Demonstration of Hadoop-GIS: A Spatial Data Warehousing System Over MapReduce

Ablimit Aji[†] Xiling Sun[#] Hoang Vo[†] Qioaling Liu[†] Rubao Lee[*] Xiaodong Zhang[*]
Fusheng Wang[†] Joel Saltz[†]
[†]MathCS and BMI, Emory University
[#]EECS, Northwestern University
[*]CSE, Ohio State University
{aaji,vho8,qliu26,fusheng.wang,jhsaltz}@emory.edu, xilingsun2013@u.northwestern.edu,
{liru,zhang}@cse.ohio-state.edu

ABSTRACT

The proliferation of GPS-enabled devices, and the rapid improvement of scientific instruments have resulted in massive amounts of spatial data in the last decade. Support of high performance spatial queries on large volumes data has become increasingly important in numerous fields, which requires a scalable and efficient spatial data warehousing solution as existing approaches exhibit scalability limitations and efficiency bottlenecks for large scale spatial applications.

In this demonstration, we present *Hadoop-GIS* – a scalable and high performance spatial query system over MapReduce. Hadoop-GIS provides an efficient spatial query engine to process spatial queries, data and space based partitioning, and query pipelines that parallelize queries implicitly on MapReduce. Hadoop-GIS also provides an expressive, SQL-like spatial query language for workload specification. We will demonstrate how spatial queries are expressed in spatially extended SQL queries, and submitted through a command line/web interface for execution. Parallel to our system demonstration, we explain the system architecture and details on how queries are translated to MapReduce operators, optimized, and executed on Hadoop. In addition, we will showcase how the system can be used to support two representative real world use cases: large scale pathology analytical imaging, and geo-spatial data warehousing.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial Database and GIS, Scientific databases*

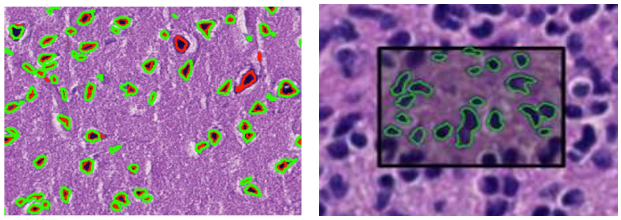
Keywords

Database, Data Warehouse, Spatial Query Processing, MapReduce, Scientific Data Management, Analytical Imaging, Hive

1. INTRODUCTION

The prevalence of cost effective and ubiquitous positioning technologies such as GPS, RFID, and more recently mobile phones, have enabled enterprises, governments and scientists to capture spatially oriented data at an unprecedented scale and rate. Collaborative efforts, such as OpenStreetMap, are also creating large volumes spatial data which are being utilized for many applications. Managing and querying large amounts of spatial data to derive value, and guide decision making have become essential for business success and scientific progress.

For example, digital pathology imaging is an emerging discipline, in which examination of high resolution images of tissue specimens enables novel, more effective ways of screening for disease, understanding and classifying disease states. Pathology image analysis offers a means of reliably and rapidly carrying out quantitative, reproducible measurements of micro-anatomical features in high-resolution pathology images. Biomedical professionals are interested in studying the spatial relationship between micro-anatomical objects such as nuclei, cells, and blood vessels which serves as "proxy" for various disease conditions. In a typical analysis scenario, regions of micro-anatomic objects are computed through image segmentation algorithms, represented with their *boundaries*, and image *features* are extracted from these objects. Exploration of such analysis results often involves complex queries such as spatial cross-matching (Figure 1(a)), overlay of multiple sets of spatial objects, feature aggregation within a spatial range (Figure 1(b)), and queries for global spatial pattern discovery [3]. These queries often involve millions to billions of spatial objects and heavy geometric computation.



(a) Spatial cross matching of multiple datasets (green vs. red) (b) Region based feature analysis and aggregation

Figure 1: Example spatial query cases in analytical medical imaging

With the rapid improvement of instrument resolutions, and the large scope of observed data, spatial queries have become increasingly compute-intensive and data-intensive. Consequently, modern spatial applications face two major challenges. Specifically, *High Computational Complexity* challenge and the *Big Spatial Data* challenge [3, 4].

