang/rhbase_proj/rhbase_cmd_0804.txt")

#create a dataframe from a scan
hb.scan.data.frame <- function( tablename, startrow, end=NULL,
colspec,sz=hb.defaults("sz"), usz=hb.defaults("usz"),
                              hbc=hb.defaults("hbc") )
{
  scn <- hb.scan( tablename, startrow, end, colspec, sz, usz, hbc )
  f <- scn$get()
  get_column_index_values <- function( column_index )
  {
    get_value <- function( row, column_name )
    {
      indices <- which( row[[ 2 ]] == column_name )
      index <- ifelse( length( indices ) == 1, indices[[ 1 ]], 0 )
      ifelse( index == 0, NA, row[[ 3 ]][[ index ]] )
    }
    column_name <- cols[[ column_index ]]
    unlist( lapply( f, get_value, column_name ) )
  }
  #get the vector of columns from the first row
  cols<-f[[1]][[2]]
  df <- as.data.frame( lapply( 1:length(cols), get_column_index_values ) )
  rownames( df ) <- unlist( lapply( f, "[[", 1 ) )
  colnames( df ) <- cols
  df
}
```

4.4 R with Hadoop database (In-memory extension solution)

A distributed database is more user-friendly than a distributed file system. On the contrary, it may not be as efficient as a distributed file system. If a user wants to perform global level geospatial computation, a distributed file system (not a database) plus MapReduce is a better solution. However, for some specific users, for example meteorologists living in New Orleans, their routine work focuses on localized data. In this case, the dedicated spatial query server on a distributed database will be the convenient choice.

HBase has many advantages when dealing with large-scale geospatial data. There are two features that make the ETL work more efficiently in our framework: one is MapReduce BulkLoad, the other is In-Memory Spatial Index.

We used these features to serve as the ETL (Extract, Transform and Load) tool.

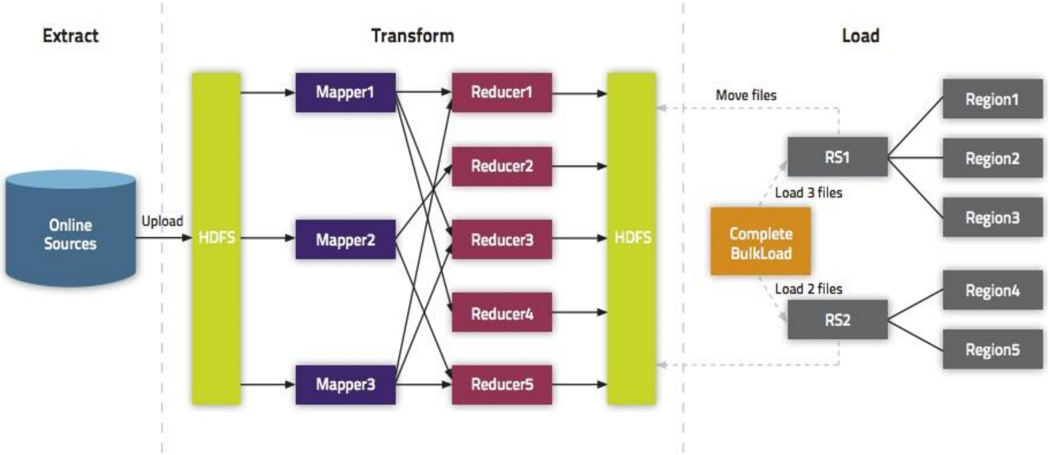


Figure 4-3. Hbase and BulkLoad[42]

HBase supports BulkLoad, such as Hadoop streaming, which allows the programmer to write utilities code as mapper/reducer on Hbase. As figure 4-3 shows, the raw data has been extracted from online sources to HDFS. Then BulkLoad utility uses the MapReduce task to load the delimiter separated values data to the Hbase table or to the HFiles stored on parallel regional servers.

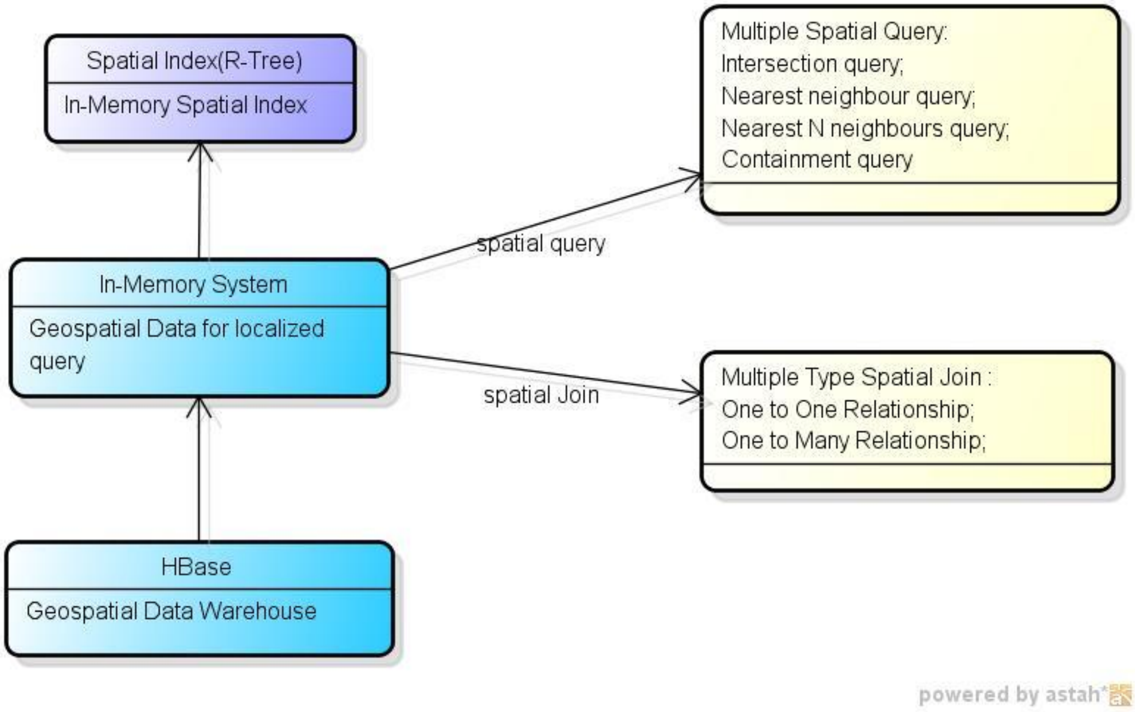


Figure 4-4. Hbase and Spatial Index

Hbase, which is a distributed database, provides high performance access method by row key. For very large data query, range scan by rowkey will be the most effective way [43]. This motivated our decision not to define a global spatial index on Hbase.

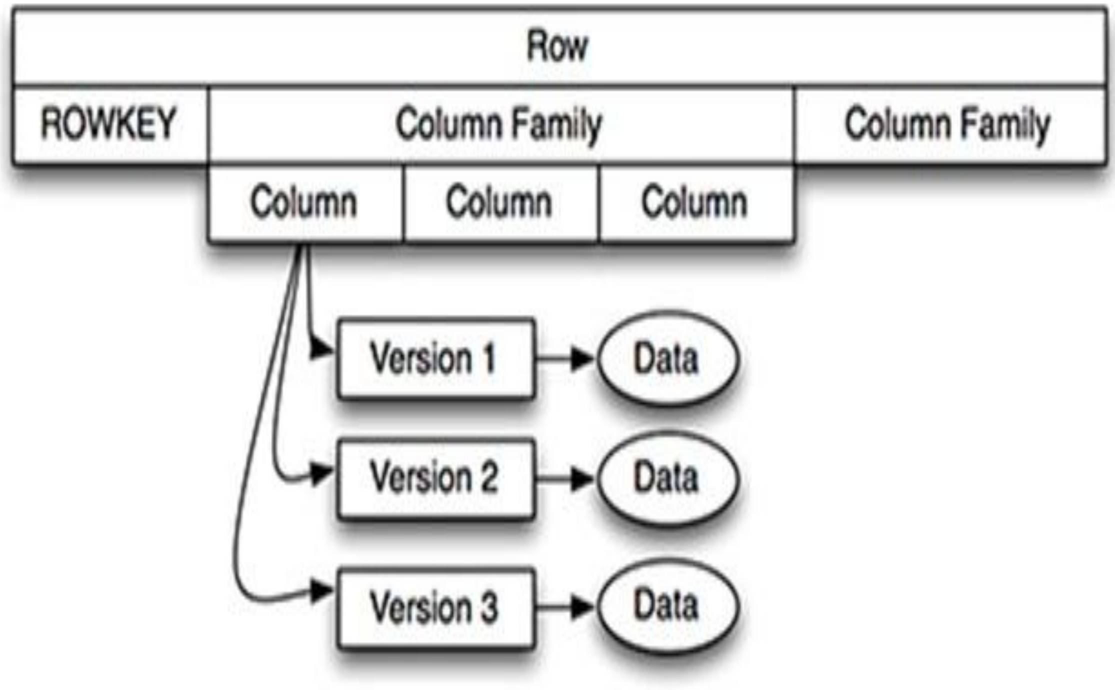


Figure 4-5. Hbase and Spatial Index[43]

Geospatial files are loaded as Grid files. The rowkey is generated by the coordinates. In the grid file POIs are sorted sequentially. That implies if two POIs are adjacent in a map, their generated rowkey are also adjacent in the key-value pairs list. Hbase scans the table by rowkey. In this distribution, the data fetch process will be a sequential read in high performance.

We chose to design an in-memory system to support Spatial Index, Spatial Query, and Spatial Join features. We used a JSI (Java Spatial Index) [44] library, which allows the user to create an R-tree spatial index on geospatial data. With the R-tree index, the user can easily perform customized geospatial queries such as a Boundary query, Range Query, and kNN(nearest neighbor) Query. In this framework, we also applied the Spatial

Join feature by using Geophile [45], which is a Java implementation of spatial join. The following figure shows the method of HBase key design.

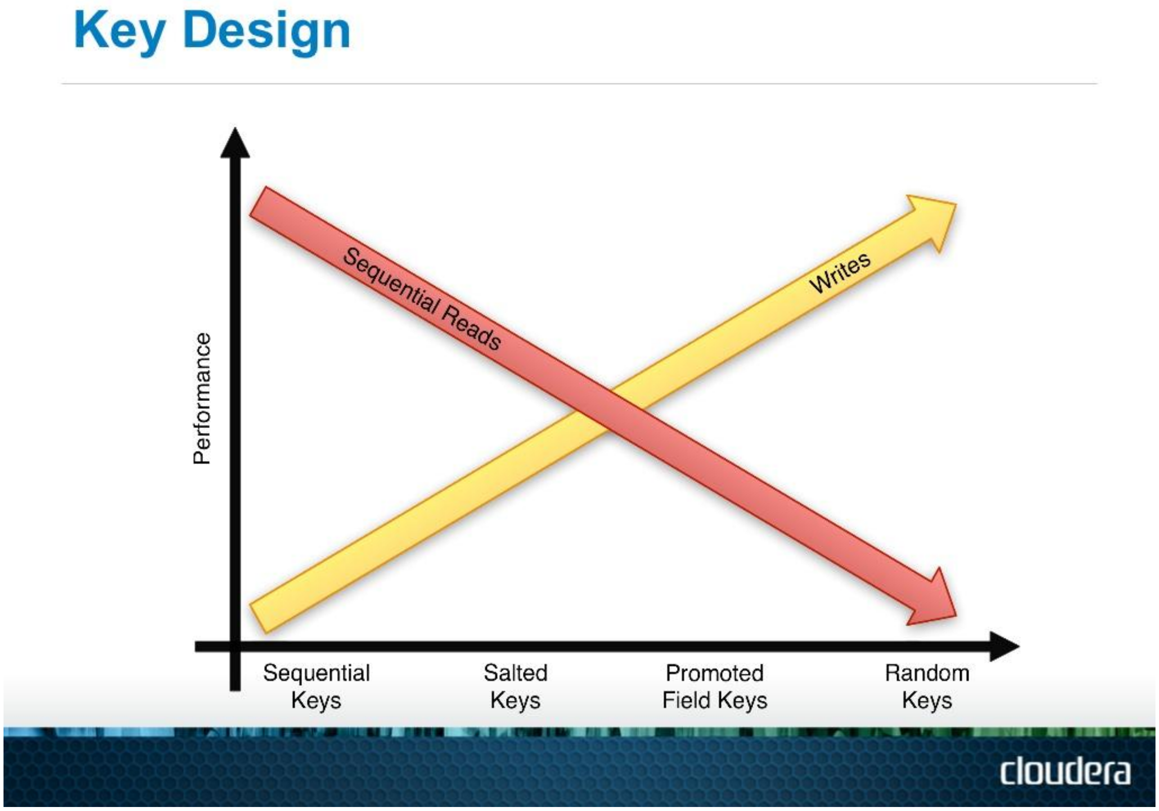


Figure 4-6. Hbase Rowkey design [46]

In this work, we add in-memory spatial index on Hbase to support Spatial Queries and Spatial Join capabilities. Our design has no physical index stored on HBase. In our design, we chose "range scan by rowkey" as the method to query HBase in high performance. The database administrator chooses a specific function to generate the rowkey by coordinates of POIs. The HBase user should use the same function to specify the range of rowkeys to retrieve the data. The range scan allows the HBase to locate the specific range of rowkeys to avoid a full table scan. Instead, the query will be randomly read if the

HBase user did not use rowkey to retrieve HBase data. The performance of sequential read in range scan is, therefore, better than random read on HBase. We defined a geometric bounding box to fetch the geospatial data. Based on the result of bounding box retrieval on HBase, we chose to build the spatial index in memory on a dedicated spatial query server. The In-Memory database works as a "Cache" system to ensure the performance of localized spatial query.

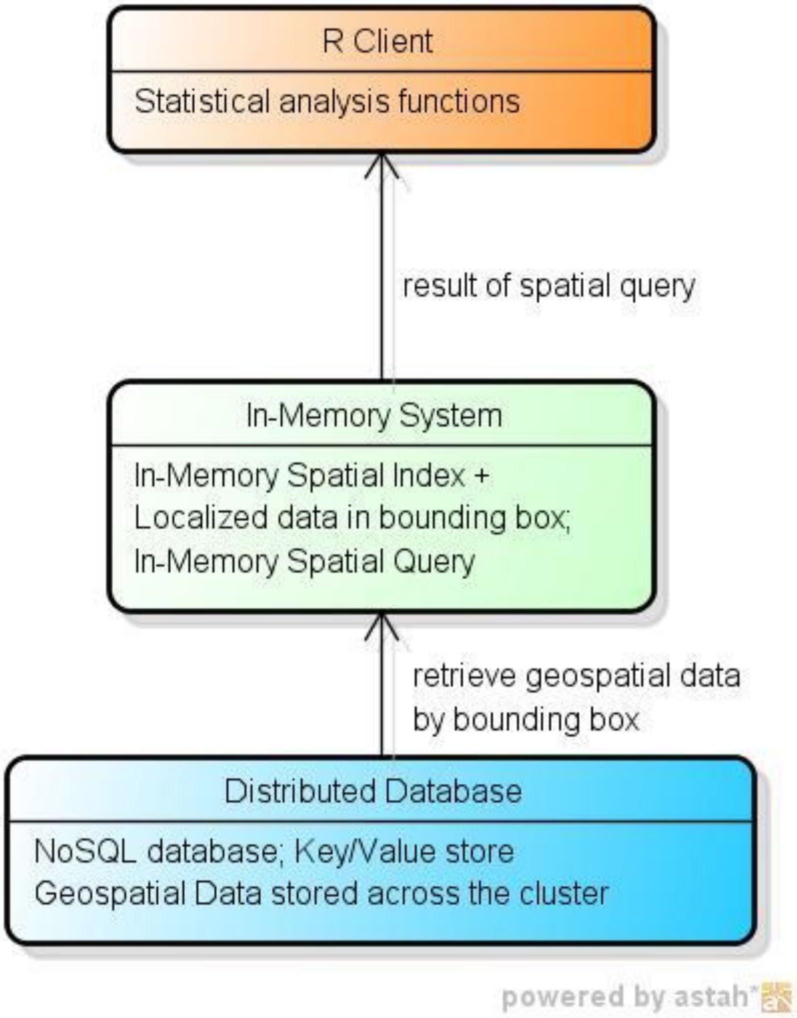

R Client
Statistical analysis functions
result of spatial query
In-Memory System
In-Memory Spatial Index +
Localized data in bounding box;
In-Memory Spatial Query
retrieve geospatial data
by bounding box
Distributed Database
NoSQL database; Key/Value store
Geospatial Data stored across the cluster
powered by astah*

Figure 4-7. Distributed/In-Memory heterogeneous architecture to perform In-Memory spatial Query

Spatial Index, Spatial Query, and Spatial Join features improve the performance of processing large-scale geospatial data. A detailed use case can be found in chapter 5.1.

4.5 R with Map-Reduce

Package Plyrmr & Rmr2[47] allow R developers to use the MapReduce programming model, developed as part of the RHadoop project.

4.5.1 R with package Plyrmr

The existing packages Plyr and Reshape2 can perform many common manipulation operations on very large data sets stored on Hadoop, and they can use the MapReduce model to perform their tasks. The Plyrmr package also provides a familiar Plyr-like interface to hide MapReduce details. The Plyrmr package is based on Rmr2 but has a more convenient, user-friendly interface. The Plyrmr package provides the following functions.

Transmute	Selects all data plus summaries
bind.cols	adds new columns
Select	select columns
Melt	converts between long and wide data frames
Dcast	converts between long and wide data frames
Gapply	takes any function that accepts and returns data frames
magic.wand	gives Hadoop powers to many functions in dplyr
(table continued)	

Group	takes an input and a number of arguments that are evaluated in the context of the data, exactly like bind.cols
group.f	grouping relative of gapply
gather	grouping recursively

4.5.2 R with package Rmr2

The Rmr2 package allows an R programmer to perform statistical analysis via MapReduce on a Hadoop cluster. For global level geospatial computation, task our framework puts the grid files on HDFS and uses MapReduce as a programming model.

The Rmr2 packages can perform MapReduce operations on very large data sets stored on Hadoop. All the nodes should install the Rmr2 package. These packages also provide familiar functions like the combination of R package lapply and a tapply. The function lapply will apply a function over a list or vector, and the function tapply will apply a function over a ragged array.

Here is an example of a MapReduce process:

