EMORY UNIVERSITY

Center for Comprehensive Informatics

Technical Report

Hadoop-GIS: A High Performance Spatial Query System for Analytical Medical Imaging with MapReduce

Fusheng Wang

Ablimit Aji

Qiaoling Liu

Joel H. Saltz

CCI-TR-2011-3
August 8, 2011

Hadoop-GIS: A High Performance Spatial Query System for Analytical Medical Imaging with MapReduce

Fusheng Wang[1] Ablimit Aji[2] Qiaoling Liu[2] Joel H. Saltz[1]

[1]Department of Biomedical Informatics, Emory University
[2]Department of Mathematics and Computer Science, Emory University
{fusheng.wang, ablimit.aji, qiaoling.liu, jhsaltz}@emory.edu

ABSTRACT

Querying and analyzing large volumes of spatially oriented scientific data becomes increasingly important for many applications. For example, analyzing high-resolution digital pathology images through computer algorithms provides rich spatially derived information of micro-anatomic objects of human tissues. The spatial oriented information and queries at both cellular and sub-cellular scales share common characteristics of "Geographic Information System (GIS)", and provide an effective vehicle to support computer aided biomedical research and clinical diagnosis through digital pathology. The scale of data could reach a million derived spatial objects and hundred million features for a single image. Managing and querying such spatially derived data to support complex queries such as image-wise spatial cross-matching queries poses two major challenges: the high complexity of geometric computation and the "big data" challenge. In this paper, we present a system *Hadoop-GIS* to support high performance declarative spatial queries with MapReduce. Hadoop-GIS provides an efficient real-time spatial query engine *RESQUE* with dynamically built indices to support on the fly spatial query processing. To support high performance queries with cost effective architecture, we develop a MapReduce based framework for data partitioning and staging, parallel processing of spatial queries with RESQUE, and feature queries with Hive, running on commodity clusters. To provide a declarative query language and unified interface, we integrate spatial query processing into Hive to build an integrated query system. Hadoop-GIS demonstrates highly scalable performance to support our query cases.

1. INTRODUCTION

Data-oriented scientific research largely relies on efficient accurate analysis of data generated through observations or computer simulations [10]. For example, Large Synoptic Survey Telescope projects [3] and earthquake analysis and simulation generate large volume of spatial and temporal oriented data. Human atlas projects provide 3D spatial modeling of objects in human bodies such as brain and heart [1].

In the past decade, devices that can acquire high-resolution images from whole tissue slides have become more affordable, faster, and practical. Together with the advances in "omic" data such as genomics and proteomics and radiology imaging, this revolutionized the medical professional's ability to rapidly capture vast amount of multi-scale, multi-dimensional data on each patient's genetic background, biological function and structure. As this decade progresses, significant advances in medical information technologies will be needed to transform very large volumes of multi-scale, multi-dimensional data into actionable information to drive the discovery, development, and delivery of new mechanisms of preventing, diagnosing, and healing complex disease. As discussed next, the huge amount of spatially derived information from pathology images contains essential knowledge to support biomedical research and clinical diagnosis, and poses major challenges for data management and queries.

1.1 Digital Pathology Imaging

The morphology of nuclei from pathology whole slide images is a pivotal attribute used to classify tumors. For example, for brain tumors (glioblastoma), nuclei appear to be round shaped with smooth regular texture in oligodendrogliomas, whereas they are generally more elongated with rough and irregular texture in astrocytomas. The classifications of brain tumor nuclei based on morphology are linked to genetic and gene expression classifications. The characterizations and classifications of micro-anatomic objects or regions offer tremendous potential to assess patient survival and response to treatment [19, 25].

Pathology image analysis offers a means of rapidly carrying out quantitative, reproducible measurements of micro-anatomical features in high-resolution pathology images and large image datasets. Systematic analysis of large-scale image data can involve many interrelated analyses on hundreds to tens of thousands of images, generating tremendous amount of quantifications such as shape and texture, as well as classifications of the quantified features. Figure 1 shows an example workflow on integrative brain tumor studies with pathology images. In this example, regions (or markups) of nuclei are computed through image segmentation algorithms, represented with their boundaries, and image features are extracted from these regions [26]. Classifications are computed based on image features or region classification algorithms. The derived data could be classified into two categories: *markups* – spatial boundaries for delineating objects, and *annotations* – features, classifications and observations derived from images or assessed by humans. In order to correlate micro-anatomic morphometry with molecular profiles and clinical outcome, summary statistics on image features are computed for each image. This process involves calculating the mean feature vectors and the feature covariance values of all

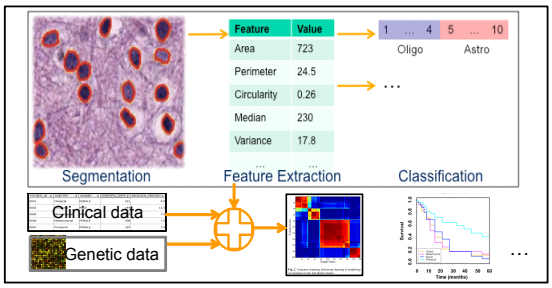


Figure 1: Integrative Biomedical Study with Pathology Images

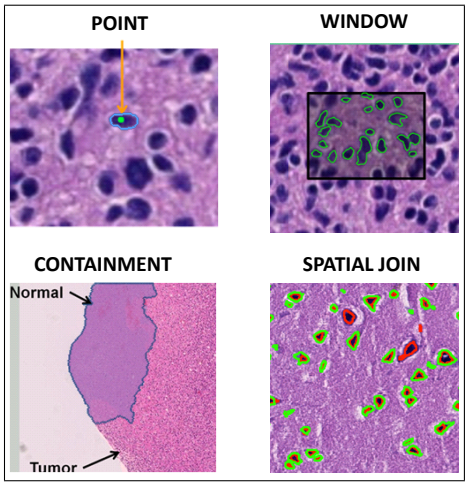


Figure 2: Example Spatial Query Cases in Analytical Medical Imaging

possible feature pairs over all nuclei for every image.