Confluence of these challenges requires a scalable and efficient

SIGSPATIAL'13, Nov. 05–08 2013, Orlando, FL, USA
ACM 978-1-4503-2521-9/13/11.
http://dx.doi.org/10.1145/2525314.2525320.

data management architecture. Recently, MapReduce based systems have emerged as a scalable and cost effective solution for massively parallel processing. Hadoop – the open source implementation of MapReduce, has been successfully applied in enterprise settings to support big data analytics. Declarative query interfaces such as Hive, Pig, and Scope have brought the large scale data analysis one step closer to common users by providing high level, developer friendly programming abstractions to MapReduce.

Motivated by those challenges and our application requirements, we have developed *Hadoop-GIS* [2, 3, 4]– a spatial data warehousing system over MapReduce. The goal of the system is to deliver a scalable, efficient, expressive spatial querying system for efficiently supporting analytical queries on large scale spatial data, and to provide a feasible solution that can be afforded for daily operations. Hadoop-GIS integrates a native spatial query engine with MapReduce, where spatial queries are implicitly parallelized through MapReduce through space partitioning, and spatial indexing is built on demand to accelerate the queries – the overhead on index building is only a small fraction of overall query cost for most queries. By integrating the framework with Hive, Hadoop-GIS provides an expressive spatial query language by extending HiveQL [5] with spatial constructs, and automates spatial query translation, optimization and execution. Hadoop-GIS supports common spatial queries (selection, join, projection and aggregation), and complex queries such as spatial cross-matching and nearest neighbor queries.

In this demonstration, we show how the system processes various spatial queries based on two real world applications: pathology analytical imaging, and large spatial parcels extracted from OpenStreetMap. We will demonstrate the efficiency and scalability of Hadoop-GIS by running queries on a large cluster. We will also prepare a poster to illustrate how spatial queries are translated to a set of operators, and how they are optimized and executed.

2. SYSTEM OVERVIEW

The core of Hadoop-GIS is a partition based spatial query engine [3, 4] that supports diverse spatial queries with optimal access methods in the MapReduce framework. Spatial queries are translated into series of MapReduce code, and can be run in parallel on a large number of cluster nodes.

2.1 Architecture

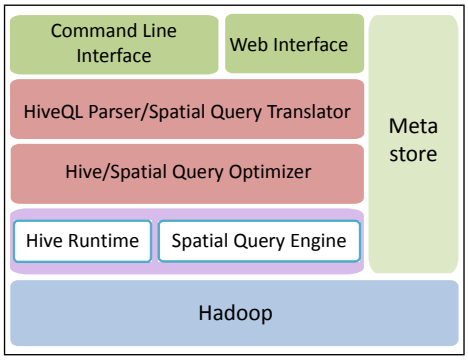


Figure 2: Architecture of Hadoop-GIS

Figure 2 shows an architectural overview of Hadoop-GIS. Users interact with the system by submitting SQL queries either from a command line or web interface; the queries are parsed and translated into an operator tree by the spatial query translator, and the query optimizer applies heuristics optimization rules to generate an optimized query plan. For a query with spatial query operator, corresponding MapReduce code are generated, which call an appropriate spatial query pipeline supported by the spatial query engine. Generated MapReduce code are submitted to the execution engine for execution. Spatial data is partitioned based on the spatial attribute which specified in the table definition, and staged to the HDFS system for parallel access. There are several key components in Hadoop-GIS that are developed to provide spatial query processing capability.

- *Spatial Query Translator* parses and translates SQL queries into an abstract syntax tree. We extended the HiveQL translator to support a set of spatial query operators, spatial functions, and spatial data types.
- *Spatial Query Optimizer* takes an operator tree as an input and applies rule based optimizations such as predicate push down or index-only query processing.
- *Spatial Query Engine* is a stand-alone spatial query engine which supports following infrastructure operations: i) spatial relationship comparison, such as *intersects*, *touches*, *overlaps*, *contains*, *within*, *disjoint*, ii) spatial measurements, such as *intersection*, *union*, *convexHull*, *distance*, *centroid*, *area*, etc; iii) creating and querying spatial access methods, such as R^**-Tree*, for efficient query processing.

2.2 Query Language

Declarative query language interfaces to MapReduce, such as Hive, Pig and Scope, have gained much momentum in recent years. Data scientists and applications developers are also more comfortable with SQL queries compared to pure programming interfaces. Moreover, most spatial applications are backed by spatial database systems which use SQL as the primary query interface. Therefore, we extended HiveQL, a SQL-like query language to Hive, to provide a high level and easy to use query interface to Hadoop-GIS.