```
small.ints = to.dfs(1:100)
MapReduce (
  input = small.ints,
  map = function(k, v) cbind(v, v^2))
```

Figure 4-8.sample MapReduce script [48]

The input statement will put the data to HDFS. The bulk of the data is stored for MapReduce to manipulate. However, Big Data cannot be written out with to.dfs. We used to.dfs to put data in the pre-defined or temporary file in HDFS. The to.dfs returns a

Big Data object and can be assigned to variables, Rmr functions, or MapReduce jobs. So, the user can access Big Data that R cannot handle. R has a memory limit for the data frame to process. Therefore, using the R-MapReduce function, R can handle Big Data which exceeds the memory size limit.

The second statement uses MapReduce to replace function lapply. The output of to.dfs will work as the input variable small.ints, which is data stored in HDFS. The Rmr2 package uses map and reduce functions to apply the MapReduce task. The functions have pairs of keys and values. The package uses the function keyval to return key value pairs, which contain vectors, lists, matrices, data frames, or NULL value. Users can also call the keyval(NULL,x) function to get the return value of x. Users can also use from.dfs to load large-scale data into memory. It also returns a <key,value> pair as a result.

The function from.dfs is useful to define MapReduce algorithms. It will fit in memory and pass to the next set of functions. This example, run on Big Data, is equivalent to the lapply function and performs the map function. The following example performs the reduce function similarly to function tapply in R.

```
  groups = rbinom(32, n = 50, prob = 0.4)
# print out groups
result =  tapply(groups, groups, length)
# print out result
```

Figure 4-9. Sample MapReduce script [48]

This example, which runs in MapReduce format, counts the number of outcomes that occurred from the binomial:

```
groups = to.dfs(groups)
from.dfs(
  MapReduce (
    input = groups,
    map = function(., v) keyval(v, 1),
    reduce =
      function(k, vv)
        keyval(k, length(vv))))
```

Figure 4-10. Sample MapReduce script [48]

The first to.dfs statement will move the data into HDFS. The user should specify the HDFS path of the input data for the MapReduce procedure. In this example, we used the variable groups which contain a Big Data object as the input parameter. The map function is implicitly included as function(k,v) keyval(k,v).

The reduce function needs two arguments, first as a key followed by the collection of all values associated with the key. The return value of the map function can be a vector, list, data frame, and matrix. When the user returns the values of a specific class, the function will preserve the values through the shuffle phase. The reduce function can return a NULL value or a key-value pair, which are generated by the function keyval or object defined by keyval(NULL,x). The default process of MapReduce is not the reducer function. The keys will be the realization of the binomial. The values are all 1.

Rmr function	Description
"Map" step	FUN will apply to the data on the worker node. The intermediate result will be stored at the storage. Duplicate data will be eliminated by the master node
"Shuffle" step	The data will be dispatched by the Keys to different worker nodes. Data on each worker node share a unique Key.
"Reduce" step	The data with unique key will be processed in parallel by worker nodes

The sample geospatial use case and code are described in chapter 5.

4.6 Estimating the System Resource

We choose to build spatial index in memory to ensure the high performance of spatial query. However, the memory consumption of spatial index is high, which may deplete the system resources resulting in system halt or crash.

Since building spatial index is a memory consuming task, we used an empirical formula to calculate the memory for spatial query.

> *"The recommended size for this temporary tablespace 【which contains the R-tree structure】 is 100*n bytes, where n is the number of rows in the table‟*
>
> *--- https://docs.oracle.com/html/A88805_01/sdo_inde.htm*

If the spatial data contains 10^8 POIs, the R-tree size in memory will be 100*10^8=10G per the formula.

In this framework, the format of geospatial data can be defined as decimal or float. Here is the sample geospatial data format in Decimal (7, 4):

```
  Longitude -129.9417    Latitude 50.0333
```

The length of the Earth’s equator is approximately 6378km in WGS84 system, which provides about 100 km latitudinal resolution near the Equator [49]. That means POIs, which using the Decimal (7, 4) format of geospatial data, can provide a level of 10 meters resolution.

The grid file of 10^8 POIs may cover an area of 10,000 km^2. The area of the New York Metro is 8683 km^2. R-tree size as big as 1TB memory can result from indexing a grid file up to 1,000,000 km^2. The area of Texas state is 695,621 km^2.

Another format of geospatial data can be IEEE754 floating point.

Common name	Base	Digits	Decimal digits	Exponent bits
Single precision	2	24	7.22	8
Double precision	2	53	15.95	11

The equivalent decimal digits of IEEE754 single precision float point is seven. The equivalent decimal digits of IEEE754 double precision float point is fifteen. That means the single precision format of geospatial data can provide half meter level resolution.

The double precision format of geospatial data can provide nanometer 10^ $^{(-9)}$ level resolution. The complexity of geospatial computation depends on the resolution of the geospatial data.

CHAPTER 5 APPLICATIONS OF THE FRAMEWORK

The framework provides effective methods to analyze the complex information from the user data and eases the development of geospatial applications in various fields. As the example of Hadoop program suggests, we will describe the methods of the framework to process and analyze gridded geospatial data in this chapter. We will provide examples of alternative applications for the framework: spatial data mining within geospatial data with information marks.

5.1 Spatial data warehouse

Our first test data come from the need of ocean investigation task. During ocean investigation cases, there is a definite need of bathymetry data. This kind of data is typical spatial statistical analysis type of point patterns. Ocean investigation applications often require multiple different sources of bathymetry data. R is not capable of handling spatial analyzing tasks using very large-scale bathymetry data because of its design limitations. Suppose we have many geospatial data files. Each file may cover a different area in the grid system, which must be properly shared and combined when the data is used. All the data is stored in HDFS and users can use RHadoop to access the data. All the data in these analysis functions are pre-processed at Hbase. Then, we use a spatial statistics package in R to analyze these geospatial files. The great advantage to using Hbase in this framework is that it provides features like ETL (Extract, Transform and Load) as the spatial data warehouse. The first feature is bulk load, such as Hadoop streaming, which allows the programmer to write utilities running as mapper/reducer on Hbase.

Here is the example code for bulk load geospatial data to HBase.

```
#bulk load geospatial data to HBase
hadoop  jar  /home/zyang/hbase/hbase-0.94.2/hbase-0.94.2.jar  importtsv
-Dimporttsv.columns=HBASE_ROW_KEY,poi_cf:longitude,poi_cf:latitude,poi_
cf:mean,poi_cf:standarddeviation geos_test DBDBV_test_tsv
##-Dimporttsv.separator=","    use csv format
##-Dimporttsv.bulk.output=/output   generate hfile to output
##hadoop jar ~/hbase-0.92.1/hbase-0.94.2.jar  completebulkload /output
users       load hfile to table users in hbase
#full table scan on the geospatial data
scan 'geos_test'
```

Our test environment is a DELL Precision T3400 PC with Intel single Core2 2.83GHZ CPU and 2GB RAM. The operating system is Ubuntu 12.04.4. In our experiment, we tried to load 2GB geospatial data into Hbase and MySQL. In MySQL, we used the LOAD DATA INFILE method. In HBase, we used BulkLoad feature to load the 2GB data.

Our test results show that the BulkLoad feature of Hbase is significantly faster than the Load feature of MySQL when dealing with large-scale data.

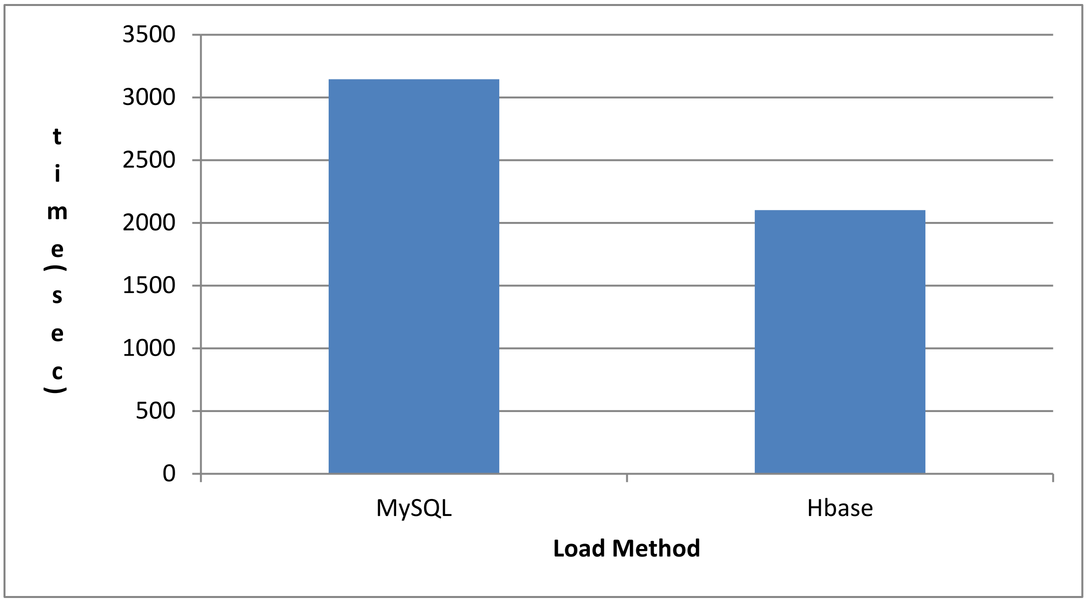


Figure 5-1. Runtime for load task

The experiment shows the advantage of using HBase to store large-scale geospatial data. The BulkLoad feature will improve the performance when a user loads mass data into the HBase.

The second extension is JSI (Java Spatial Index) [44], which allows a user to create an R-tree [51] spatial index on geospatial data. Using the R-tree index, users can easily perform customized geospatial queries such as Boundary query, Range Query, and Knn (nearest neighbor) Query. In our framework, this feature will help improve the performance to query with large-scale geospatial data.

The third extension is Geophile [45], which provides a Java implementation of spatial join. Users can write code to specify two spatial objects like R and S. The result will be the overlap of objects R and S.

Here is part of the example of Range Query based on R-Tree index on geospatial data. The whole example can be found in the Appendix.

```
#import JSI(R-Tree index library)
import com.infomatiq.jsi.Rectangle;
import com.infomatiq.jsi.rtree.RTree;

  //boundary query for geo-spatial data
       String cf ="poi_cf";
       String column1 = "longitude";
       String column2 = "latitude";
       String column3 = "mean";
       String column4 = "standarddeviation";
       //set POI boundary
       String long_lower = "-129.9900";
       String lat_lower = "49.0400";
       String long_upper = "-129.9250";
       String lat_upper = "49.0500";
       //set central POI