Algorithm evaluation. To support computer aided diagnosis of diseases with the emerging pathology imaging technology, it is essential to develop and evaluate high quality image analysis algorithms, in following scenarios: i) Algorithm Validation. Algorithms are tested, evaluated and improved in an iterative manner by validating algorithm results such as segmentations with human annotations made by pathologists. ii) Algorithm Consolidation. Multiple algorithms can be developed in a study to solve the same problem. Different algorithm results are aggregated to generate more confident analysis results. iii) Algorithm Sensitivity Studies. An algorithm often includes a set of parameters that can be adjusted to adapt to different types, resolutions, and qualities of images. Exploring the sensitivity of analysis output with respect to parameter adjustments can provide a guideline for the best deployment of algorithms in different scenarios and for rapid development of robust algorithms. One essential task for such evaluation is to cross-match spatial boundaries of micro-anatomic objects such as nuclei segmented by different approaches, such as different algorithms and varied parameters.

1.2 Medical Imaging GIS and its Challenges

Pathology image analysis produces large scale spatially derived information, and share common characteristics of "GIS" flavor queries. Figure 2 demonstrates a few common queries: i) *Point Query*: a simple human marked point can be used to identify an object containing the point; ii) *Window Query*: a visualization tool can highlight objects contained in a window from a whole slide image with high dimension; iii) *Containment Query*: Objects contained in certain regions (e.g., tumor regions) pre-marked by humans or segmented by region classification algorithms are used for evaluation; and iv) *Spatial Join Query (Spatial Cross-Matching)* : to compare and consolidate (segmentation) algorithm results, multiple result sets will be cross-matched to compute the extent of intersection. For example, a query to compute the distance and intersection ratio of intersected boundaries segmented from an image by different algorithms is a common query type, among one of the most expensive query types. To compare two results from a single image, we are cross-matching a million spatial objects with another million spatial objects.

1.2.1 Use Cases

Scientific research is an exploratory process in which large amount of preliminary analytical results may be quickly generated for validating different approaches or initial discoveries, and validated and curated results may need to be made persistent for archiving, querying and sharing. Similarly for digital pathology imaging, there are two major scenarios:

Scenario 1*: Data models and data management architecture to manage image data products, feature sets and results from computer algorithms. The data management infrastructure will provide public shared data archives with well-understood results and algorithm performance and to support further studies and algorithm evaluation in a community of biomedical researchers.*

Scenario 2*: High performance computing architecture for result analysis, mining and evaluation. This is for on the fly data processing for algorithm validation and comparison, or discovery of preliminary results. In this case, short response time for queries is needed, cost effective architecture is preferred, and results are transient and may not need to be made persistent.*

1.2.2 "Big Data" Challenges

Pathology images such as whole-slide images made by scanning microscope slides at diagnostic resolution are very large: a typical WSI may contain 100,000x100,000 pixels. One image may contain millions of objects such as cells or nuclei and a hundred millions of features. A study may involve hundreds of images obtained from a large cohort of subjects, and and a moderate-size healthcare operation can routinely generate thousands of whole slide images per day. For large scale interrelated analysis, there may be dozens of algorithms – with varying parameters – to generate many different result sets to be compared and consolidated. Thus, derived data from images of a single study is often in the scale of tens of terabytes, and petabytes of data is common when analytical pathology imaging is adopted in the clinical environment in the future. Such big data combined with complexity of spatial queries poses major challenges for developing effective solutions.

1.2.3 High Complexity of Geometric Computation

A typical spatial join such as spatial cross-matching first finds matching intersected polygon pairs and then does spatial measurement. A naive brute force approach by comparing all possible polygon pairs is extremely expensive and may take hours or days to finish even for a single image. This is mainly due to the complexity of common computational geometry algorithms [7] used for verifying intersections of polygon pairs, which often come with hundreds of points to represent each shape. While spatial access methods provide efficient matching of polygons based on their minimal boundary rectangles (MBR), computations on verification of intersections and the spatial measurements often dominate the cost of queries.

1.3 The Database Solution for Scenario 1

To support Scenario 1, an open source system called PAIS (Pathology Analytical Imaging Standards) was developed. [8, 21]. PAIS provides a comprehensive logical model for representing spatial objects and annotations associated with them. Geometric shapes such as polygons or multipolygons are used to represent the boundaries of segmented objects, such as tumor regions, blood vessels, and nuclei. PAIS employs a spatial DBMS based implementation for managing data. *Spatial tables* are used for representation of markup objects with spatially extended data types such as ST_POLYGON. *Feature and observation tables* are used to capture features and classifications. The SDBMS provides functions to support comparison of relationships across spatial objects, such as ST_intersects, ST_overlaps, ST_within, ST_contains, and ST_touches. It also provides numerous spatial measurement functions, including those to compute the area and centroid of a spatial object, to calculate the distance of two spatial objects, and to generate an intersected region. PAIS database provides powerful support of most queries required, and is extended with user-defined functions for additional queries or operations.

Parallel Spatial Database Architecture. To scale out to multiple nodes, parallel database implementation could be used through data partitioning with a shared-nothing parallel database architecture. For example, in work [37], load balancing and co-location aware partitioning algorithms are developed to generate partition keys, which are then used to distribute data evenly across partitions. We use a commercial DBMS with spatial extension for the implementation.

1.4 Research Questions for Scenario 2

PAIS provides a comprehensive data model and supports expressive powerful queries with spatial extended SQL. It can also be scaled out to multiple partitions for managing large volume of data. PAIS database, however, is not suitable for Scenario 2, where on the fly data processing is required and massive computation is needed. Although the database supports complex queries such as spatial joins, such queries are highly computationally intensive and could take hours for comparing two result sets from a single image on a database with a single partition. Scaling out such queries through a large scale parallel database infrastructure is possible, as demonstrated in work [37]. In the work, a five partition parallel spatial database delivers significant performance improvement. One bottleneck of database approach is data loading [31], and it takes minutes to load the result from a single image into the database. The parallel database approach is also highly expensive on software and hardware [31, 20, 32], and requires sophisticated maintenance and tuning. A recent quote from a commercial vendor for a small scale parallel database server costs nearly a million US dollars.

Our question is, could we develop a cost effective solution to support Scenario 2, without expensive software license, hardware cost and administration complexity, but could be easily scaled out to support spatial queries? If so, could the solution provide efficient queries as those provided by DBMS with indexing technologies, and expressive query languages such as SQL? This motivates us to develop *Hadoop-GIS*, a system that marries MapReduce (scalability, fault tolerance and low cost) and DBMS (indexing and declarative query language), as discussed next.