The query language inherits major operators and functions from ISO SQL/MM Spatial, and extends it for more complex pattern queries and data partitioning constructs to support parallel query processing in MapReduce. Major spatial operations include spatial query operators, spatial functions, and spatial data types. The spatial query operators include topology based spatial relationships, such as `ST_INTERSECTS`, `ST_CONTAINS`, `ST_TOUCHES`, and nearest neighbor operator `ST_KNN`. They can be categorized into two main types: binary operators to find spatial topology relationship between two spatial objects, and aggregate operators to find spatial patterns among a group of spatial objects. The spatial functions include unary functions, for example `ST_AREA`; binary functions such as `ST_DISTANCE`, and aggregate functions. Currently supported spatial data types include `Point`, `Polygon`, `Box`, and `LineString`.

We also developed query constructs which facilitate the parallel query processing by partitioning the input data on spatial attribute. The query language provides a `PARTITION BY` clause, which specifies a partition attribute on which the data is partitioned, and a partitioning method by which the input data is partitioned. So far, the systems only supports regular fixed grid based tile partitioning. In future, we are planning to implement other optimal partitioning approaches.

Example queries include: i) Spatial feature aggregation: aggregation or summary statistics on computed features or spatial attributes. ii) Spatial range query: find spatial object(s) which contained in a specified space (possibly followed by an aggregation query). iii) Spatial join and spatial cross-matching: find correlations between multiple sets based on topology relationships of the spatial objects. In a typical spatial data warehousing scenario, user

may also need to work with other business data sources that are warehoused in the same system with spatial data. In such case, users can write a mixture of spatial query and non-spatial query without switching to a different system. Hadoop-GIS supports nested and mixed queries in a streamlined fashion.

In future, we are planning to extend those 2D query constructs to support 3D spatial application.

2.3 Storage Layer

At the storage layer, the original HDFS is kept intact to insure backward compatibility with existing Hadoop platforms. However, the internal organization of the spatial data is optimized to insure better performance. Specifically, after tile based partitioning, each record is assigned a internal *tile id* which indicates which tile this record belongs to. There are two advantages of such tile based storage organization. First, it servers as a logical parallelization unit for task creation and assignment. Second, similar to the range partitioning in parallel database systems, it works as a "filtering" mechanism to avoid redundant processing.

We also extend Hive index utility for creating persistent spatial index on the spatial columns. This index data is stored in separate internal table which later used by the query planner for query optimization. Since certain queries, such as window query, can be processed more efficiently with the help of a spatial index, this would increase the spatial query performance. Moreover, for queries that can be answered only using an index, the I/O overhead can be significantly reduced by only scanning the index data which has much smaller storage footprint compared to the whole table scan.

2.4 Query Processing

Hadoop-GIS uses the traditional *plan-first, execute-next* approach for query processing. There are three key steps: query translation, logical plan generation, and physical plan generation. To process a query expressed in SQL, the system first parses the query and generates an equivalent abstract syntax tree representation of the query. Preliminary query analysis is performed in this step to ensure that the query is syntactically and grammatically correct, and table metadata information is valid. Next, the abstract syntax tree is translated into a logical plan which is expressed as an operator tree, and simple query optimization techniques such as predicate push down, and column pruning are applied in this step. Currently, system has a simple rule based query optimizer which has limited capability. How to incorporate a cost based query optimizer is an ongoing research effort. Then, a physical plan is generated from the operator tree which eventually consists of series of MapReduce jobs. Finally, the generated MapReduce jobs are submitted to the Hive runtime for execution.

Major differences between Hive and Hadoop-GIS are in the logical plan generation step. If a query does not contain any spatial operations, the resulting logical query plan is exactly the same as the one generated from Hive. However, if the query contains spatial operations, the logical plan is regenerated with special handling of spatial operators. Specifically, two additional steps are performed to rewrite the query. First, operators involving spatial operations are replaced with internal spatial query engine operators. Second, serialization/deserialization operations are added before and after the spatial operators to prepare Hive for communicating with spatial query engine.