       String long_central = "-129.9450";
       String lat_central = "49.0450";
       String distance = "0.01";
```

Here is the performance test result of the spatial index on Hbase. There are 9,000,000 POIs in the test data.

In this figure, the range means the Euclidean distance we chose to calculate the distance between two POIs.

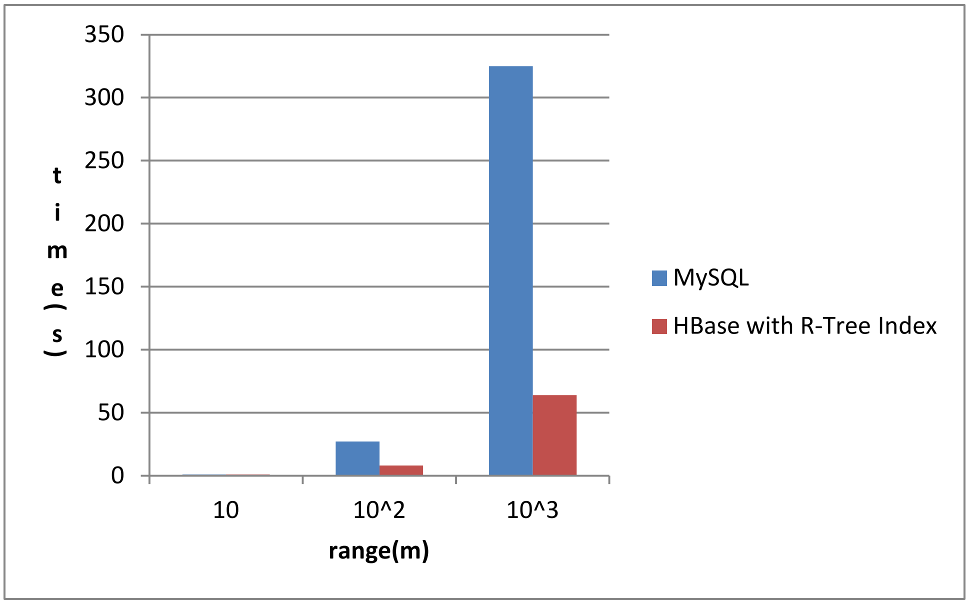


Figure 5-2. Runtime for Range query

In our test results, the BulkLoad feature of Hbase is significantly faster than the Load feature of MySQL when dealing with large-scale data.

In this figure, the boundary means the length of the boundary of the rectangle we chose to select the POIs inside the boundary.

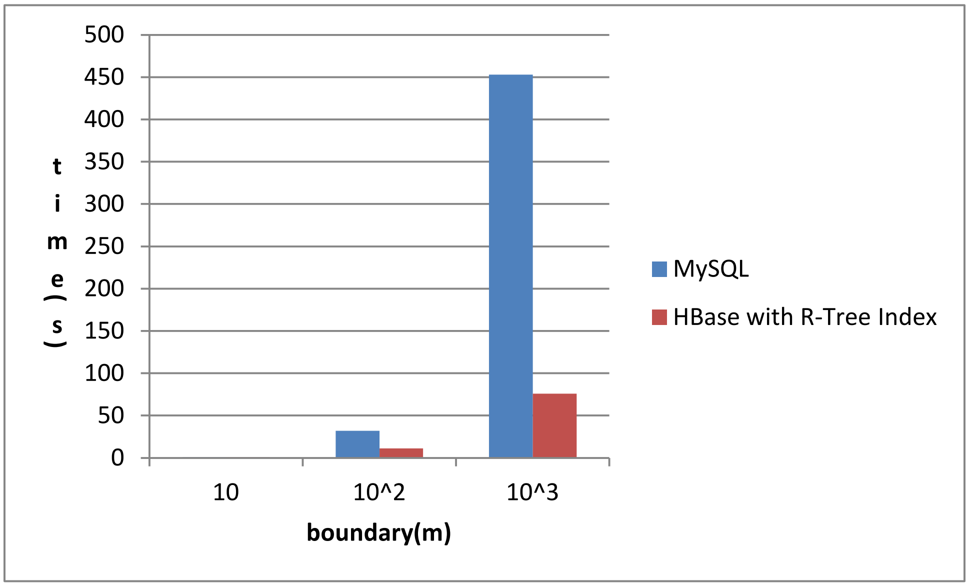


Figure 5-3. Runtime for Boundary query

In this figure, the K means the value of the nearest neighbors we chose to select the POIs.

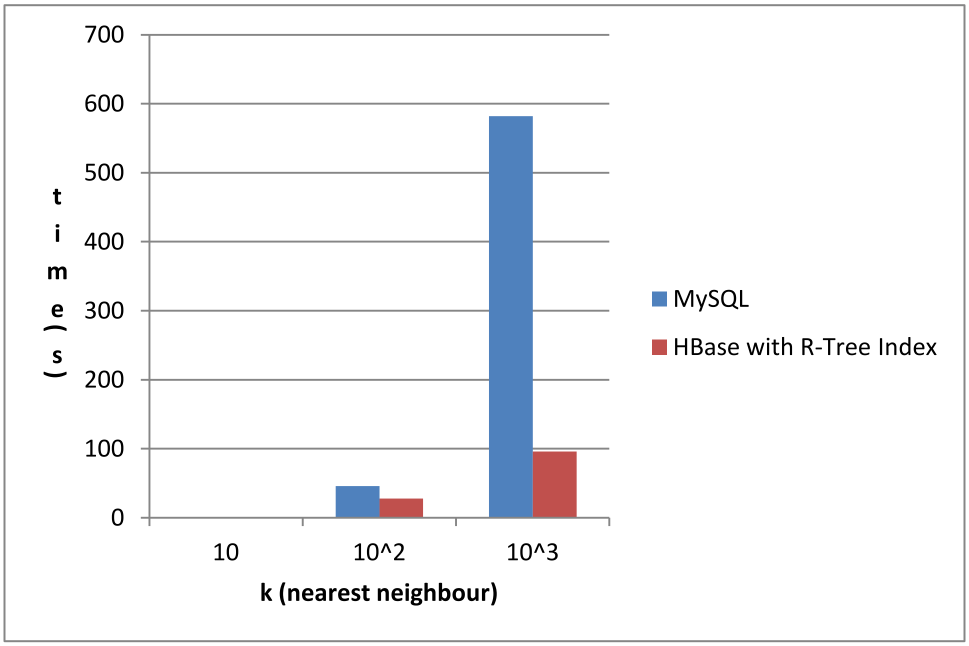


Figure 5-4. Runtime for Knn query

In this figure, the R and S are two spatial objects which contain a specific number of POIs.

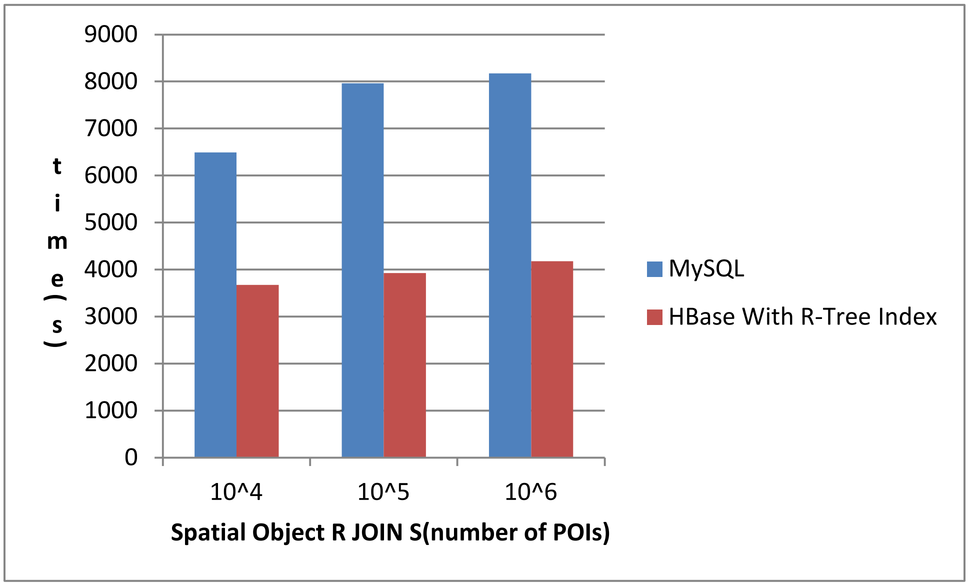


Figure 5-5. Runtime for spatial join query

The following test proceeds with large-scale data on HPC. Our test environment was a 16-Cores AMD Opteron(tm) Processor 4386 3099.94 GHZ CPU and 32GB RAM. The operating system was Ubuntu 14.04. During our experiment, we tried to run spatial query on large-scale geospatial data on Hbase and MySQL. In MySQL, we used SQL to retrieve data. In HBase we used the In-Memory R-Tree feature to test spatial query.

Here is the performance test result of spatial index on Hbase. There are 10^6, 10^7, 10^8 and 10^9 POIs in the test data. In this table, the range means the Euclidean distance we chose to calculate the distance between two POIs.

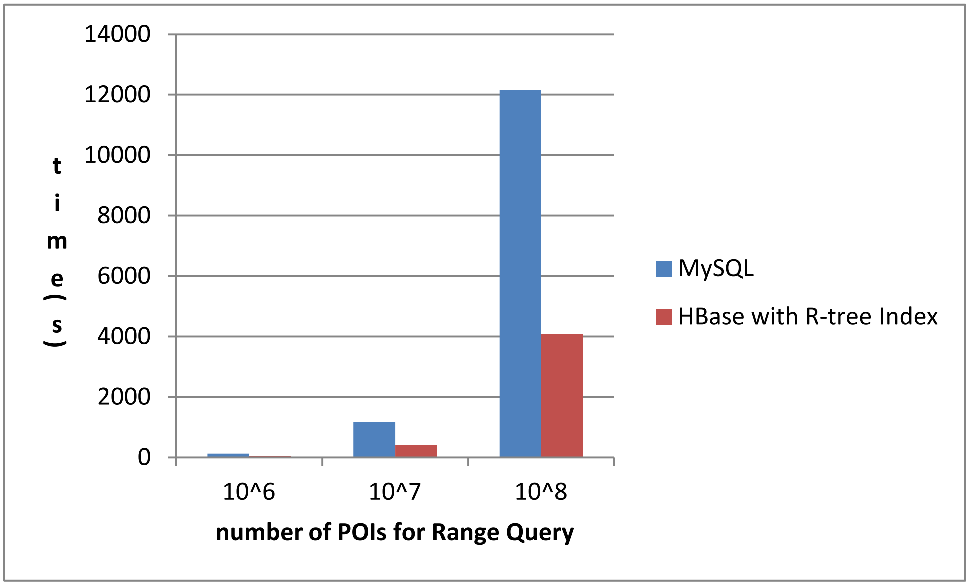


Figure 5-6. Runtime for Large-scale geospatial data Range query

The performance test results of spatial index on Hbase are shown next. There are 10^6, 10^7, 10^8 and 10^9 POIs in the test data. In this table, the boundary means the length of the boundary of the rectangle we chose to select the POIs inside the boundary.

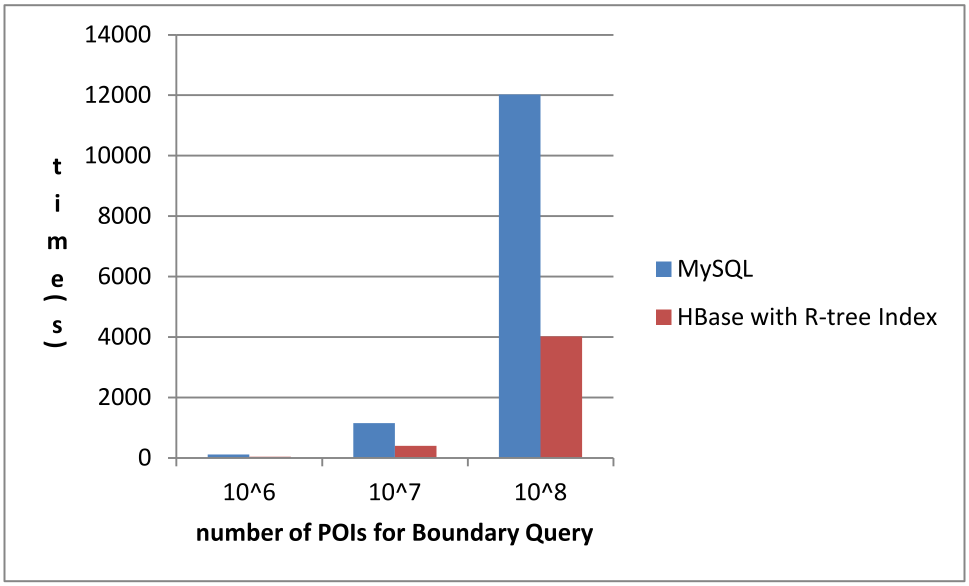


Figure 5-7. Runtime for Large-scale geospatial data Boundary query

Here is the performance test result of spatial index on Hbase. There are 10^6, 10^7, 10^8 and 10^9 POIs in the test data. In this table, the K means the value of the nearest neighbors we chose to select the POIs.

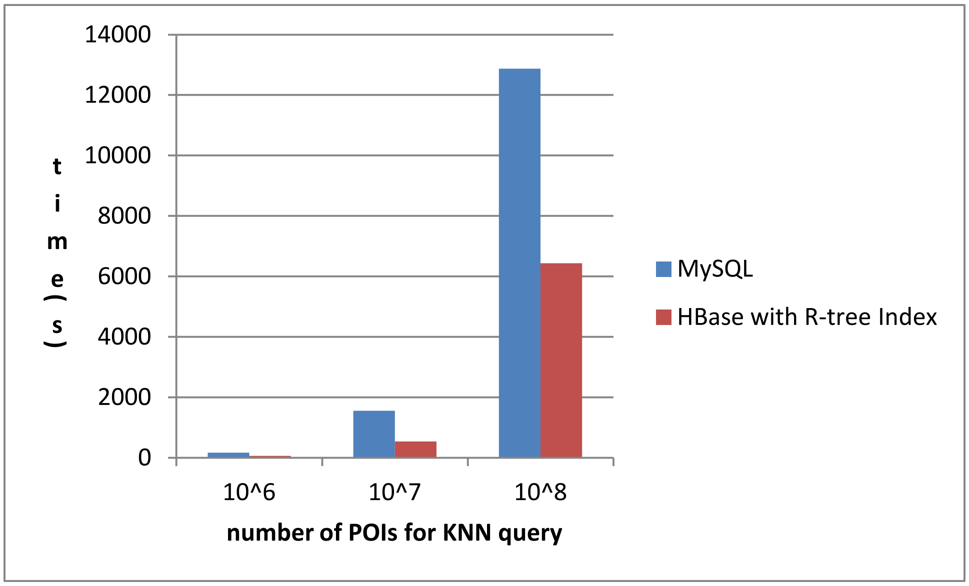


Figure 5-8. Runtime for Large-scale geospatial data Knn query

Next, we show the test results of spatial index on Hbase. There are 10^6, 10^7, 10^8 and 10^9 POIs in the test data. In this table, the R and S are two spatial objects which contain a specific number of POIs.

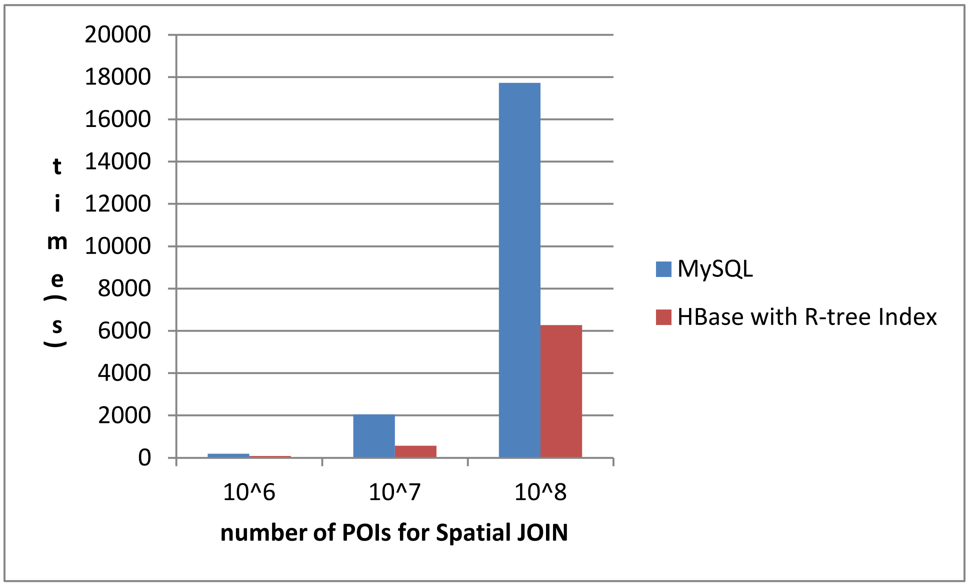


Figure 5-9. Runtime for Large-scale geospatial data Spatial JOIN

Next, we show the test results of Hbase with In-memory system. There are 1G, 10G, 100G size in the test data. In this table, the range means the Euclidean distance we chose to calculate the distance between two POIs.

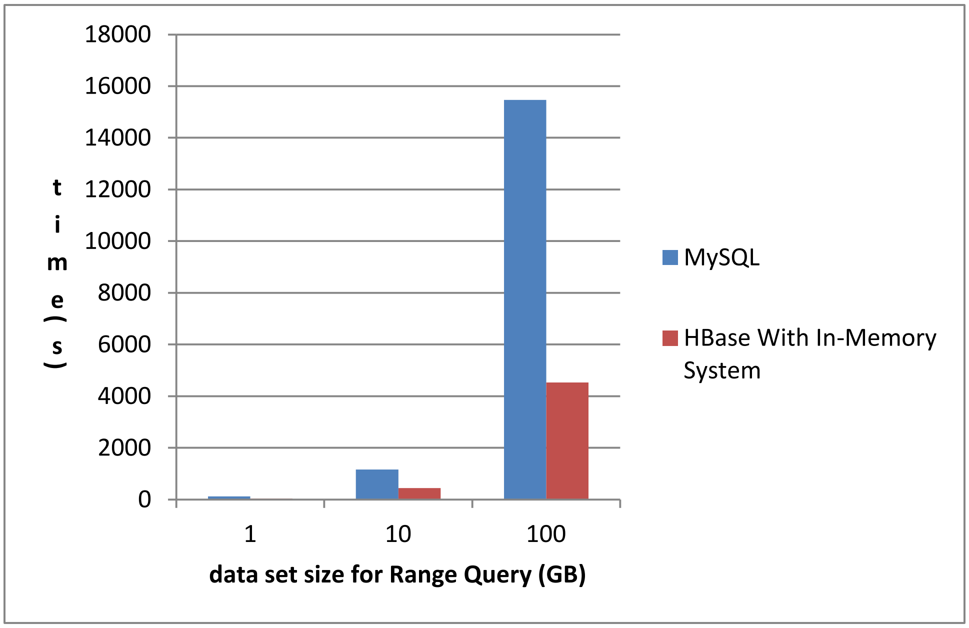


Figure 5-10. Runtime for Large data set Range Query

There are 1G, 10G, 100G size in the test data. In this table, the boundary means the length of the boundary of the rectangle we chose to select the POIs inside the boundary.

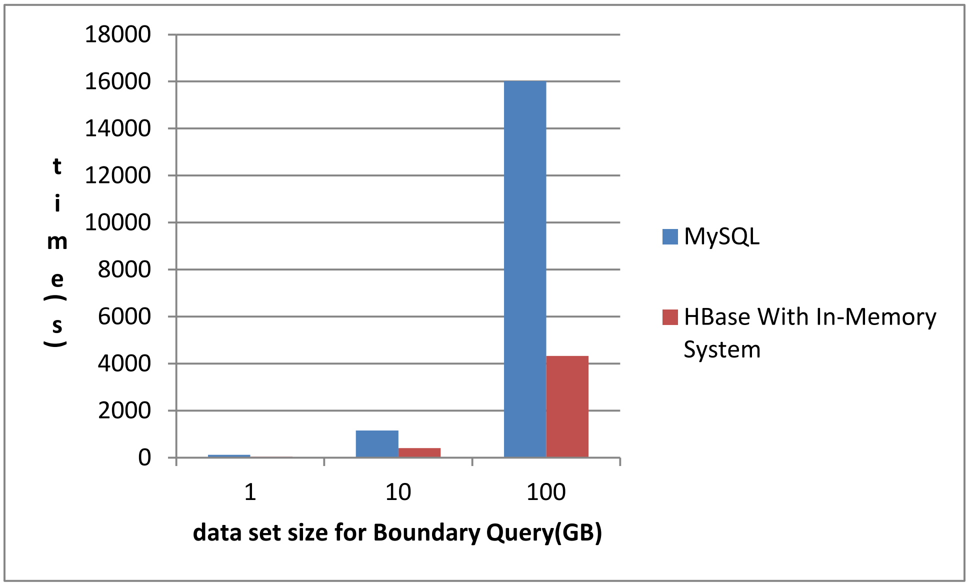


Figure 5-11. Runtime for Large data set Boundary Query

There are 1G, 10G, 100G size in the test data. In this table, the K means the value of the nearest neighbors we chose to select the POIs.

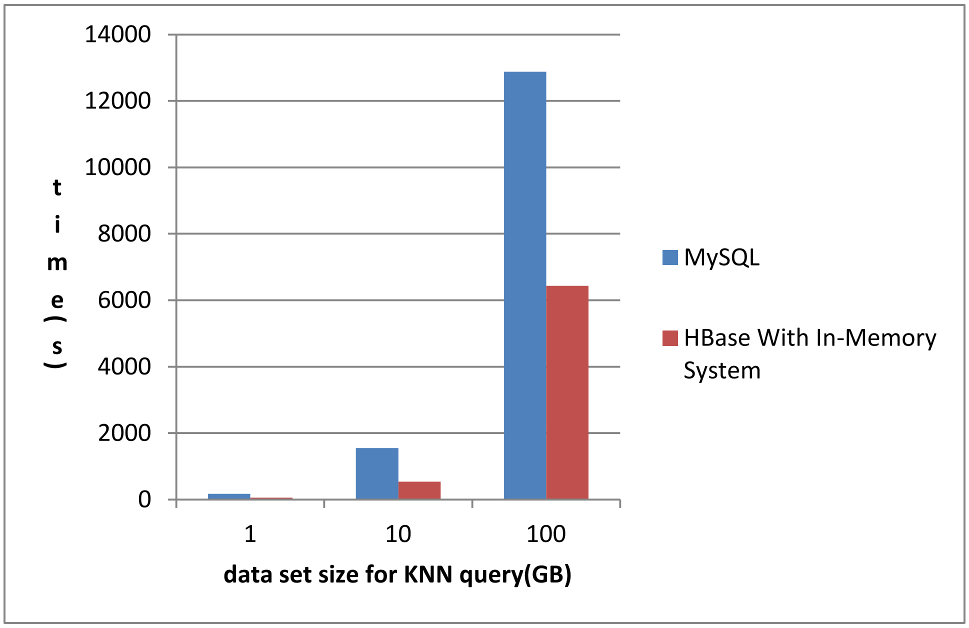


Figure 5-12. Runtime for Large data set KNN Query

There are 1G, 10G, 100G size in the test data. In this table, the R and S are two spatial objects which contain a specific number of POIs.

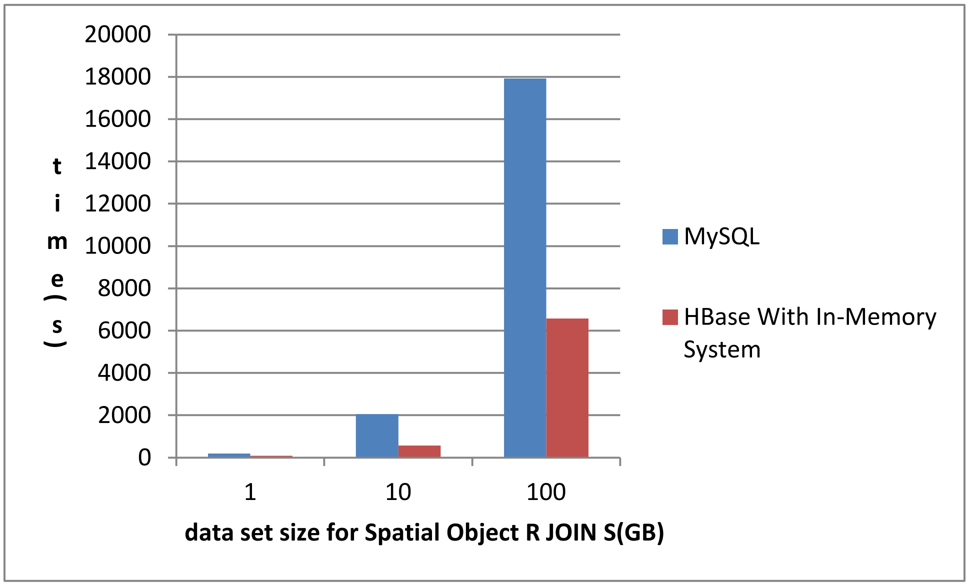


Figure 5-13. Runtime for Large data set Spatial JOIN

The spatial query and spatial join based on R-Tree index are significantly faster than the traditional method. The reason for this is that our spatial index and spatial query/join calculation all runs in memory. Another factor is that HBase is column stored NoSQL DBMS, which makes I/O cost of HBase as fast as the File system. Meanwhile, traditional DBMS like MySQL take more time to maintain data integrity such as ACID rules.

This means our framework provides an efficient way to do the ETL (Extract, Transform, and Load) work for geospatial data. In real cases, users sometimes get raw data over a very large area. With this spatial index on Hbase, the user can easily and efficiently run spatial query to get the specific set of geospatial data over large-scale raw data.

5.2 Spatial data mining case study

After ETL, users can run an analysis function over the clean data.

In addition to the spatial query (data selection process), users could run an analysis functions over the clean data. Below is the code to draw a bathymetry data plot.