1.5 Our Approach: Hadoop-GIS

Hadoop-GIS is a high performance spatial query system to support complex queries for analytical pathology imaging. The approach is general and can be applied to similar "GIS" applications in other scientific domains. Our contributions include:

- *Efficient query processing and parallelization* – build standalone query application to facilitate query optimization, and partition data to support parallelization of queries. We develop a real-time spatial query engine (RESQUE) to dynamically build up indexes to support efficient spatial queries;
- *Scalable and cost effective solution* – take advantage of cost effective commodity clusters, and rely on MapReduce computing architecture to ease application development. We develop highly scalable MapReduce (Hadoop) based spatial query processing with data staging on HDFS, and support feature based queries through Hive running on MapReduce;
- *Declarative query language* – ease the complexity of query writing for biomedical researchers or developers. We develop generalized interfaces for spatial operations, and extend Hive's SQL like declarative query language HiveQL to express also spatial queries;
- *Integrated system.* We integrate spatial querying capabilities into Hive engine to provide a single unified system to support queries for analytical pathology imaging.

The paper is organized as follows. We first present the architecture overview of Hadoop-GIS in Section 2. The real-time spatial query engine is discussed in Section 3. MapReduce based spatial query processing is presented in Section 4, and Hive based feature query processing is discussed in Section 5. Section 6 discusses integrated queries and query processing through integration of spatial queries into Hive. Performance study is discussed in Section 7, followed by Related Work and Conclusion.

2. ARCHITECTURE OVERVIEW

2.1 Related Approach

A database approach *PAIS* has been developed to support Scenario 1 [8, 21, 36]. PAIS provides a comprehensive data model to manage algorithm results, human annotations and provenance. In PAIS model, spatial shapes are used to represent tissue regions, cellular or subcellular objects. The PAIS database takes a spatial DBMS to manage the spatial objects as geometric objects supported by SDBMS, and features are managed as structured tables. The database implementation provides dozens of spatial predicates such as *INTERSECTS*, and a comprehensive set of spatial measurement functions. Comprehensive queries including spatial queries can be supported directly in spatial extended SQL, and internal spatial indexing can facilitate efficient query support. To scale out to multiple nodes, a shared-nothing parallel database architecture is provided for data partitioning and parallel data access. Load balancing and co-location aware partitioning algorithm is used to generate partition keys, which are then used to distributed data during the loading process [37].

PAIS is developed for managing, querying and sharing results, to support Scenario 1 summarized in the introduction. PAIS provides a comprehensive data model to represent data and provenance, and supports expressive powerful queries with spatial extended SQL. It can also be scaled out to multiple partitions for managing large volume of data. PAIS database, however, is not suitable for Scenario 2, where on the fly data processing and heavy computation is required. For example, data loading takes about 10 minutes for an image with about half million spatial objects. Although the database supports complex queries such as spatial joins, such highly computationally intensive queries could take hours for comparing two result sets for a single image on a database with a single partition. Scaling out such queries through a large scale parallel database infrastructure is possible [37] but very expensive on software and hardware [31,

20, 32] cost and requires sophisticated tuning and maintenance. The objective of the work presented in this paper is to provide a scalable and cost effective approach to support expressive and high performance spatial queries.

2.2 Goals

The main goal of Hadoop-GIS is to develop a highly scalable, cost-effective, efficient and expressive integrated query processing system for data intensive GIS applications, such as analytical pathology imaging. With the rapid advancement of network technologies, and increasingly wide availability of low-cost and high-performance commodity computers and storage systems, large-scale distributed cluster systems can be conveniently and quickly built to support biomedical applications. MapReduce is a distributed computing programming framework with unique merits of automatic job parallelism and fault-tolerance, which provides an effective solution to the big data analysis challenge. As an open-source implementation of MapReduce, Hadoop has been widely used in practice. This motivates us to develop a MapReduce based solution to support complex queries, especially spatial queries on large volume of data. Meanwhile, biomedical researchers or users often prefer to use a convenient system with declarative query interfaces. It could be difficult or impossible for them to code MapReduce programs (implementing map and reduce functions) for queries. High-level declarative languages can greatly simplify the effort on developing applications in MapReduce without hand-coding programs. These systems include Pig Latin/Pig [29], SCOPE [18], and HiveQL/Hive [34]. Recently we developed a system *YSmart* [27], a correlation aware SQL to MapReduce translator for optimized queries, and have it integrated into Hive [1]. This inspires us to integrate spatial query capabilities into such declarative query language on top of MapReduce, as discussed next.

Query Cases. There are three major categories of queries: i) feature aggregation queries, for example, queries on finding mean feature vector for each image and correlations between all feature pairs. Such queries are commonly used for integrative biomedical studies. ii) Spatial queries, especially spatial join queries. Algorithm result comparison and consolidation will eventually involve spatial join queries to cross-match spatial boundaries between different result sets. iii) Integrated spatial and feature queries, for example, feature aggregation queries in certain regions. Queries of this type can normally be decomposed as two steps: first step for spatial object filtering with containment relationship queries, and second step on aggregation on filtered spatial objects. More complex queries are discussed in future work (Section 9).

2.3 Methods

A fundamental component we aim to provide is a standalone spatial query engine with such requirements: i) is generic enough to support a variety of spatial queries and can be extended; ii) can be easily parallelized on clusters with decoupled spatial query processing and (implicit) parallelization; and iii) leverage existing indexing and querying methods. Porting a spatial database engine for such purpose is not feasible, due to a different system architecture and tight integration with RDBMS engine, and complexity on maintenance and optimization. We develop a Real-time Spatial Query Engine (RESQUE) to support spatial query processing, as shown in the architecture in Figure 3. RESQUE builds indexes on the fly and uses an index based spatial join to support efficient spatial queries. Besides, RESQUE is fully optimized, supports data compression, and comes with very low overhead on data loading.

[1]https://issues.apache.org/jira/browse/HIVE-2206

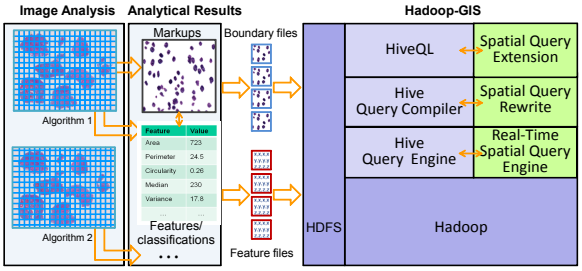


Figure 3: Architecture Overview of Hadoop-GIS

This makes RESQUE a highly efficient spatial query engine compared to traditional SDBMS engine. RESQUE is compiled as a shared library, without the need of installation on every cluster node, which is required for SDBMS engines.