An example query plan is given in Figure 3, which is generated by translating the SQL query in Figure 6. As the query plan shows, the query translator generates the *table scan* operators in the first step; during this process, query predicates are applied to filter the input data; then, upon determining this is a spatial join operator, the query translator generates a tile based join processing workflow, where each tile forms a simple join task; each such task is marked for executing in the spatial query engine, and appropriate input/output data formats are specified; each task is scheduled for execution on Hadoop; after the spatial query engine finishes processing a tile, the result is returned, and the execution privilege is returned to the Hive execution engine which continues the processing task. As the workflow shows, the interaction between spatial query engine and Hive execution engine is transparent to the user.

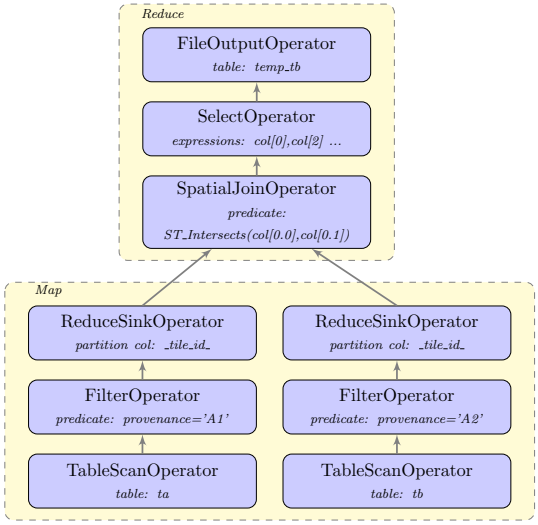


Figure 3: Two-way Spatial Join Query Plan

2.5 Software Setup

Hadoop-GIS is designed to be completely hot-swappable with Hive. Hive users only need to deploy the spatial query engine on Hadoop cluster nodes, and turn the spatial query processing switch on. To use the system, users can follow the same user guidelines of using Hive. Any query that runs on Hive, runs on Hadoop-GIS without any modification. Current version of Hadoop-GIS utilizes a set of UDFs from *ESRI GIS-tools-for-hadoop* [1] library.

Schema Creation: Users create all the necessary table schema depending on their warehousing need. The schema information is stored in the metastore. Spatial columns need to be specified with corresponding data types defined in ISO SQL/MM Spatial. Optionally users can specify spatial partition column to speed up query processing. An example SQL query for creating the example table schema is given in Figure 4.

```
1: CREATE TABLE tcga_markups ( markup_id BIGINT,
2:       provenance STRING, hand_marked BOOLEAN,
3:       center ST_POINT, polygon ST_POLYGON )
4: PARTITIONED BY TILE(polygon, 4096, 4096)
5: ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
6: STORED AS TEXTFILE ;
```

Figure 4: An example table schema

Data Loading: Then users can upload data which will be used to populate the schema. Figure 5 shows an example. Data can be read either from a local path or a HDFS path.

```
1: LOAD DATA [LOCAL] INPATH '/data/tcga.wkt'
2: [OVERWRITE] INTO TABLE tcga_markups ;
```

Figure 5: Data loading command

Querying: Users can submit SQL queries from the terminal within Hive Shell, or from a web interface which designed for interactively

viewing and analyzing data. Here, Figure 6 illustrates a spatial join query for evaluating to two image segmentation results.

```
1: SELECT
2:   ST_AREA(ST_INTERSECTION(ta.polygon,tb.polygon))/
3:   ST_AREA(ST_UNION(ta.polygon,tb.polygon)) AS ratio,
4:   ST_DISTANCE(ST_CENTROID(tb.polygon),
5:   ST_CENTROID(ta.polygon)) AS distance,
6: FROM  tcga_markups ta  JOIN  tcga_markups tb
7:   ON (ST_INTERSECTS(ta.polygon, tb.polygon) = TRUE)
8: WHERE
9:      ta.provenance='A1' AND tb.provenance='A2' ;
```

Figure 6: A spatial join query in HiveQL

3. PERFORMANCE

In this section, we briefly discuss performance of Hadoop-GIS and demonstrate its efficiency by running query shown in Figure 6. For this test we use our in-house cluster, and a set of 18 images that are segmented with two image analysis algorithms to generate the spatial data. The average number of polygons per image per result set is about 0.5M, and each dataset size is about 21.5 GB.

A spatial join query shown in Figure 6 is issued to the system and query runtime is measured. System performance is illustrated in Figure 7. As the figure shows, Hadoop-GIS is a very scalable and efficient system for querying large scale spatial data. Here we skip other test details, and refer interested readers to [3, 4], in which we provide a detailed evaluation of the system performance, and discuss related technical details.