```
To have some more flexibility with coloring I use the following, e.g.
with rainbow colors:

col<- rainbow(255,end=5/6)

colid <- function( x, range=NULL, depth=255 )
{
    if ( is.null( range ) )
        y <- as.integer(x-min(x))/(max(x)-min(x))*depth+1
    else
    {
        y <- as.integer(x-range[1])/(range[2]-range[1])*depth+1
        y[ which( y < range[1] ) ] <- 1
        y[ which( y > range[2] ) ] <- depth
    }
```

```
   y
}

plot(  geo_data$`poi_cf:longitude`,geo_data$`poi_cf:latitude`,col=col[  c
olid(geo_data$`poi_cf:mean`) ] )

-----------------------------------------------------------------------
<script>
     // This example uses a GroundOverlay to place an image on the map

     var dataOverlay;

     function initMap() {
       var map = new google.maps.Map(document.getElementById('map'), {
         zoom: 7,
         center: {lat: 49.65, lng: -126.65},
         mapTypeId: google.maps.MapTypeId.HYBRID
       });

       var imageBounds = {
         north: 51.,
         south: 49,
         east: -127.,
         west: -130.
       };

       var overlayOpts = {
         opacity:0.5
       }

       dataOverlay = new google.maps.GroundOverlay(
           'path/to/file',
           imageBounds,overlayOpts);
       dataOverlay.setMap(map);

     }
   </script>
```

The following figures show the Plot using point pattern of the bathymetry data in R.

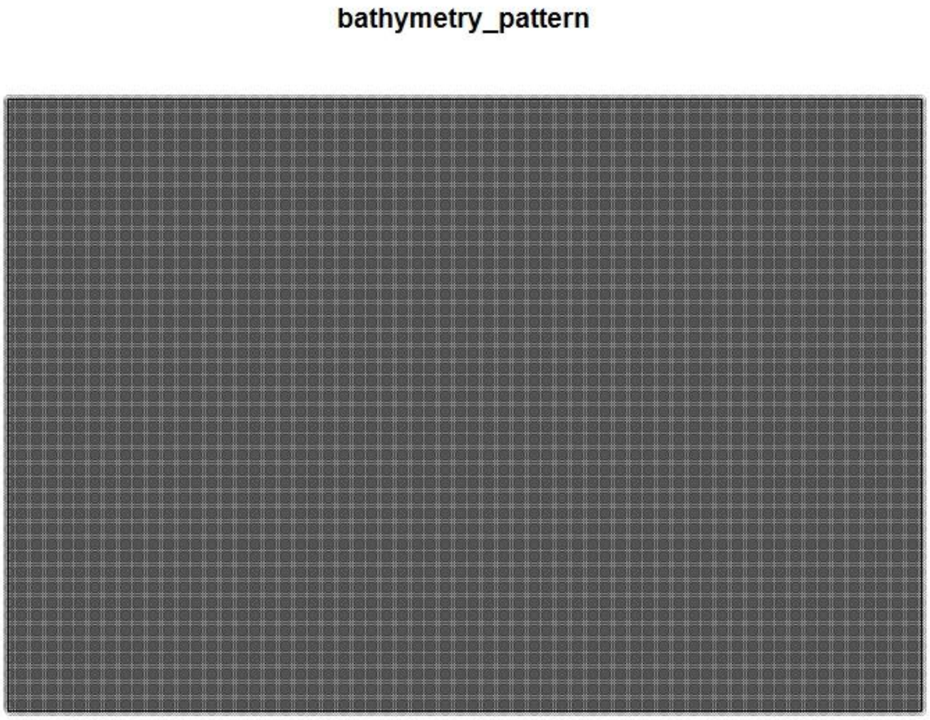


Figure 5-14. R Plot using bathymetry data point pattern

The bathymetry data was downloaded from website [50]. The point pattern shows the tracking information of the ocean investigation vessel.

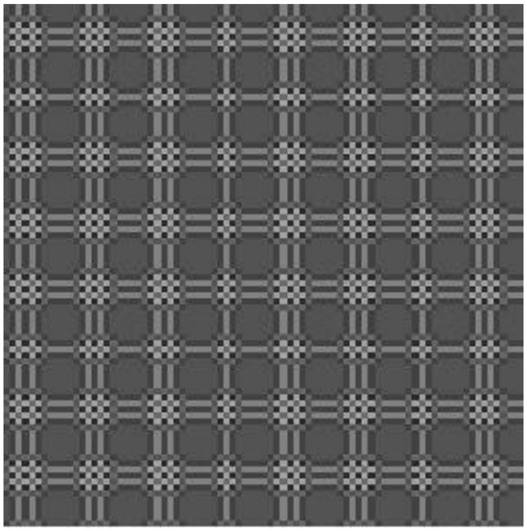


Figure 5-15. Detail of the bathymetry data point pattern

The above figure shows the detail of the tracking point pattern.

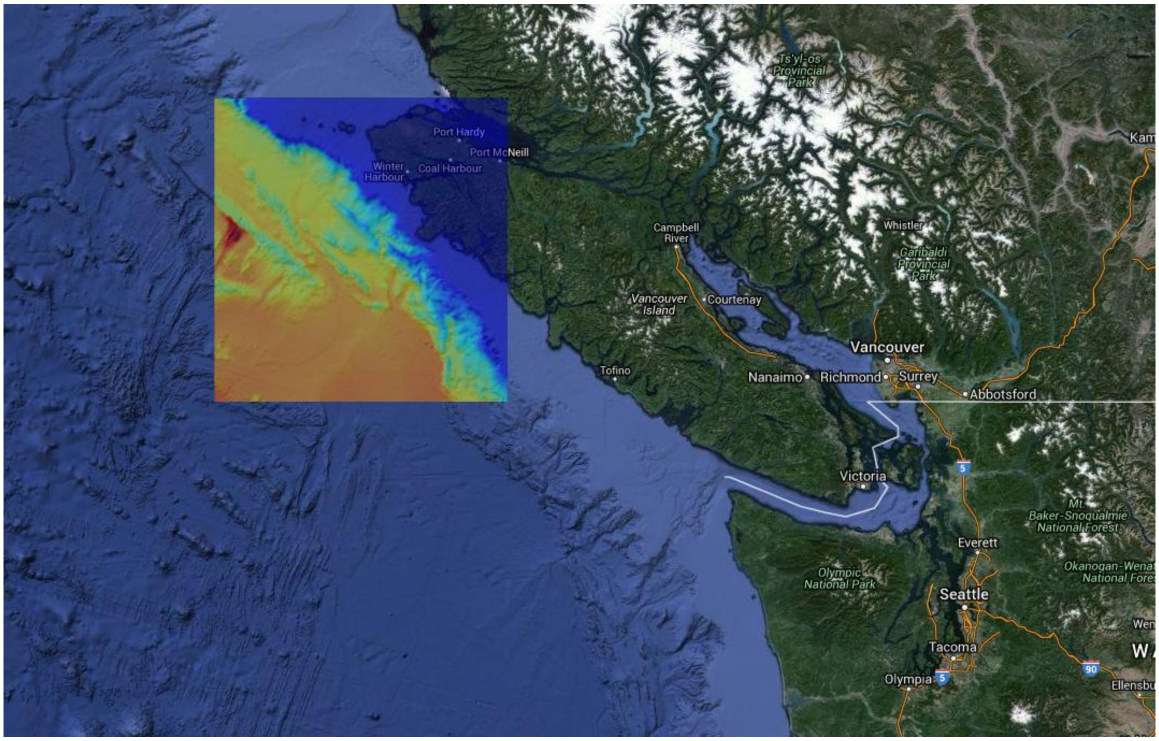


Figure 5-16. R Plot using bathymetry data above Google Maps Layer

The bathymetry data was downloaded from website [50]. The bathymetry statistical layer has been visualized above the satellite image layer. This provides an effective visualized report to the end-user. This function shows the basic analytical work on the spatial data.

The following code sample shows how to plot the bathymetry data in 3-D.