Instead of using explicit spatial query parallelization as summarized in [15], we take an implicit parallelization approach by leveraging MapReduce. This will much simplify the development and management of query jobs on clusters. A whole slide image is partitioned into many small tiles for analysis (shown as grid cells in Figure 3), and the dimension of tiles is optimized for image analysis algorithms and determined ahead. Analytical results are derived from each tile as one boundary file and one feature file, often in the sizes of a few MBs each. Such tiles form the natural unit for parallelization on MapReduce. To run spatial queries on MapReduce, data is first staged onto HDFS. However, the small sizes of these files will generate huge redundancy as HDFS has large block size, thus lead to inefficient queries. The metadata used for HDFS is stored in the name node memory, thus a large number of small files will eat-up memory of namenode and degrades server performance as well. Instead, we merge all small files of a single whole slide image into a large file and then copied it to HDFS. We then develop MapReduce programs to support different types of spatial queries including spatial join queries by invoking RESQUE, with tasks managed by MapReduce.

To support feature queries with a declarative query language, we take advantage of Hive, which provides a SQL like language and supports major aggregation queries on top of MapReduce. Similar to the practice of DBMS to reduce I/O cost, data compression methods are also provided by Hive. For example, the RCFile method [24] takes a column partition based compression approach, and is integrated into Hive to provide efficient query support.

To provide an integrated query language and unified system on MapReduce, we extend Hive with spatial query language by providing spatial query rewriting and integrated queries with Hive query engine and spatial query engine (Figure 3). The spatial indexing aware query optimization will take advantage of RESQUE for efficient spatial query support in Hive.

3. REAL-TIME SPATIAL QUERY ENGINE

To support high performance spatial queries, the first requirement is a standalone spatial database engine with following capabilities: i) spatial relationship comparison, such as *intersects, touches, overlaps, contains, within, disjoint*, ii) spatial measurements, such as *intersection, union, convexHull, difference, distance, centroid, area*, etc; iii) spatial access methods for efficient query support; and iv) optimization for real-time processing environment. The engine should also be easily executed across multiple cluster nodes for parallelization. RESQUE is developed with such capabilities.

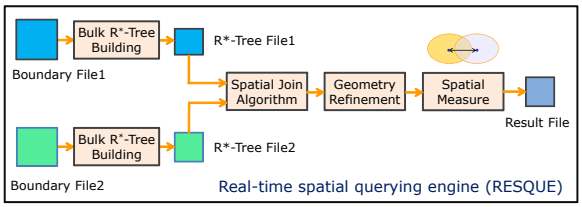


Figure 4: RESQUE Workflow for Spatial Join

3.1 Overview of RESQUE

One essential requirement for spatial queries here is on-the-fly queries, as new data and queries could be sent in and immediate query results need to be returned. For example, for algorithm sensitivity studies, parameters of an algorithm could be dynamically adjusted in a large parameter space to generate analysis results, and queries on these results would send immediate feedback for further parameter adjustments and image analysis. Previous parallel spatial query processing techniques [30, 40, 39] take partitioning based approach without creating indexes. We take an approach on combining partitioning with spatial indexing – RESQUE provides real-time spatial query processing that builds up spatial indices on the fly, and can run on partitions (discussed in Section 4.2). The approach is highly effective, and the index building overhead is only a small fraction of the total query time, thanks to the rapid development of CPU speed.

Here we take the example of a spatial join query to compare two result sets (more details discussed in Section 4.1) and demonstrate the workflow of queries (Figure 4). Boundary file 1 and file 2 contain the markup polygons from a tile generated from two algorithms, and are to be spatially joined for comparison. Bulk spatial index building is performed on each boundary file to generate index files – here we use R*-Trees [11]. The R*-Tree files contain minimal boundary rectangles (MBRs) in their interior nodes and polygons in their leaf nodes, and will be used for further query processing. The spatial join component performs MBR based spatial join filtering with the two R*-Trees, and refinement on the spatial join condition is further performed on the polygon pairs through geometric computations. The spatial measurement step is performed on intersected polygon pairs to calculate results required, such as centroid distance for each pair of intersecting markups. Other spatial join operators such as *overlaps* and *touches* can be run in a similar way. Spatial containment queries or point based queries are simpler as only one index file is needed.

Next we discuss the components in RESQUE.

3.2 Spatial Indexing and Optimization

In R*-tree, each non-leaf node of the tree stores pointers to its child nodes and corresponding MBRs, while each leaf node stores pointers to the actual spatial objects and corresponding MBRs. In our work, we modified and extended the SpatialIndex library [6] for building R*-Tree indexes. As data and indexes are read-only and no further update is needed, bulk-loading techniques [12] are used. To minimize the number of pages, the page utilization ratio is also set to 100%.

Index Compression. The polygons in the leaf nodes are encoded with additional information for retrieval. Each polygon record is represented as *(ID, N, Point1, Point2, ... , PointN)*, where *id* is the markup id, and *N* is the number of points. The markup polygons usually consist of hundreds or thousands of vertices, and two adjacent vertices usually have a distance of one pixel, either horizontally or vertically. For example, a polygon can be represented as *(10, 1000, 40961 8280, 40962 8280, 40962 8281, ... , 40961 8279)*, where markup id is 10, the number of points is 1000, and the other number pairs delimited by space represent (x y) coordinates. With a chain code representation, only the offset value between two adjacent points is represented, for example: *(ID, 1000, 40961 8280, 1 0, 0 1, ... , 1 0)*. The simple chain code compression approach saves space and reduces I/O significantly, as shown in our performance study in section 7.

3.3 Spatial Join and Filtering

Once the two R*-trees for two sets of markup polygons are built, spatial join is performed through an algorithm with a depth-first synchronized traversal of the two R*-trees. Starting from the two root nodes, the algorithm checks each pair of their child nodes. If the MBRs of a pair of nodes intersect, it then continues to join these two nodes and check each pair of their child nodes. The process is repeated until the leaf nodes are reached. The algorithm then checks each pair of the markup polygons indexed in these two leaf nodes to find all the pairs of markup polygons whose MBRs intersect. We build the join algorithm on top of the SpatialIndex library [6] and the code is in the process of being incorporated into SpatialIndex.