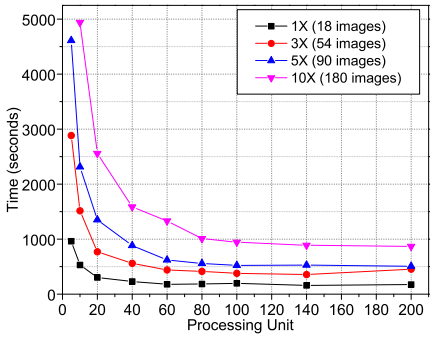


Figure 7: Join query performance

Performance of containment query is shown in Figure 8 where the horizontal axis indicates containment query size; the vertical axis indicates number of scanned partitions to process a query; and column labels denote the query processing time. As the figure shows, Hadoop-GIS has a good query performance, and can economically utilize I/O resources.

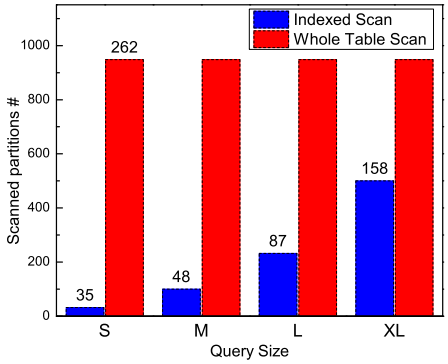


Figure 8: Containment query performance

4. DEMONSTRATION DETAILS

We present an end-to-end implementation of Hadoop-GIS, and the demonstration has two main components. First, we illustrate how a spatial query is translated into an operator tree, and how the MapReduce codes are generated using this operator tree. Second, we illustrate system scalability and high performance by running a set of pre-defined spatial queries on a large cluster.

4.1 Demo Setup

Hadoop-GIS Clusters. We will setup two clusters for the demonstration. One is a medium scale cluster which comes with 8 physical nodes and 192 cores which is deployed in our institution. This cluster is mainly used for demonstrating system performance for smaller scale queries. We will also have a $20 \sim 50$ node cluster which will be hosted on Amazon EC2 for running large scale queries and demonstration of system scalability.

Data Sets. We will stage 70GB (spatial + features) of data which are derived from a set of 18 brain tumor images. This dataset contains roughly 40 million spatial objects and each object has 74 features associated with it. This is a medium scale dataset which we will use to demonstrate the system scalability on expensive spatial queries. We also use another large scale parcel dataset which is extracted from OpenStreetMap dump. This dataset is roughly 300GB, and it contains few hundred million spatial objects. Both datasets are stored and processed in standard WKT format.

User Interface. We will provide a command line interface where users can view, select the data, issue query, and monitor job progress. We will also prepare a poster to illustrate some technical details such as query parsing, plan generation and optimization.

4.2 Demonstration Scenarios

One example application we will focus on in the demonstration is a pathology analytical imaging "GIS" use case [3], in which large scale image derived spatial data need to be warehoused to support analytics. We will first demonstrate the use case by showing spatial objects of pathology images of large scale in a digital slide viewer. Example queries include: i) The researcher wants to compute the average features such as area and perimeter of all nuclear objects contained in tumor regions. ii) The algorithm developer wants to compare the results of two algorithms from the same set of images, by computing the overlapping ratios of two sets of polygons representing nuclear boundaries. This is performed by a spatial join query (Figure 6) to cross match spatial objects; iii) A pathologist wants to identify nearest blood vessels for each stem cell and performs feature aggregation. We will demonstrate these queries interactively by explaining the whole query processing pipeline, and visualization of input/output data and system performance.

Acknowledgments

This work is supported in part by Grant Number R01LM009239 from NLM, by Contract No. HHSN261200800001E from NCI, and by Google through the Summer of Code program.

5. REFERENCES

[1] Esri gis-tools-for-hadoop. https://github.com/Esri/gis-tools-for-hadoop, 2013.

[2] Hadoop-gis wiki. https://web.cci.emory.edu/confluence /display/hadoopgis, 2013.

[3] A. Aji, F. Wang, and J. H. Saltz. Towards Building A High Performance Spatial Query System for Large Scale Medical Imaging Data. In *SIGSPATIAL/GIS*, pages 309–318, 2012.

[4] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. volume 6. VLDB Endowment, 2013.

[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. volume 2, pages 1626–1629, August 2009.