```
##select 5000 points as sample
> geo_data_sample <- geo_data[sample(nrow(geo_data), 5000), ]
#display points in color
> plot_ly(geo_data_sample, x = longitude, y = latitude, z = depth, type =
"scatter3d", mode = "markers", color=depth)
```

The following figure shows the same bathymetry data plotted by 3-D analysis function.

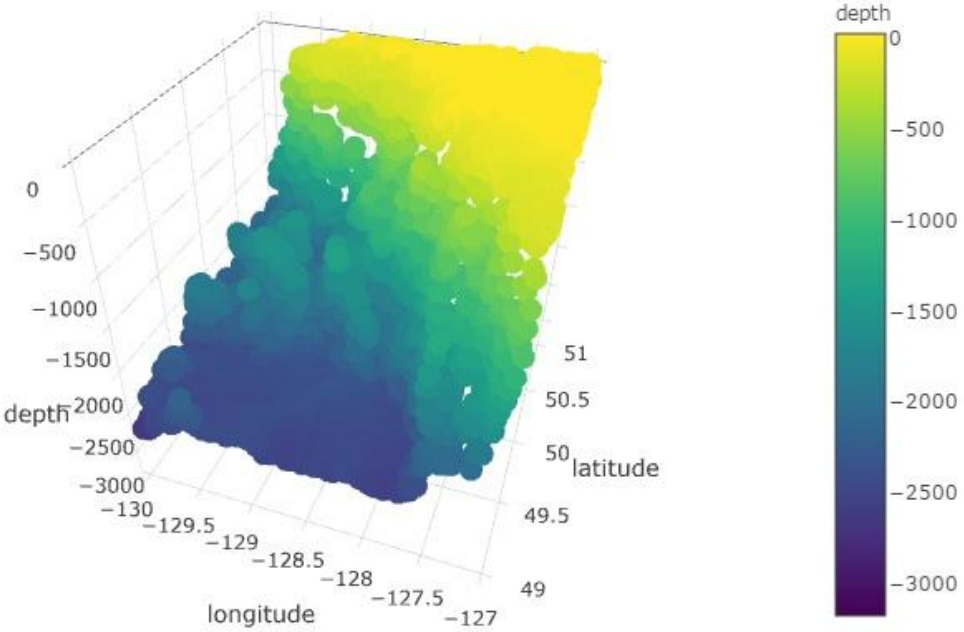


Figure 5-17. 3-D Plot using bathymetry data

The Morisita Index is a common method for analyzing spatial patterns. To get the Morisita index, the function will firstly calculate the quadrat counts of the spatial point pattern. Then the generated index of spatial aggregation for the previous pattern will be the Morisita index. The algorithm is to divide the spatial domain to Q quadrat with equal size and shape. Then the algorithm will count the number of points falling in every quadrat. Lastly, the n[i] number of points in the i-th quadrat will be counted as the Morisita Index. The sum number of points is represented as N value.

The point pattern is the analysis class as object. We can also plot the result of this analysis. The Morisita overlap index can calculate the overlap between samples. The founder, Massaki Morisita, believes the diversity will be modified when the sample size is modified. The formula used in the analysis package is

```
MI = Q * sum(n[i] (n[i]-1))/(N(N-1))
n[i] - points number in the i-th quadrats
n - total number of points [52]
```

The sample data we used contained 87000 bathymetry points. The analytical package uses POIs between the coordinates [-130.0000, -126.0000] x [49.0000, 51.0000] window to choose the sample of POIs. Here is the code to draw a Morisita index plot:

```
> miplot Morisita Index plot
> miplot(X)
```

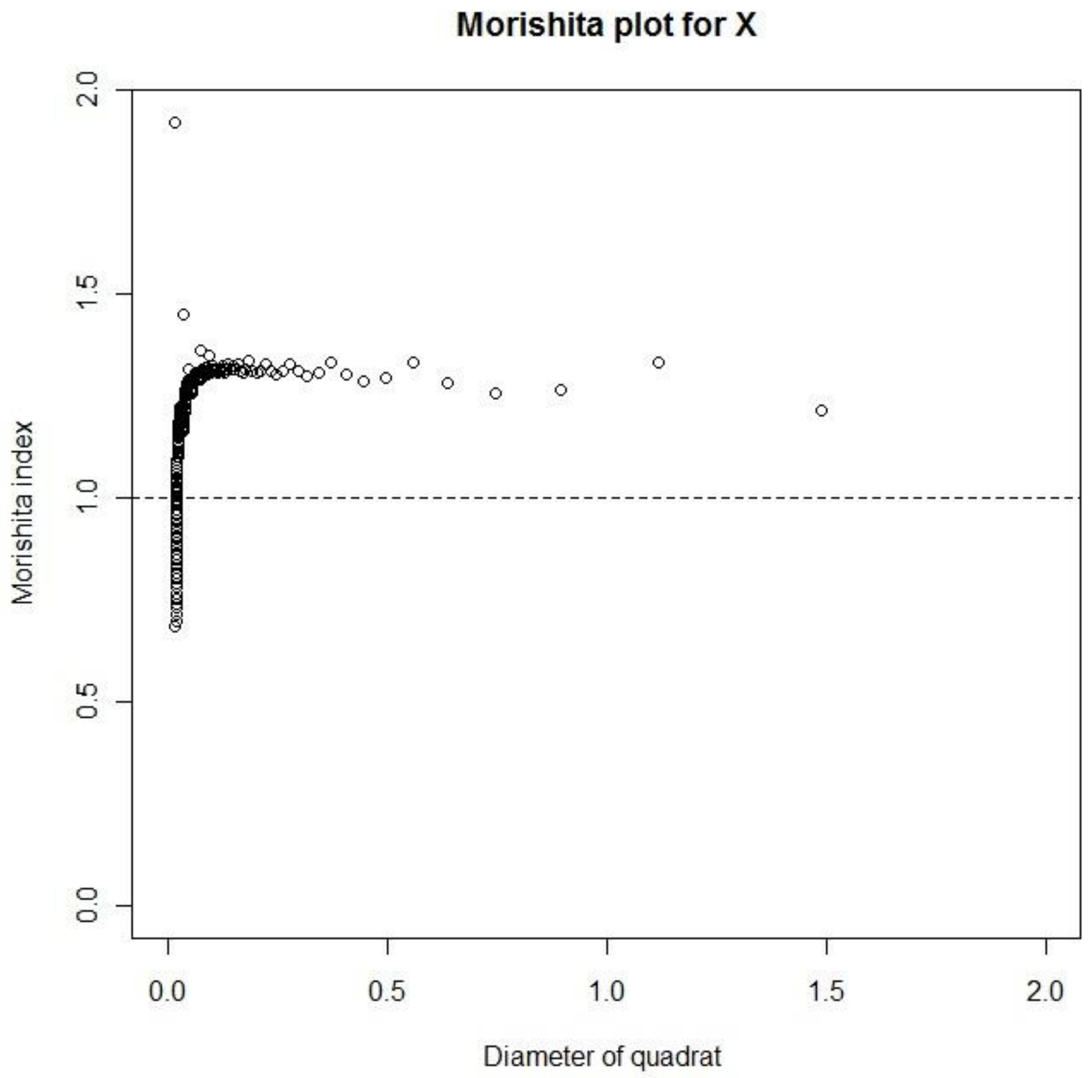


Figure 5-18: Morisita Index analysis on spatial data

The Morisita index is superior for comparing the similarity between different samples. The advantage of the Morisita index is that it does not vary with population density. If the data is mostly in uniformly distributions, the Morisita index value falls between 0 and 1. However, if the data is in clumped distribution then the Morisita value falls between 1 and n [52].

In this case, we used our framework to get the distribution and spatial pattern of the large-scale geospatial data. For the most part, the Morisita index values did not fall into 0 to 1, which meant our raw data was not in uniform distribution. Users should check the raw data to determine if it is bad sampling or not. This example shows the function that can help users find the distribution of their raw geospatial data.

The second analysis example is the clustering model. The analysis function Kppm can perform the task in homogeneous or inhomogeneous mode. A user can use the Kppm function to predict and plot the result model. The Kppm function will analyze the intensity of the trend argument. If users set trend argument as 1, it will be a homogeneous model. The function will calculate the intensity of the points inside every area of the window. If the trend argument is not 1, the function will use the inhomogeneous model. The fitting process is a Poisson process with intensity.

Here is the sample code. We specify the model as Thomas process which is a Poisson cluster process.

```
> kppm(X)
number of data points exceeds 3000 - computing border estimate only
Stationary cluster point process model
Fitted to point pattern dataset 'X'
Fitted using the K-function
Cluster model: Thomas process
Fitted parameters:
      kappa       sigma          mu
9.881081e-01 5.192104e-01 1.100588e+04

> kppm_model<-kppm(X)
number of data points exceeds 3000 - computing border estimate only
> kppm_model
```

```
Stationary cluster point process model
Fitted to point pattern dataset 'X'
Fitted using the K-function
Cluster model: Thomas process
Fitted parameters:
      kappa       sigma          mu
9.881081e-01 5.192104e-01 1.100588e+04
> plot.kppm(kppm_model)
```

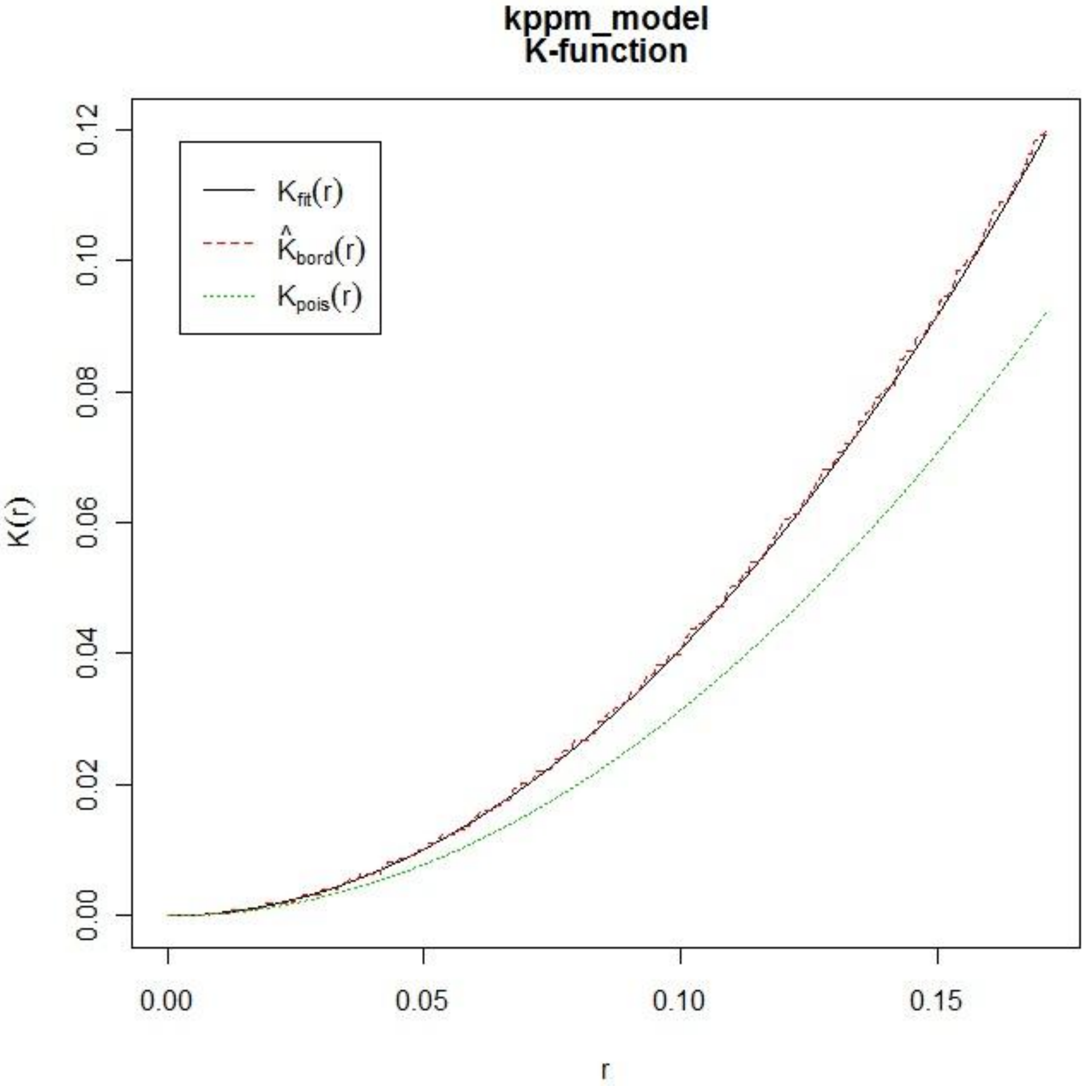


Figure 5-19. Kppm function analysis result

The kppm function fits a Matern clustering model for the cluster point patterns based on our geospatial data. The plot shows that the estimated model fits the real data well. That means the raw data is a Poisson distribution.

These two functions show the basic analytical work on the spatial data. As such, we performed data analytical work on our framework. The framework is a whole process to run the spatial data mining work. The first step is to use HBase as the ETL (Extract, Transform, and Load) tool. Then we build a spatial index which may be R-Tree on the cleaned data. Users can write some Java code to retrieve the data by various methods such as Boundary query, Range query, Knn (Nearest Neighbor) query, or any other user-defined geospatial query. After these steps, the raw data has been cleaned and selected. Users can run the spatial data mining function to find the information behind the geospatial Big Data. Since we chose HBase as the high performance NoSQL database and built-in spatial index, the framework has great flexibility. It is easy to write API to call the spatial index to do the ETL work.

5.3 Random forest method to process large file

When trying to process large files, the users may get the following message:

```
> data3 <- ncvar_get(ncid,"water_temperature")
Error: cannot allocate vector of size 1.3 Gb
```

The data objects in R are stored in virtual memory. The user will get error messages if the system cannot obtain the appropriate memory space when trying to allocate a large size vector. The limit is not only a question of having the space for the process but also the limits of the system itself. R also defines the limit of the number of bytes in characters and the limit on each dimension of an array.

However, in geospatial applications there are, usually, very large data files to process. One solution for analyzing a large-scale data set is to divide it into several small data frames. But the problem with this approach is that the user will get different fitted models because the data set fragments are analyzed in parallel. Since R is unable to fit overly large geospatial data sets into a single model, one common method for handling this bottleneck is sampling with replacement. The training set of the Random Forests [53] algorithm is drawn by sampling with a replacement technique.

The Random Forests method [53] is used to build multiple decision trees to train the model. The task will get the class models as classification result or predicted mean value as the regression result for each tree. So, the Random Forests method avoids the “overfitting” problem inherent in the traditional decision tree method.

We chose Poisson re-sampling as our sampling method. The reason we chose Poisson sampling is that the sum of multiple Poisson variables is still Poisson. The data sample is extracted from the input data set based on a Poisson distribution. The sampling task runs several times. The result of multi-run Poisson distribution is also a Poisson distribution [54]. The sampling task can be run in parallel by the Map Reduce task, since each sampling task runs in isolation. The MapReduce process is as follows: the mapper runs the sampling task on the input data set; the reducer shuffles the samples, and then performs the model fitting task.

The following is the algorithm [55] to perform the sampling task through MapReduce.