3.4 Spatial Refinement and Measurement

For each pair of markup polygons whose MBRs intersect, they are decoded from the representation, and geometry computation algorithm is used to check whether the two markup polygons actually intersect. If so, the spatial measurements are computed and returned. We rely on an open source library *Computational Geometry Algorithms Library (CGAL)* [7] for computing the refinement and measurements. Based on our experiments, spatial refinement based on geometric computation is computationally intensive, and dominates the query execution cost of RESQUE (88% of querying time).

3.5 Discussion

Generalized interfaces. To generalize the queries, we develop a set of spatial join operators and a set of spatial measurement functions, thus RESQUE is parameterized and could be easily called for executing different types of queries.

Granularity of indexing. The granularity of indexing could be at image level or tile level. Tile level indexing approach (many small R*-Tree indexes) not only preserves query performance compared to image level indexing approach (a big R*-Tree index) as shown in our performance study, but also enables parallelization and integration of spatial queries into MapReduce.

The indexing based spatial join and fully optimized operations, and the generalized interfaces make RESQUE a full-fledged highly efficient spatial query engine, which is much more efficient than spatial DBMS engines. Our performance shows that RESQUE is twice faster than PostGIS with GiST based indexing [4], and four times faster than SDBMS-X (a commercial spatial DBMS with a grid based indexing), not to mention the lightweight data loading time in RESQUE compared to other systems.

4. MAPREDUCE BASED SPATIAL QUERY PROCESSING

RESQUE provides a core query engine to support on the fly spatial queries. This enables us to develop high performance large scale spatial query processing based on MapReduce framework. In this section, we first present a typical spatial query example, and then discuss the workflow of our MapReduce programs for the query.

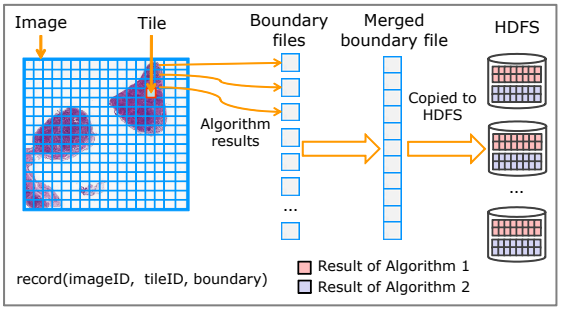


Figure 5: Data Staging of Spatial Data in MapReduce

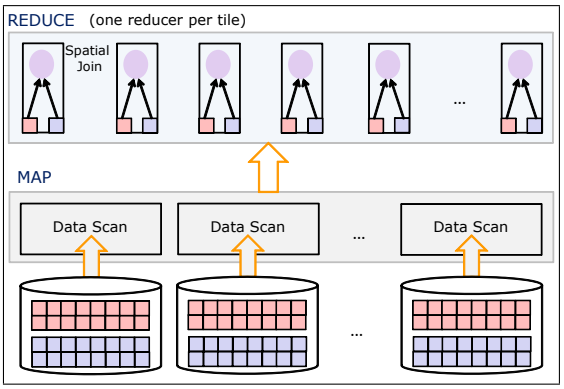


Figure 6: MapReduce Based Spatial Query Processing

4.1 Example Spatial Query: Spatial Join

RESQUE supports different types of queries, such as Point, Window, Containment and Spatial Join. Among them, spatial join is the mostly used and expensive queries for result validation and consolidation. We first show the semantics of this query in SQL, and then discuss how it can be mapped into a MapReduce application based on RESQUE. Figure 7 shows an example spatial join query in SQL to find all intersected markup pairs between two result sets generated from an image, and compute the intersection-to-union ratios and centroid distances of the pairs. The SQL syntax comes with spatial extensions such as spatial relationship operator *ST_INTERSECTS*, spatial object operators *ST_INTERSECTION*, *ST_UNION*, and spatial measurement functions *ST_CENTROID*, *ST_DISTANCE*, and *ST_AREA*.

```
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  markup_polygon ta, markup_polygon tb
7: WHERE
8    ta.algrithm_uid='A1' AND tb.algrithm_uid='A2' AND
9:   ta.pais_uid = 'IMG1' AND tb.pais_uid = 'IMG1' AND
10:   ta.tile_uid = tb.tile_uid AND
11:ST_INTERSECTS(ta.polygon, tb.polygon) = 1;
```

Figure 7: An Example Spatial Join Query in SQL with Spatial Extensions

In this example, table (*markup_polygon*) is a spatial table that represents markups. This table has three major columns, namely *pais_uid*, *tile_uid*, and *polygon*, respectively. In each record, *algorithm_uid* represents algorithm UID, *pais_uid* represents image UID, and *tile_uid* is the UID of a tile the polygon is contained. This is a self join of the same table *markup_polygon* by selecting polygons generated from the same image 'IMG1' and produced by different algorithms 'A1' or 'A2'. The join condition in line 10 implies that two polygons would not be compared unless they are in the same tile. The spatial join predicate in line 11 means that two qualified polygons must intersect with each other. In the SELECT clause of this query, we calculate intersection-to-union ratios and centroid distances of the polygon pairs with a few computational geometry functions. The query can be summarized as two parts: the regular part and the spatial part. The regular part contains no spatial operations but has only regular predicates and join. The spatial part contains the predicate of polygon intersection and spatial measurement functions.

Next we discuss how this example query can be implemented in the MapReduce framework.

4.2 Data Partitioning

One essential requirement for MapReduce applications is that data records can be partitioned based on certain keys. This matches the way how pathology images are processed, where original large images are partitioned into many small regions – tiles (Figure 3). Each tile is a rectangle, and often of the same size. For example, in our studies, we have tile size of 4096x4096 pixels. Those tiles contain no objects, for example, tiles on non-tissue regions, are discarded. A single whole slide image could be partitioned into around one hundred tiles, and the number of tiles varies based on the size and shape of specimens and image resolution. Such tile based partitioning not only enables parallel image analysis on clusters, but also provides the same partitioning unit for querying the analytical results. In our MapReduce programs, tiles are the unit for partitioning, and used as keys.

Image analysis algorithms generate two types of data files, spatial data files (or boundary files) and feature data files, and boundary files will be used for spatial queries. By default, each algorithm on one tile of a partitioned image will create one feature result file and one boundary result file. These files are small in sizes, for example, the average boundary file size is 3.2 MB in our example dataset. We use the following folder structure to organize original data files: *algorithm_uid/image_uid/filetype(feature or markup)/tile_result_filename*.

4.3 Data Staging

The first step for MapReduce is to stage data onto HDFS. Such small sized input files, however, are not suitable to be stored directly onto HDFS due to the nature of HDFS, which is optimized for large data blocks (default block size 64MB) for batch processing. Large number of small files leads to deteriorated performance for MapReduce due to following reasons. First, each file block consumes certain amount of main memory on the *namenode* and this directly compromises cluster scalability and disaster recoverability. Secondly, in the Map phase, the large number of blocks for small files leads to *large number of small map tasks* which has significant overhead, as shown in our experiment study.

Instead, we propose to merge all small tile based result files for each image as a single large file, and then stage the merged large files onto HDFS, as shown in Figure 5. Such file merging will lose the file name and folder structure information, which represents essential information for identifiers and grouping of data. To amend that, We add metadata from filenames and folder structures into the records. The records in boundary files have the following structure: *(algorithm_uid, pais_uid, tile_uid, markup_id, boundary)*. *markup_id* represents a sequential number of the boundary in the tile. Each boundary contains a set of (x,y) coordinates that form the polygon. This merging approach dramatically improves

Algorithm 1: Map Function

input: k_i, v_i
do projection on v_i and get the value of
$algorithm\_uid$, $pais\_uid$, $tile\_uid$, and $polygon$
if $algorithm\_uid$ == *'A1' and* $pais\_uid$ == *'IMG1'* **then**
    k_m = $tile\_uid$;
    v_m = ('ta', $polygon$);
    emit(k_m, v_m);
if $algorithm\_uid$ == *'A2' and* $pais\_uid$ == *'IMG1'* **then**
    k_m = $tile\_uid$;
    v_m = ('tb', $polygon$);
    emit(k_m, v_m);

Algorithm 2: Reduce Function

input: k_m, a list of values
initialize file fl for left side;
initialize file fr for right side;
dispatch each value to fl or fr;
//build R*-Tree indexes
tl = RESQUE.build_index(fl);
tr = RESQUE.build_index(fr);
//execute queries using indexes
$result$ = RESQUE.execute_query(tl, tr);
//final output
parse $result$ and output to HDFS;

the map performance, as shown in our performance studies (Section 7).

Another approach for handling large number of small files on HDFS is HAR [2]. However, there is an overhead on preparing data in HAR format before data can be staged. Our testing shows that while HAR achieves similar querying performance, HAR takes more than double time to get data loaded compared to file merging approach. Thus the file merging based approach outperforms HAR on overall performance.

4.4 MapReduce Program Structure

A MapReduce program to execute the spatial join query (Figure 7) will have similar structure to execute a regular relational join operation, but with all the spatial part executed by invoking RESQUE engine from the program.

According to the equal-join condition, the program uses the Standard Repartition Algorithm [13] to execute the query. Based on the MapReduce structure, the program has three main steps:

1. In the map phase, the input table is scanned, and the WHERE condition is evaluated on each record. Only those records that can satisfy the WHERE condition will proceed to the next step.

2. In the shuffle phase, all records with the same *tile_uid* would be shuffled to be the input of the same reduce function, since the join condition is based on *tile_uid*.

3. In the reduce phase, the join operation is finished by the execution of the reduce function. The spatial part is executed by invoking the RESQUE engine in the reduce function.

Algorithm 1 shows the workflow of the map function. Each record in the table is converted into the map function input key/value pair (k_i, v_i), where k_i is not used by the program and v_i is the record itself. Inside the map function, if the record can satisfy the select condition, then an intermediate (k_m,v_m) is generated. The key k_m is the value of *tile_uid* of this record, and the value v_m is the values of required columns of this record. There are two remarkable points. First, since (k_m,v_m) will participate a two-table join, a tag must be attached to v_m in order to indicate which table the record belongs to. Second, since the query is a self-join of the same table, we use a shared scan in the map function to execute the data filter operations on both instances of the same table. Therefore, a single map input key/value could generate 0, 1 or 2 intermediate key/value pairs, according to the SELECT condition and the values of the record.

The shuffle phase is controlled by Hadoop itself. Algorithm 2 demonstrates the workflow of the reduce function. According to the main structure of the program, the input key of the reduce function is the join key (*tile_uid*), and the input values of the reduce function are all records with the same *tile_uid*. In the reduce function, we first initialize two temporary files, then we dispatch records into corresponding files. After that, we invoke RESQUE engine to build R*-tree indexes and execute the query. The execution result data sets are stored in a temporary file. Finally we parse that file, and output the result to HDFS. Note that the function *RESQUE.execute_query* here performs multiple spatial functions together, including evaluation of WHERE condition, projection, and computation (e.g., *ST_intersection* and *ST_area*), which could be customized (Section 3).

We highlight two major advantages of this MapReduce program. First, it uses one shared scan to execute the operations on both instances of the same table for the self join, thus avoids redundant network or disk I/O cost from an additional table scan. Second, the program has a general interface with RESQUE, and all the spatial parts are pushed into RESQUE. Thus our program is not tied to a specific spatial operation, and general for a variety of spatial queries. For join algorithms, other join algorithms such as Improved Repartition Join [13] will not help on improving the query performance, as the query is mainly a spatial join, and not a regular join. In spatial joins, computational geometry functions are computational intensive. According to our performance testing results, the RESQUE part dominates the execution time in the reduce function.

5. FEATURE QUERIES WITH HIVE

Queries on spatially derived image features such as summary statistics and correlation analysis are commonly used for integrative biomedical studies. Instead of writing our own MapReduce programs to perform such queries, we rely on MapReduce based system Hive, which fits well for the queries.

5.1 Hive Overview

Hive [35] is an open source MapReduce based query system that provides a declarative query language for users. By providing a virtual table like view of data, SQL like query language HiveQL, and automatic query translation, Hive achieves scalability while it greatly simplifies the effort on developing applications in MapReduce without hand-coding programs. HiveQL supports a subset of standard ANSI SQL statements which most data analysts and scientists are familiar with. In addition to common SQL functions including aggregation functions, Hive provides high extensibility through User Defined Functions (UDF) and User Defined Aggregation Functions (UDAF) implemented in Java, or custom mapreduce scripts written in any language using a simple streaming interface. By using Hive, we are able to create "tables", load data and specify feature queries in HiveQL. For example, Figure 8 shows an example aggregation query in HiveQL.