```
a <- MapReduce (input="~/user/zyang/DM_test_tsv ",
            input.format="text",
            map=poisson.subsample,
            reduce=fit.trees,
            output=NULL)

b <- from.dfs(a)

training.data <- read.table("~/training.csv",
                       header=FALSE,
                       sep="\t",
                       quote=FALSE,
                       row.names=NULL,
                       col.names=column.names,
                       fill=TRUE,
                       na.strings=c("NA"),
                       colClasses=c(longitude="numeric",
                                  latitude="numeric",
                                  mean="numeric",
                                  stdv="numeric"))
a <- randomForest(formula=model.formula, data=training.data,
na.action=na.roughfix, ntree=10)
```

With the above approach, the user now has the ability to handle large-scale geospatial files in R.

The performance test is based on large-scale geospatial data. We chose test set sizes from 10^5 to 10^8 POIs. Current applications, for instance ocean investigation, continuous climatology data, or vessel GPS tracking always generate large-scale geospatial data. A normal GPS tracker on a truck will generate almost 10^5 POIs one day if it records geospatial data every second. However, the UPS Company has 60000 trucks in operation every day. The total raw data of whole fleet will exceed 10^ 9 POIs.

The following is the sample of the test data.

```
 poi_cf.longitude poi_cf.latitude poi_cf:mean poi_cf:stdv
-130.0000   49.0000  2698   26.98
-129.9917   49.0000  2678   26.78
-129.9833   49.0000  2657   26.57
-129.9750   49.0000  2637   26.37
-129.9667   49.0000  2639   26.39
```

The test data is POIs with salinity and temperature marks. Every POI contains the randomly generated salinity and temperature marks.

number of POI	runtime in R (sec)	runtime In R-MapReduce (sec)
10^5	1	1
10^6	2	8
10^7	4	8
5*10^7	18	362
10^8	(Cannot allocate memory due to Limit of R software；Divide to several files run sequentially) 1732	924
10^9	(Cannot allocate memory due to Limit of R software; Divide to several files run sequentially) 2869	1328

Table 5-1: runtime for R analytical task

When the data amount is not large enough, the runtime in R is significantly faster than the task in R-MapReduce. The reason for this is that the R function running in memory will be faster than multiple MapReduce tasks. MapReduce has a high communication cost and I/O cost. However, when the size of the data set passes some threshold amount, R cannot allocate memory for such a huge data analytical task. In the meantime, R-MapReduce shows its great scalability for processing large-scale data.

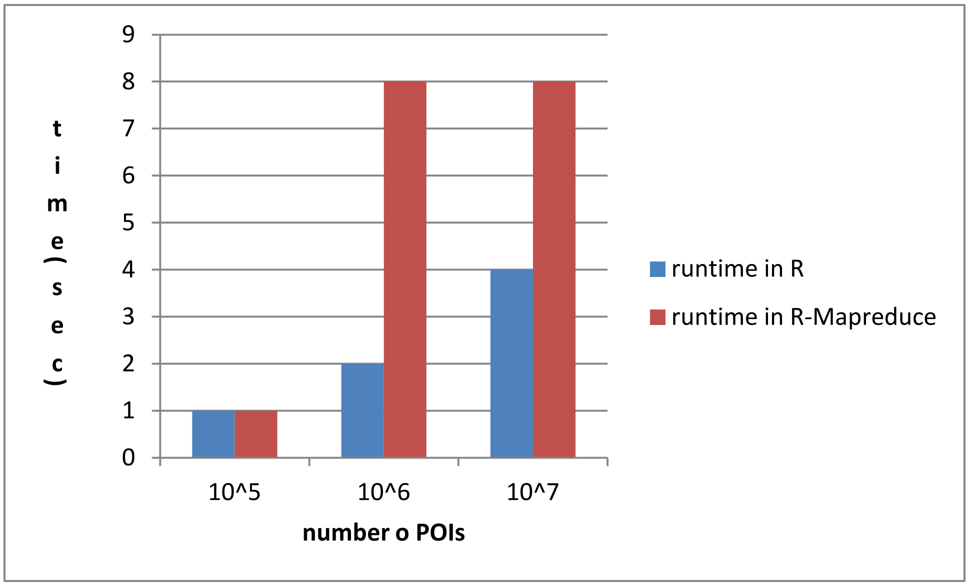


Figure 5-20: performance test in small amount of data

As figure 5-16 shows, the traditional R function yielded small runtime compared with the R-MapReduce task when processing a small amount of data.

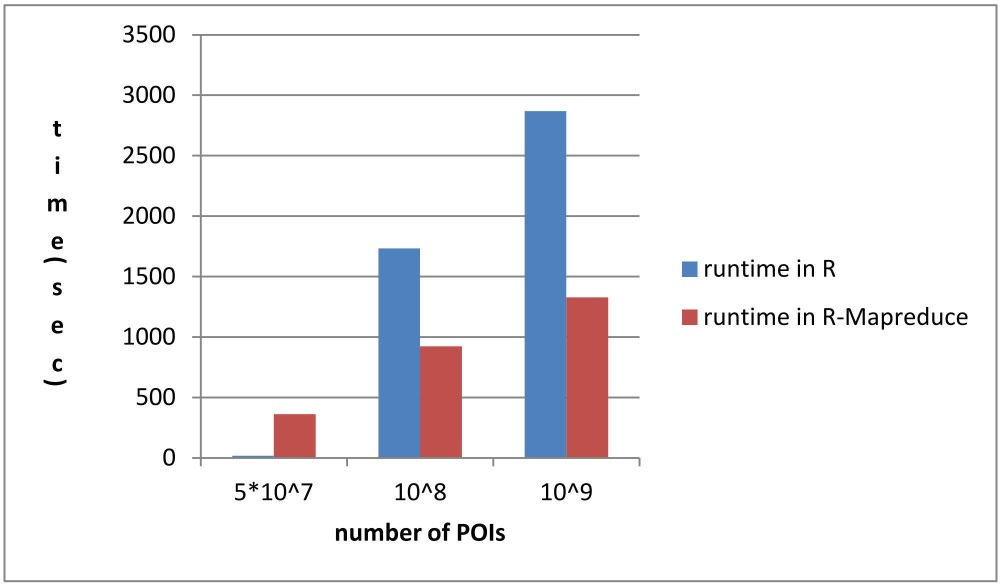


Figure 5-21: performance test in large-scale data

As figure 5-5 shows, the traditional R function cannot handle such large-scale data because R cannot allocate memory beyond its limit. However, the R-MapReduce task shows its advantage for handling large-scale data.

5.4 Shark alert map

The goal of this case study is to predict the probability of the appearance of a shark. The more the shark appears, the greater the number of shark attack news reports and vice versa. Below is a map that shows the details regarding the number of confirmed shark attacks near California's coastline during year 1926-2014. This map is from the Florida Museum of Natural History. The value is based on real world cases ranging from a high incidence of attacks, in red, to a low incidence of attacks, in green.

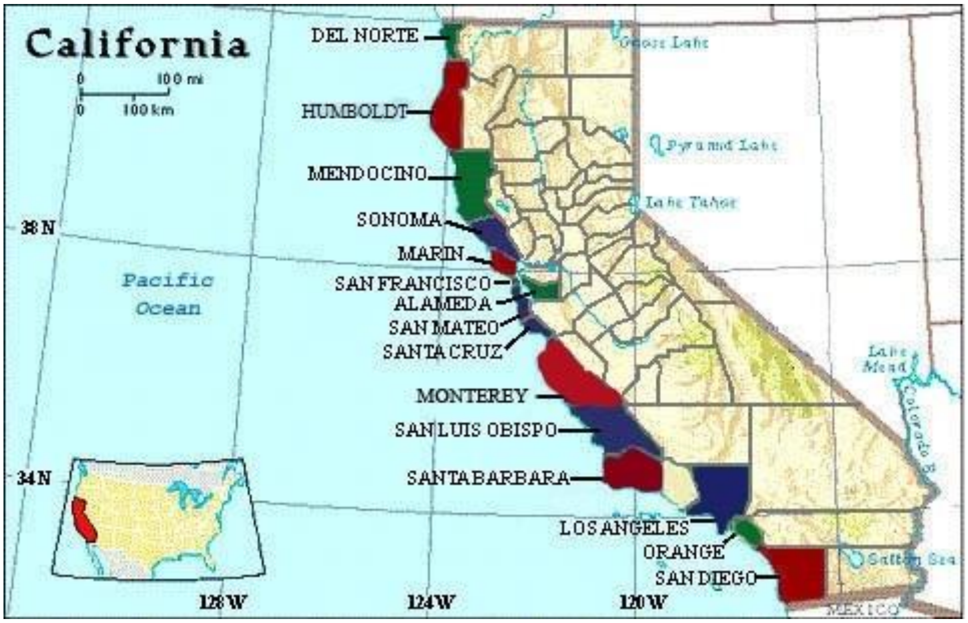


Fig 5-22. 1926-2014 Map of California's Confirmed Unprovoked Shark Attacks (N=114) [56]

The movement of a shark is affected by a number of parameters, including water temperature and salinity. We have already built a simulated physical model to analyze the relationships between shark attacks and these environmental parameters. The probability of a shark appearance will be calculated and visualized by our framework.

This case uses HDFS as the only source of data. The first data file contains the coordinates and the temperature information. The second data file contains the coordinates and the salinity information. The water temperature of the ocean is usually in the -2~30 degrees Celsius range. The mean value of the temperature in the Atlantic Ocean is 16.9 degrees Celsius. Water temperature is also related to the water’s depth. The average salinity of ocean water is 34.7 degrees.

This is the sample of the temperature and salinity data:

```
> head(geo_data_temperature)
  poi_cf.longitude poi_cf.latitude poi_cf:temperature
1        -129.9917              49          28.448851
2        -129.9833              49          29.578226
3        -129.9750              49          32.618710
4        -129.9667              49          20.189329
5        -129.9583              49           2.937873
6        -129.9500              49          23.619744

> head(geo_data_salinity)
  poi_cf.longitude poi_cf.latitude poi_cf:salinity
1        -129.9917              49        36.03154
2        -129.9833              49        44.41937
3        -129.9750              49        22.14991
4        -129.9667              49        52.35602
5        -129.9583              49        30.63463
6        -129.9500              49        31.35200
```

The geospatial data has been divided into different area files. The user calls package Rmr2 to parallel fit the probability model to calculate the probability of a shark's appearance.

```
library(rmr2)