5.2 Hive Storage

Hive provides alternative data storage formats to be used for different applications to optimize query performance. The way of how

```
SELECT AVG(AREA), AVG(PERIMETER), ...
  STDDEV(AREA),  STDDEV(INTENSITY), ...
  CORR(AREA, INTENSITY), CORR(AREA, ENERGY),...
FROM IMG_x GROUP BY Algorithm_Id;
```

Figure 8: An Example Query in HiveQL

the data is being stored affects not only query performance, but also data loading performance. Example formats supported by Hive include: *Text*, in which each line in the text file is a record; *bzip2*, which compresses data into blocks of a few hundred KBs; and *RCFile*: Record Columnar File (RCFile), a Hive specific column oriented storage structure [24]. RCFile gives considerable performance improvement especially for queries that do not access all the columns of the table. To exploit the best approach for our use case, we benchmark different formats for query performance (Section 7). As shown in the performance study in Section 7, RCFile performs best for feature aggregation queries, and seems an ideal solution.

6. INTEGRATED MAPREDUCE BASED GIS QUERYING SYSTEM

Our MapReduce programs for spatial queries were initially developed as a set of MapReduce programs written in Java, which call RESQUE for executing spatial query operations. Such approaches require adjustment of codes or addition of new codes whenever there are new data or queries. This poses a high barrier for end users – biomedical researchers. They normally have limited knowledge on programming MapReduce based queries, and prefer declarative query languages such as SQL.

Providing a declarative query language and building an integrated system are among the major goals of Hadoop-GIS. As Hive provides declarative query language for feature queries and standard interfaces, it is natural to extend Hive with spatial queries to provide an integrated platform. Our previous work on patching Hive with query optimization [27] motivates us to extend Hive to support spatial queries.

We design two solutions for spatial extension to Hive. The first solution, called *Pure UDF Solution (PUS)*, aims to implement spatial functions using Hive's UDF mechanism without modifying Hive itself. The second solution, called *Spatial Indexing Aware Solution (SIAS)*, implements special R*-Tree aware operators in Hive and makes corresponding extensions on Hive's query optimizer.

6.1 Pure UDF Solution (PUS)

In this approach, all spatial functions used in our queries are implemented as Hive UDFs. The implementation is based on the CGAL library for spatial predicates and spatial measurements. Essentially, those UDFs are wrappers of corresponding CGAL functions. Here we take an example on the UDF *HU_intersects* that maps to the function *ST_intersects*. To determine whether two polygons intersect with each other, *HU_intersects* invokes the CGAL function *intersects* to calculate the intersected regions and their area.

Fig. 9(a) shows an example of the query plan tree for the query in Fig. 7. Here we focus on the operators in the reduce phase. The first operator *JoinOperator* is used to execute the regular join predicate in the query. Then the joined results are sent to the second operator *FilterOperator* to execute the spatial join predicate *ST_Intersects(ta.polygon, tb.polygon)*. FilterOperator will invoke the UDF for the *intersects* function. The third operator *SelectOperator* will further invoke corresponding UDFs to execute spatial measurement functions (e.g., *ST_area()*). The last operator *FileOutputOperator* will output the final results.

One major advantage of this approach is its simplicity: there is no need to modify Hive codes. As shown in the figure, all operators are existing operators in Hive. However, this naive approach has inferior performance, as Hive would not consider the spatial join function as a type of spatial join condition. Rather, it evaluates the *HU_intersects* UDF on all possible pairs of polygons returned by the equal-join operation. That means, the spatial join is executed by applying a predicate after a Cartesian product in the JoinOperator. This leads to exponential increase of computational complexity, as demonstrated in the brute force experiment in Section 7. Thus the approach is not useful in practice.

6.2 Spatial Indexing Aware Solution (SIAS)

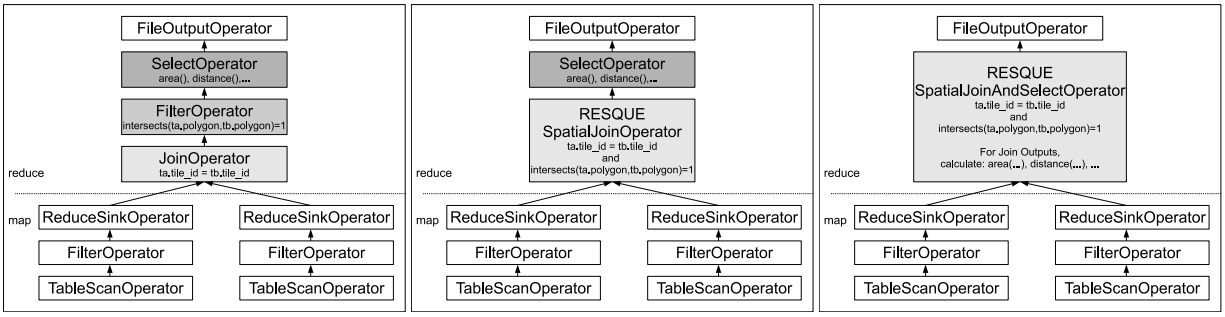
The Spatial Indexing Aware Solution will consider spatial indexing, and integrate RESQUE spatial query engine into Hive query processing. This requires modification of Hive, but it carries the performance advantage of Hadoop-based spatial query processing with RESQUE. We take this approach for our implementation.

In SIAS, the spatial measurement functions are implemented as Hive UDFs, same as in PUS. For example, *ST_area* is implemented as *HU_area* via corresponding CGAL functions. Spatial join functions such as *ST_intersects*, *ST_touches*, etc, are not implemented as simple UDFs. Rather, a special operator *SpatialJoinOperator* is implemented and integrated into Hive's query optimizer. The operator works in the reduce phase, and calls corresponding RESQUE program to execute spatial join on input data sets, as shown in the example in Algorithm 2. Since Hive uses a rule-based optimizer, a new special rule is added to Hive query optimizer to insert the SpatialJoinOperator in the query plan trees. Fig. 9(b) shows the query plan tree using the SIAS approach. Compared to PUS (Fig. 9(a)), in SIAS, the SpatialJoinOperator executes both the functionalities of JoinOperator and FilterOperator, and the operator is supported by RESQUE engine with R*-tree indexing. This provides significant performance advantage.