   X_Probability =
  values(
    from.dfs(
      MapReduce (
   %% X is the data file contains the temperature and salinity values
        input = X,
        map =
          function(., Xi) {
       %%pseudo code to calculate the probability of temperature
           Temperature_probability  = cal_prob[Xi[,3]],
           %%pseudo code to calculate the probability of salinity
            Salinity_probability  = cal_prob[Xi[,4]]
           %%pseudo code to model the probability of shark
            keyval(1, Temperature_probability * Salinity_probability  ))},
        reduce = function(., YY)
            keyval(1, list(Reduce('+', YY))),
        combine = TRUE)))[[1]]
```

This MapReduce work enables the system to handle large data files in parallel mode in an R system. As shown in the results of this analytic task, the probability data of shark attacks was obtained. Thereafter, the probability map was extracted by loading the obtained data into Panoply. The probability map is shown below, with red representing a high alert area and green representing a low alert area.

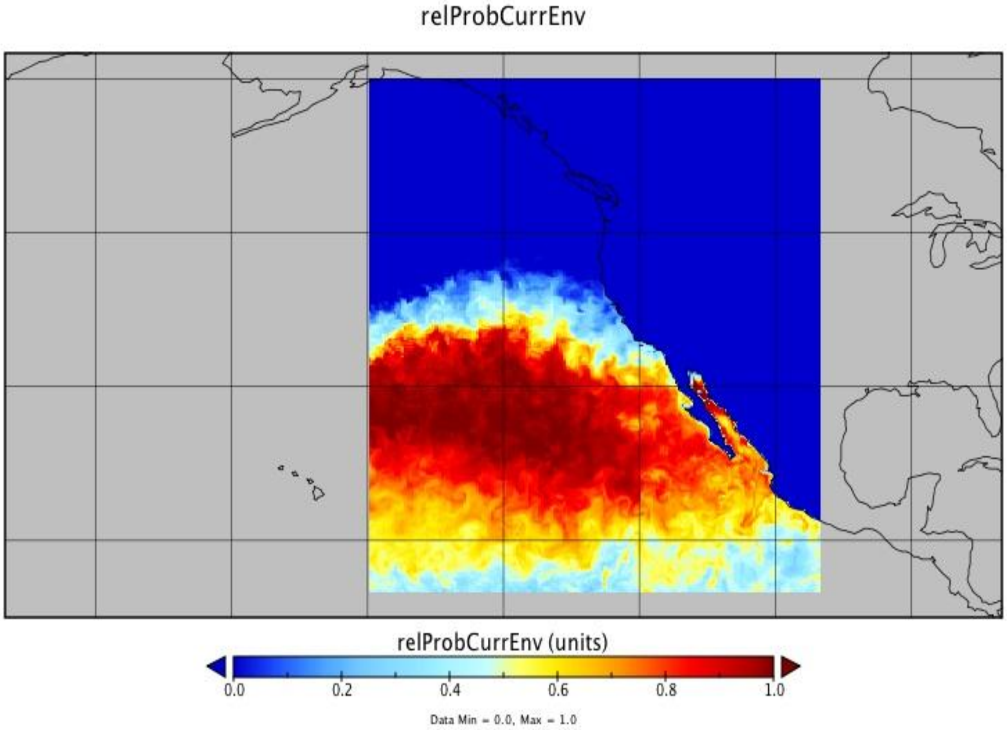


Fig 5-23. The probability map of Shark Attack

We ran multiple tests to compare our framework with a traditional analytical job. In this case the test environment was the same as in the previous case. The test objects are multiple geospatial data files, which including 20M (1file), 128M (5 files), 460M (20 files), 1200M (60 files) and 2800M (150 files). Using the traditional method, the analytical task has to be running sequentially in R. Using our framework, the analytical task can be running in a paralleled task.

This table and chart show the runtime data for the traditional method and in R-MapReduce.

data size	runtime in R(s)	runtime In R-MapReduce (s)
20M	15	18
128M	54	71
460M	280	263
1200M	1436	532
2800M	3404	1660

Table 5-2. Runtime for R analytical task

The above table shows that there is only a slight difference in performance when users analyze a small number of geospatial data files. However, when users analyze a large amount of geospatial data, R-MapReduce is significantly faster than traditional R analytical work.

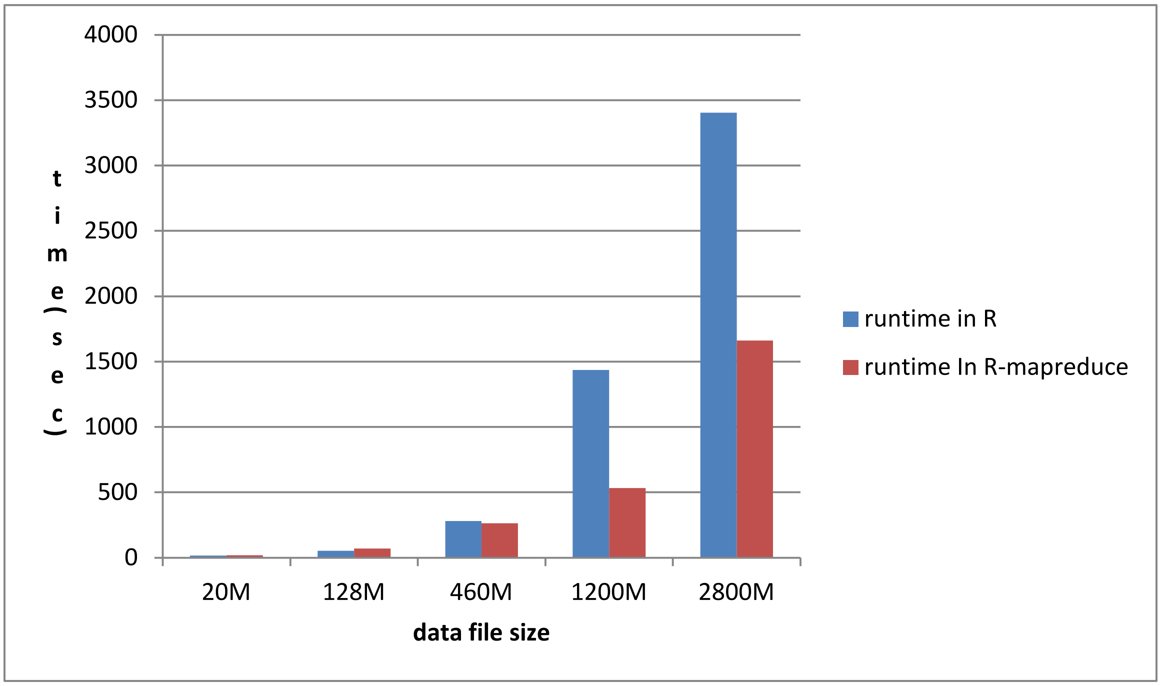


Fig 5-24.   Runtime for shark attack analytic task

The results of the performance test show that R-MapReduce has no advantage when dealing with a small number of geospatial files because R analytical work is designed to be run in memory. In contrast, The Hadoop has cost for network, job management, and file read/write. However, when users are dealing with large-scale data—especially multi files—R must divide into small files and sequentially run the analytical task. Then the R user should manually combine the results. R-MapReduce shows its great advantage over using a single R node.

CHAPTER 6 CONCLUSION AND FUTURE WORK

When combining strong data processing and analyzing features with Big Data capabilities supported by Hadoop, it is certainly worth taking a closer look at HBase and RHadoop’s features. RHadoop includes packages to integrate R with MapReduce, HDFS, and HBase, the key components of the Hadoop ecosystem. Also, the SPATSTAT package is very useful for spatial data mining analytical tasks. It is expected that geographic analysts would most benefit from the applications of large-scale geospatial data analytic methods because Big Data management and statistical analysis are the two central activities in geospatial application. In this framework, we have provided several specific methods to improve the query performance on Hbase. With the BulkLoad feature of Hbase, users can make the ETL (extract, transform and load) work over geospatial data more efficiently than by using the traditional solution. Some in-memory spatial indexes can be built on large-scale geospatial data and thus the spatial index will improve the performance of ETL work. Users also can write Map/Reduce tasks to run analytical tasks in parallel mode. The Map/Reduce feature combined with R extends the scalability of R analytical work. In general, the framework provides a universal solution for spatial analytical tasks which is needed for statisticians and researchers.

In Section 3, we presented the framework and its objectives and features. The designs of the various components that make up the framework have been presented in detail in

Section 4. HBase has been customized and optimized as the spatial data warehouse platform. HDFS plus MapReduce has been implemented as the large-scale geospatial data mining method. Section 5 presents examples on how to use the framework to analyze large-scale geospatial data and four such applications have been presented. The use case of spatial data warehouse gives a sample of high performance geospatial data query in HBase. The use case of spatial data mining shows the details of geospatial data analysis in R. The use case of Random Forest demonstrates the sample code of distributed machine learning in R and Hadoop platform. The use case of Shark Alert Map presents the method to run statistical computation among large-scale test data.

In this project, a framework that enables programmers and non-programmers to perform statistical analysis based on large-scale geospatial data has been presented. Four case studies demonstrating the strength of our framework have been outlined. These case studies comprise large-scale geospatial data handling, statistical analysis, and software component (GUI) generation. These examples have highlighted the benefits of utilizing our framework in terms of reduction in development efforts. In this framework, we have provided several methods to improve the performance for processing and analyzing large-scale geospatial data. This framework also has the potential to extend the statistical supercomputing environment, which will be widely used for various fields such as weather forecasting, climate research, oil and gas exploration, and physical simulations (for example airplane and vessel route planning) [57]. Using this framework is more a convenient, flexible, and scalable way for data analysts and statisticians to process and analyze large-scale geospatial data.

Bibliography

1. Abhishek Sharma , Drone Data Adds a New Horizon for Big Data Analytics: http://www.infoq.com/news/2014/09/drone-data-big-data-analytics
2. Geospatial Intelligence: https://en.wikipedia.org/wiki/Geospatial_intelligence
3. Elias Ioup, Zhao Yang, Brent Barre, John Sample, Kevin B. Shaw, Mahdi Abdelguerfi, "Annotating Uncertainty in Geospatial and Environmental Data", IEEE Internet Computing, vol. 19, no. , pp. 18-27, Jan.-Feb. 2015, doi:10.1109/MIC.2014.39
4. A. W. M. Van Eekeren, J. R. van Huis, P. T. Eendebak, J. Baan, Vehicle tracking in wide area motion imagery from an airborne platform, *Proc. SPIE* 9648, p. 96480I, 2015. doi:10.1117/12.2196392
5. Alexandros Labrinidis, H. V. Jagadish: Challenges and Opportunities with Big Data, Proceedings of the VLDB Endowment VLDB Endowment Hompage archive Volume 5 Issue 12, August 2012
6. Konstantin Shvachko , Hairong Kuang , Sanjay Radia , Robert Chansler: The Hadoop Distributed File System, Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium
7. Tyler Harter, Dhruba Borthakur,Siying Dong,Amitanand Aiyer,Liyin Tang, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau: Analysis of HDFS Under HBase: A Facebook Messages Case Study, FAST'14 Proceedings of the 12th USENIX conference on File and Storage Technologies
8. Anylitics, Business Intelligence and Data Management | SAS: https://www.sas.com/en_us/home.html
9. Michael G.Noll, Running Hadoop On Ubuntu linux multi-node cluster:http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/
10. Brad Hedlund, Understanding Hadoop Clusters and the Network: http://bradhedlund.com/2011/09/10/understanding-hadoop-clusters-and-the-network/
11. Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, Benjamin Reed: ZooKeeper: Wait-free coordination for Internet-scale systems, USENIXATC'10 Proceedings of the 2010 USENIX conference on USENIX annual technical conference
12. Gavin Badcock, GOOGLE'S BIGQUERY VS HADOOP: COMPLIMENTORS OR COMPETITORS? : https://gavinbadcock.wordpress.com/2013/02/06/googles-bigquery-vs-hadoop-complimentors-or-competitors/
13. Fay Chang, Jeffrey Dean,etc: Bigtable: A Distributed Storage System for Structured Data , ACM Transactions on Computer Systems (TOCS) Volume 26 Issue 2, June

2008 Article No. 4

14. Ihaka, Ross; Gentlman, Robert (Sep 1996). "R: A Language for Data Analysis and Graphics". Journal of Computational and Graphical Statistics (American Statistical Association) 5 (3): 299–314. doi:10.2307/1390807. Retrieved 12 May 2014.
15. Chambers, John M (1998). Programming with Data: A Guide to the S Language. Springer. ISBN 978-0-387-98503-9.
16. NASA Goddard Institute for Space Studies (GISS), Panoply netCDF, HDF and GRIB Data Viewer: http://www.giss.nasa.gov/tools/panoply/
17. Pavel Michna, Milton Woods: RNetCDF – A Package for Reading and Writing NetCDF Datasets, The R Journal Vol. 5/2, December
18. Cisco-based Hybrid Cloud Storage – Architecture for Trusted Cloud Outsourcing: http://cloudventures.sys-con.com/node/1555525
19. Why In-Memory Computing Is Cheaper And Changes Everything: http://timoelliott.com/blog/2013/04/why-in-memory-computing-is-cheaper-and-changes-everything.html
20. In-memory computing, Vijay Seethepalli: https://www.linkedin.com/pulse/in-memory-computing-vijay-seethepalli
21. IBM:What is In-memory computing?: www.ibm.com/software/data/what-is-in-memory-computing.htm
22. Dimitris Papadias, Panos Kalnis, Jun Zhang, Yufei Tao: Efficient OLAP Operations in Spatial Data Warehouses： Advances in Spatial and Temporal Databases Volume 2121 of the series Lecture Notes in Computer Science pp 443-459
23. Marco Morais: Spatial Data Mining： https://www.gislounge.com/spatial-data-mining/
24. Shashi Shekhar, Yan Huang, Weili Wu, C. T. Lu, S. Chawl： What’s Spatial About Spatial Data Mining: Three Case Studies， Data Mining for Scientific and Engineering Applications Volume 2 of the series Massive Computing pp 487-514
25. Michael Franklin, Alon Halevy, David Maier: From Databases to Dataspaces: A New Abstraction for Information Management, ACM SIGMOD Record Volume 34 Issue 4, December 2005
26. GeoWave Documentation: https://ngageoint.github.io/geowave/documentation.html
27. Luiz Velho, Jonas de Miranda Gomes: Digital halftoning with space filling curves, ACM SIGGRAPH Computer Graphic Volume 25 Issue 4, July 199
28. Kisung Lee, Raghu K.Ganti, Mudhakar Srivatsa, Ling Liu: Efficient spatial query processing for Big Data, SIGSPATIAL '14 Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems
29. Jeffrey Cohen, Brian Dolan: MAD Skills: New Analysis Practices for Big Data, Proceedings of the VLDB Endowment, Volume 2 Issue 2, August 2009: http://pivotal.io/big-data/pivotal-greenplum-database
30. Revolution Analytics, How to program MapReduce jobs in Hadoop with R:

http://blog.revolutionanalytics.com/2011/09/mapreduce-hadoop-r.html

31. Ahmed Eldawy and M. F. Mokbel, SpatialHadoop: towards flexible and scalable spatial processing using mapreduce; SIGMOD'14 PhD Symposium Proceedings of the 2014 SIGMOD PhD symposium
32. A. Eldawy and M. F. Mokbel. Pigeon: A Spatial MapReduce Language. In ICDE, 2014
33. Teradata Aster R, Scalable Data Science Package for R: http://www.teradata.com/products-and-services/aster-r
34. Mark Schneider, Donna Bennett, Connie Robison: Understanding the Anatomy of a SAS® Deployment: What's in My Server Soup?
35. Bill Jacobs, Using Hadoop with R: It Depends. http://blog.revolutionanalytics.com/2015/06/using-hadoop-with-r-it-depends.html
36. R and Hadoop: http://blog.fens.me/r-hadoop-book-big-data
37. Jeffrey Dean , Sanjay Ghemawat: MapReduce : Simplified Data Processing on Large Clusters, Communications of the ACM - 50th anniversary issue: 1958 – 2008, Volume 51 Issue 1, January 2008
38. Spark Streaming with HBase: http://sercanbilgic.com/
39. Adam Muise, HBase a Technical Introduction: http://www.slideshare.net/adammuise/2013-sept-17thughbasetechnicalintroduction
40. Schubert Zhang, HBase-0.20.0 Performance Evaluation: http://cloudepr.blogspot.com/2009/08/hbase-0200-performance-evaluation.html
41. A. Baddeley, E. Rubak and R.Turner. Spatial Point Patterns: Methodology and Applications with R. Chapman and Hall/CRC Press, 2015
42. How to Use Hbase Bulk Loading and Why: http://blog.cloudera.com/blog/2013/09/how-to-use-hbase-bulk-loading-and-why/
43. HBase and Schema Design,Secondary Indexes and Alternate Query Paths: http://hbase.apache.org/0.94/book/secondary.indexes.html
44. Java Spatial Index: http://jsi.sourceforge.net/
45. Geophile: https://github.com/geophile/geophile
46. Lars George, Advanced Hbase, Architecture and Schema Design, JAX UK, October 2012: http://www.slideshare.net/jaxlondon2012/hbase-advanced-lars-george
47. Revolution Analytics, RHadoop wiki: https://github.com/RevolutionAnalytics/RHadoop/wiki
48. Revolution Analytics, MapReduce in R: https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md
49. World Geodetic System: https://en.wikipedia.org/wiki/World_Geodetic_System
50. Fiseries and Oceans Canada: http://www.charts.gc.ca/data-gestion/bathy/bathymetri-eng.asp
51. Antonin Guttman: R-Trees - A Dynamic Index Structure for Spatial Searching, SIGMOD '84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data

52. Kasper Klitgaard Berthelsen, Abdollah Jalilian, Marie-Colette van Lieshout, Tuomas Rajala, Dominic Schuhmacher and Rasmus Waagepetersen: Spatstat Quick Reference guide
53. Ho, Tin Kam (1995). Random Decision Forest (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.
54. Sums of Independent Random Variables: https://www.stat.wisc.edu/courses/st311-rich/convol.pdf
55. Uri Laserson, How-to: Resample from a Large Data Set in Parallel
56. Florida Museum of Natural History: International Shark Attack File: http://www.flmnh.ufl.edu/fish/sharks/statistics/gattack/mapca.htm
57. Supercomputer: https://en.wikipedia.org/wiki/Supercomputer

Appendix

A.1 Sample code for range query based on JSI

```
package hbase_test;

import java.io.IOException;
import java.math.BigDecimal;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.filter.CompareFilter.CompareOp;
import org.apache.hadoop.hbase.filter.FilterList;
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter;
import org.apache.hadoop.hbase.util.Bytes;

import java.util.ArrayList;
import java.util.List;

import org.slf4j.*;
import com.infomatiq.jsi.*;

import gnu.trove.*;

import com.infomatiq.jsi.Rectangle;
import com.infomatiq.jsi.rtree.RTree;



public class RangeQuery {
    /**
    * range query by distance
    * @param tablename
    * @param distance
    */

   public static void main(String args[]) throws IOException {

       String tablename = "geos_test";

       Configuration conf = HBaseConfiguration.create();
       HTable table = new HTable(conf, tablename);


       //boundary query for geo-spatial data
       String cf ="poi_cf";
       String column1 = "longitude";
       String column2 = "latitude";
       String column3 = "mean";
       String column4 = "standarddeviation";
       //set POI boundary
       String long_lower = "-129.9900";
       String lat_lower = "49.0400";
       String long_upper = "-129.9250";
       String lat_upper = "49.0500";
       //set central POI

       String long_central = "-129.9450";
       String lat_central = "49.0450";
       String distance = "0.01";


       //convert data for convenience because the longitude is negative value
       String temp = long_lower;
       long_lower = long_upper;
       long_upper = temp;

       Scan s = new Scan();
       s.addFamily(Bytes.toBytes(cf));

       FilterList                list                =                new
FilterList(FilterList.Operator.MUST_PASS_ALL);
```

```
        SingleColumnValueFilter filter1 = new SingleColumnValueFilter(
            Bytes.toBytes(cf),
            Bytes.toBytes(column1),
            CompareOp.GREATER_OR_EQUAL,
            Bytes.toBytes(long_lower)
            );
        filter1.setFilterIfMissing(true);
        list.addFilter(filter1);
          SingleColumnValueFilter filter2 = new SingleColumnValueFilter(
            Bytes.toBytes(cf),
            Bytes.toBytes(column2),
            CompareOp.GREATER_OR_EQUAL,
            Bytes.toBytes(lat_lower)
        );
        list.addFilter(filter2);
        filter2.setFilterIfMissing(true);
        SingleColumnValueFilter filter3= new SingleColumnValueFilter(
            Bytes.toBytes(cf),
            Bytes.toBytes(column1),
            CompareOp.LESS_OR_EQUAL,
            Bytes.toBytes(long_upper)
        );
        list.addFilter(filter3);
        filter3.setFilterIfMissing(true);
        SingleColumnValueFilter filter4 = new SingleColumnValueFilter(
            Bytes.toBytes(cf),
            Bytes.toBytes(column2),
            CompareOp.LESS_OR_EQUAL,
            Bytes.toBytes(lat_upper)
        );
        list.addFilter(filter4);
        filter4.setFilterIfMissing(true);

        s.setFilter(list);


        ResultScanner scanner = table.getScanner(s);
        String key = new String("~");
        String keyFlag = new String("~");
        System.out.println("Scanning table... ");
        for (Result result : scanner) {
```

```
//System.out.println("getRow:"+Bytes.toString(result.getRow()));
          key = "~";
          int print_flag = 0;
          for (KeyValue kv : result.raw())
          {

             if (key.compareTo(keyFlag)==0)
             {
                 key = Bytes.toString(kv.getRow());
                 System.out.print("Key: " + key + "\r\n");


                 Get g = new Get(Bytes.toBytes(key));
                 Result r = table.get(g);
                 byte[] lon_value = r.getValue(Bytes.toBytes("poi_cf"),
                        Bytes.toBytes("longitude")
                       );
                 byte[] lat_value = r.getValue(Bytes.toBytes("poi_cf"),
                        Bytes.toBytes("latitude")
                       );
                 String valueStr_lon = Bytes.toString(lon_value);
                 String valueStr_lat = Bytes.toString(lat_value);
//                         System.out.println("GET longitude value: " +
valueStr_lon + "\r\n");
//                 System.out.println("GET latitude value: " + valueStr_lat
+ "\r\n");

                 BigDecimal lon_c= new BigDecimal(long_central);
                 BigDecimal lat_c= new BigDecimal(lat_central);
                 BigDecimal dis= new BigDecimal(distance);
                 BigDecimal lon_act= new BigDecimal(valueStr_lon);
                 BigDecimal lat_act= new BigDecimal(valueStr_lat);

                 BigDecimal bigResult = new BigDecimal(4);
                 bigResult                                              =
(lon_act.subtract(lon_c)).multiply(lon_act.subtract(lon_c)).add(

(lat_act.subtract(lat_c)).multiply(lat_act.subtract(lat_c)));
                 BigDecimal dis_sqr = new BigDecimal(4);
                 dis_sqr = dis.multiply(dis);
//                  System.out.println("GET Euclidean Distance_sqr value: "
+ bigResult + "\r\n");
//                  System.out.println("GET Radium_sqr value: " + dis_sqr +
"\r\n");

                  if (bigResult.compareTo(dis_sqr) !=1 )
                  {
                   System.out.println("Euclidean Distance_sqr " + bigResult
+ " is less than or equal to Radium_sqr " + dis_sqr + "\r\n");
                   print_flag = 1;
                  } else
                  {
                   System.out.println("Euclidean Distance_sqr " + bigResult
+ " is greater than Radium_sqr " + dis_sqr + "; This POI skipped\r\n");

                  }



              }

              if (print_flag == 1)
              {
              //System.out.print("Family                          -
"+Bytes.toString(kv.getFamily()));
              //System.out.print(",                Buffer                -
"+Bytes.toString(kv.getBuffer() ));
              //System.out.print(",      FamilyOffset      -      "      +
kv.getFamilyOffset() );
              System.out.print("POI:
"+Bytes.toString(kv.getFamily())+"."+Bytes.toString(kv.getQualifier()))
;
              System.out.print("=" +Bytes.toString(kv.getValue())+"  ");
              }



              }
          System.out.println("");
          System.out.println("-------------------");
          }
```

```
       scanner.close();
       System.out.println("Range       query       Completed\r\n       ");


      table.close();
    }
}
```

A.2 Sample code for range query based on JSI

```
package hbase_test;

import java.io.IOException;
import java.math.BigDecimal;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.filter.CompareFilter.CompareOp;
import org.apache.hadoop.hbase.filter.FilterList;
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter;
import org.apache.hadoop.hbase.util.Bytes;

import java.util.ArrayList;
import java.util.List;

import org.slf4j.*;
import com.infomatiq.jsi.*;

import gnu.trove.*;

import com.infomatiq.jsi.Rectangle;
import com.infomatiq.jsi.rtree.RTree;



public class BoundaryQuery {
    /**
     * find one row
     * @param tablename
     * @param rowkey
```

```
 */

@SuppressWarnings("resource")
public static void main(String args[]) throws IOException {

    String tablename = "geos_test";
    String rowKey = "732";

    Configuration conf = HBaseConfiguration.create();
    HTable table = new HTable(conf, tablename);

    //retrive row by rowkey
    Get g = new Get(rowKey.getBytes());
    Result rs = table.get(g);
    for (KeyValue kv : rs.raw()) {
        System.out.print(new String(kv.getRow()) + "  ");
        System.out.print(new String(kv.getFamily()) + ":");
        System.out.print(new String(kv.getQualifier()) + "  ");
        System.out.print(kv.getTimestamp() + "  ");
        System.out.println(new String(kv.getValue()));
    }


    //boundary query for geo-spatial data
    String cf ="poi_cf";
    String column1 = "longitude";
    String column2 = "latitude";
    String column3 = "mean";
    String column4 = "standarddeviation";
    //set POI boundary
    String long_lower = "-129.9550";
    String lat_lower = "49.0400";
    String long_upper = "-129.9250";
    String lat_upper = "49.0500";


    //convert data for convenience because the longitude is negative value
    String temp = long_lower;
    long_lower = long_upper;
    long_upper = temp;
```

```
       Scan s = new Scan();
       s.addFamily(Bytes.toBytes(cf));

       FilterList                list                =                new
FilterList(FilterList.Operator.MUST_PASS_ALL);
       SingleColumnValueFilter filter1 = new SingleColumnValueFilter(
           Bytes.toBytes(cf),
           Bytes.toBytes(column1),
           CompareOp.GREATER_OR_EQUAL,
           Bytes.toBytes(long_lower)
           );
       filter1.setFilterIfMissing(true);
       list.addFilter(filter1);
         SingleColumnValueFilter filter2 = new SingleColumnValueFilter(
           Bytes.toBytes(cf),
           Bytes.toBytes(column2),
           CompareOp.GREATER_OR_EQUAL,
           Bytes.toBytes(lat_lower)
       );
       list.addFilter(filter2);
       filter2.setFilterIfMissing(true);
       SingleColumnValueFilter filter3= new SingleColumnValueFilter(
           Bytes.toBytes(cf),
           Bytes.toBytes(column1),
           CompareOp.LESS_OR_EQUAL,
           Bytes.toBytes(long_upper)
       );
       list.addFilter(filter3);
       filter3.setFilterIfMissing(true);
       SingleColumnValueFilter filter4 = new SingleColumnValueFilter(
           Bytes.toBytes(cf),
           Bytes.toBytes(column2),
           CompareOp.LESS_OR_EQUAL,
           Bytes.toBytes(lat_upper)
       );
       list.addFilter(filter4);
       filter4.setFilterIfMissing(true);

       s.setFilter(list);
```

```
        ResultScanner scanner = table.getScanner(s);
        String key = new String("~");
        String keyFlag = new String("~");
        System.out.println("Scanning table... ");
        for (Result result : scanner) {

//System.out.println("getRow:"+Bytes.toString(result.getRow()));
            key = "~";
            for (KeyValue kv : result.raw()) {

                if (key.compareTo(keyFlag)==0)
                {
                    key = Bytes.toString(kv.getRow());
                    System.out.print("Key: " + key);
                }
                //System.out.print("Family                              -
"+Bytes.toString(kv.getFamily()));
                //System.out.print(",               Buffer              -
"+Bytes.toString(kv.getBuffer() ));
                //System.out.print(",      FamilyOffset      -      "      +
kv.getFamilyOffset() );
                System.out.print(",
"+Bytes.toString(kv.getFamily())+"."+Bytes.toString(kv.getQualifier()))
;
                System.out.print("=" +Bytes.toString(kv.getValue()));
            }
            System.out.println("");
            System.out.println("-------------------");
        }

        scanner.close();
        System.out.println("Boundary query Completed\r\n ");



        //full table scan
        System.out.println("Full table scan query Started ");
        Scan s_full = new Scan();
        s_full.addFamily(Bytes.toBytes("poi_cf"));
        ResultScanner scanner_full = table.getScanner(s_full);
        try {
            for (Result rr : scanner_full) {
             //System.out.println("Found row: " + rr);
          }
       } finally {
       System.out.println("Full table scan query Completed ");
          scanner.close();
       }




      table.close();
    }
}
```

A.3 Sample code for KNN(nearest neighbor) query based on JSI

```
package hbase_test;
import java.io.IOException;
import java.math.BigDecimal;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.KeyValue;
import org.apache.hadoop.hbase.client.Get;
import org.apache.hadoop.hbase.client.HTable;
import org.apache.hadoop.hbase.client.Put;
import org.apache.hadoop.hbase.client.Result;
import org.apache.hadoop.hbase.client.ResultScanner;
import org.apache.hadoop.hbase.client.Scan;
import org.apache.hadoop.hbase.filter.CompareFilter.CompareOp;
import org.apache.hadoop.hbase.filter.FilterList;
import org.apache.hadoop.hbase.filter.SingleColumnValueFilter;
import org.apache.hadoop.hbase.util.Bytes;

import java.util.ArrayList;
import java.util.List;

import org.slf4j.*;
import com.infomatiq.jsi.*;

import gnu.trove.*;

import com.infomatiq.jsi.Rectangle;
import com.infomatiq.jsi.rtree.RTree;


public class KnnQuery {
    /**
     * K-nearest neighbor query
     * @param tablename
     * @param k
     */

    public static void main(String args[]) throws IOException {
        String tablename = "geos_test";

        Configuration conf = HBaseConfiguration.create();
        HTable table = new HTable(conf, tablename);



        //boundary query for geo-spatial data
        String cf ="poi_cf";
        String column1 = "longitude";
        String column2 = "latitude";
        String column3 = "mean";
        String column4 = "standarddeviation";

        //set POI boundary
        String long_lower = "-129.9000";
        String lat_lower = "49.0400";
        String long_upper = "-128.0250";
        String lat_upper = "49.5500";
        //set central POI
        String long_central = "-129.9450";
        String lat_central = "49.0450";
        //set number K
        int k =10;


        //convert data for convenience because the longitude is negative value
        String temp = long_lower;
        long_lower = long_upper;
        long_upper = temp;



        //boundary query to filter data
        Scan s = new Scan();
        s.addFamily(Bytes.toBytes(cf));

        FilterList                list                =                new
FilterList(FilterList.Operator.MUST_PASS_ALL);
        SingleColumnValueFilter filter1 = new SingleColumnValueFilter(
               Bytes.toBytes(cf),
               Bytes.toBytes(column1),
               CompareOp.GREATER_OR_EQUAL,
               Bytes.toBytes(long_lower)
               );
        filter1.setFilterIfMissing(true);
        list.addFilter(filter1);
        SingleColumnValueFilter filter2 = new SingleColumnValueFilter(
               Bytes.toBytes(cf),
               Bytes.toBytes(column2),
               CompareOp.GREATER_OR_EQUAL,
               Bytes.toBytes(lat_lower)
               );
        list.addFilter(filter2);
        filter2.setFilterIfMissing(true);
        SingleColumnValueFilter filter3= new SingleColumnValueFilter(
               Bytes.toBytes(cf),
               Bytes.toBytes(column1),
               CompareOp.LESS_OR_EQUAL,
               Bytes.toBytes(long_upper)
               );
        list.addFilter(filter3);
        filter3.setFilterIfMissing(true);
        SingleColumnValueFilter filter4 = new SingleColumnValueFilter(
               Bytes.toBytes(cf),
               Bytes.toBytes(column2),
               CompareOp.LESS_OR_EQUAL,
               Bytes.toBytes(lat_upper)
               );
        list.addFilter(filter4);
        filter4.setFilterIfMissing(true);

        s.setFilter(list);


        BigDecimal array_poi[][];
        array_poi =  new BigDecimal[99999][5];

        //get POI data
        ResultScanner scanner = table.getScanner(s);
        String key = new String("~");
        String keyFlag = new String("~");
        System.out.println("Scanning table... ");
        int count_poi=-1;
        for (Result result : scanner) {

    //System.out.println("getRow:"+Bytes.toString(result.getRow()));
            key = "~";
            int print_flag = 0;
            int column = -1;
            for (KeyValue kv : result.raw())
            {

                if (key.compareTo(keyFlag)==0)
                {
                    key = Bytes.toString(kv.getRow());
                    System.out.print("Key: " + key + "\r\n");

                    print_flag=1;
                    count_poi++;

                    System.out.println("count_poi is: " + count_poi +
"\r\n");
                    array_poi[count_poi][0]=new BigDecimal(key);

                    column = 1;
                }


                if (print_flag == 1)
                {
                    if (column < 5) {
                        //System.out.print("Family -
"+Bytes.toString(kv.getFamily()));
                        //System.out.print(", Buffer -
"+Bytes.toString(kv.getBuffer() ));
                        //System.out.print(", FamilyOffset - " +
kv.getFamilyOffset() );
                        System.out.print("POI:
"+Bytes.toString(kv.getFamily())+"."+Bytes.toString(kv.getQualifier()))
;
                        System.out.print("="
+Bytes.toString(kv.getValue())+"  ");
                        array_poi[count_poi][column]=new
BigDecimal(Bytes.toString(kv.getValue()));
                        column ++ ;
                    } else {
                        column = 1;
                    }
                }

            }
            System.out.println("");
            System.out.println("-------------------");
        }

        scanner.close();


        //define R-tree
        int count_real = count_poi+1;
        System.out.println("Boudary query Completed, The number of POIs is:"
+ count_real +" \r\n ");

        System.out.println("Creating rectangles of R-tree \r\n");
        long start,end;
        float offset = 0.0005f;

        final Rectangle[] rects = new Rectangle[count_poi];
        int id = 0;
        for (int i = 0; i < count_poi; i++)
        {
            System.out.print("longitude value: " +
array_poi[i][2].floatValue() + " latitude value: " +
array_poi[i][1].floatValue() + "\r\n");
            rects[id++] = new Rectangle(array_poi[i][2].floatValue(),
array_poi[i][1].floatValue(), array_poi[i][2].floatValue()+ offset,
array_poi[i][1].floatValue() + offset); //
        }

        System.out.print("Indexing " + count_real + " rectangles\r\n");
        start = System.currentTimeMillis();
        SpatialIndex si = new RTree();
        si.init(null);
        for (id=0; id < count_poi; id++) {
            si.add(rects[id], id);
        }
        end = System.currentTimeMillis();
        System.out.print("Average time to index rectangle = " + ((end - start)
/ (count_poi / 1000.0)) + " us\r\n");


        // Run a performance test, find the k nearest rectangles
        float long_central_f = Float.valueOf(long_central);
        float lat_central_f = Float.valueOf(lat_central);
        final Point p = new Point(long_central_f, lat_central_f);
        System.out.print("Querying for the nearest " + k + " rectangles to
" + p + " \r\n");
        si.nearestN(p, new TIntProcedure() {
            public boolean execute(int i) {
                System.out.print("Rectangle " + i + " " + rects[i] + ",
distance=" + rects[i].distance(p)+ " \r\n");
                return true;
            }
        }, k, Float.MAX_VALUE);


        // Run a performance test, find the k nearest rectangles
        final int[] ret = new int[1];
        System.out.print("Running 10000 queries for the nearest " + k + "
rectangles\r\n");
        start = System.currentTimeMillis();
        for (int row = 0; row < 100; row++) {
            for (int column = 0; column < 100; column++) {
                p.x = row + 0.6f;
                p.y = column + 0.7f;
                si.nearestN(p, new TIntProcedure() {
                    public boolean execute(int i) {
                        ret[0]++;
                        return true; // don't do anything with the results,
for a performance test.
                    }
                }, k, Float.MAX_VALUE);
            }
        }
        end = System.currentTimeMillis();
        System.out.print("Average time to find nearest " + k + " rectangles
= " + ((end - start) / (10000 / 1000.0)) + " us\r\n");
        System.out.print("total time = " + (end - start) + "ms\r\n");
        System.out.print("total returned = " + ret[0]+ " \r\n");


        table.close();

    }


}
```

VITA

The author was born in Shanxi, China. He obtained his Bachelor Degree in Computer Science from Northwestern Polytechnical University, Xi’an China. He received a M.S. in Computer Science from the University of New Orleans in 2012. He was enrolled in the graduate program in Computer Science at the University of New Orleans since August 2010.