Merging Based Spatial Indexing Aware Solution. A further optimization based on SIAS is that we can push down spatial measurement UDFs in Hive's SELECT operator into the *SpatialJoinOperator*. For example, in the MapReduce program in Section 4, RESQUE not only executes *ST_intersects* (spatial relationship operator), but also executes *ST_area* (spatial measurement operator). The benefit of this optimization is that all spatial operations can be fully executed by RESQUE in a combined single step, and the communication overhead between RESQUE and Hive operators is minimized. Fig. 9(c) shows the query plan tree with this optimization: the SelectOperator in Fig. 9(b) is pushed down to the RESQUE engine.

7. PERFORMANCE STUDY

Experiment setup. We have two systems on performance study, a dedicated small cluster, and a real world medium sized cluster with shared resources. We use the small cluster for benchmarking RESQUE, comparing RESQUE with SDBMS, testing loading performance, and performing small scale MapReduce studies. We use the medium cluster as a scalability test for real world environment. 1) The small scale cluster comes with 10 nodes and 192 cores: 4 nodes with 24 cores (AMD 6172 2.1GHz), 2.7TB hard drive 7200rpm, and 128GB memory per node; and 6 nodes with 16 cores (AMD 6172 2.0GHz), 7TB hard drive 7200rpm, and 128GB memory per node. 1 Gb interconnecting network is used. The OS is CentOS 5.6 (64 bit). The version of Hadoop is 0.20.2-cdh3u2, and the version of Hive is 0.7.1. 2) The medium scale cluster



(a) PUS (b) SIAS (c) Improved SIAS

Figure 9: Extend Hive with Spatial Queries: (a) Pure UDF Solution (PUS), (b) Spatial Index Aware Solution (SIAS), and (c) Merging Based SIAS

named "Hotel" on futuregrid [2] comes with following configurations: 40 nodes with 8 cores per node, a total of 320 cores. It is installed with a distributed file system based on IBM GPFS (file split size 128MB). The OS is Scientific Linux 5. Hadoop version is 0.20.203.0, and Hive version is 0.7.1.

The version of CGAL library used is V3.8, and the version of SpatialIndex library is 1.6.0. The version of PostGIS is 1.5.2. We use dataset of whole slide images for brain tumor study provided by Emory University Hospital, with two results computed from two methods. We have dataset sizes at 1X(18 images, 44GB), 3X(54 images, 132GB), 5X(90 images, 220GB), 10X(180 images, 440GB), and 30X(540 images, 1,320GB) for different testings. The average number of nuclei per image is 0.5 million, and each nucleus has 74 features generated.

7.1 RESQUE

Query Performance of RESQUE. To test the performance of RESQUE itself, we run it on a single node as a single thread application. We run the spatial join query (Figure 7) with RESQUE, as it is a common used expensive query type. We first test the *effect of spatial indexing*, by taking a single tile with two result sets (5506 markups vs 5609 markups) (Figure 10). A *brute-force approach* compares all possible pairs of boundaries using a computational geometry function without any index, and takes 673 minutes. Such slow performance is due to polynomial complexity on pair-wise comparisons and high complexity on geometric computation function. An *optimized brute-force approach* will first compare all possible pairs of boundaries using MBRs, and then only filter pairs with MBR intersections using the computational geometry function. This approach takes 4 minutes 41 seconds, a big saving with minimized geometric computations. Using RESQUE with indexing based spatial join, the number of computations is significantly reduced, and it only takes 9.2 seconds. With SDBMS based approach, we load the data into spatial tables in PostGIS and SDBMS X respectively and create corresponding spatial indexes, and then run the query as spatial extended SQL queries. It costs 12.8 seconds and 38.9 seconds for PostGIS and SDBMS X respectively. This demonstrates high efficiency of RESQUE.

Effect of Data Loading. Besides query performance, RESQUE enjoys the light loading cost compared to SDBMS approach (Section 4.3). We run three steps to get the overall response time (data loading, indexing and querying) on RESQUE on a single slot MapReduce with HDFS, PostGIS and SDBMS X with a single partition. The data used for the testing is two results from a single image (106 tiles, 528,058 and 551,920 markups respectively). Figure 11(a)

[2]http://www.futuregrid.org

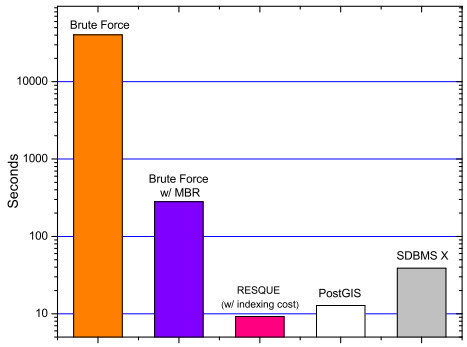


Figure 10: Comparison of Query Performance (Single Tile)

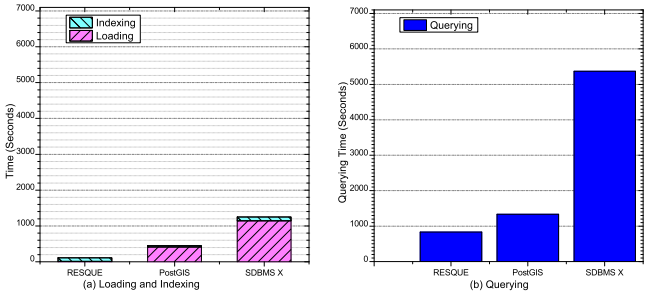


Figure 11: Comparison of Performance (Single Image)

shows the loading and index building performance of the three systems, and Figure 11(b) shows the query performance. RESQUE not only dominates on the query efficiency, but also has minimal loading and index building time. This makes RESQUE a powerful candidate for building a fast response query system.

Scalability on Indexing Size. With RESQUE, there are two approaches for indexing: image level indexing, and tile level indexing. We take a whole slide image with two result sets, same as previous testing. In the first approach, we build a single big R*-Tree for all tiles of the image, and the method takes 15 minutes 32 seconds to build indexes and run the query with RESQUE. For the second approach, we build one R*-Tree for each tile and aggregate query results from all tiles. It takes 16 minutes 5 seconds, with slight time increase (3.5%) due to more indexing time. This clearly demonstrates that our partitioned based approach with small

R*-Trees not only preserves performance, but also enables parallelization. The breakdown of RESQUE execution time (multiple small R*-Tree approach) is 2 minutes 16 seconds for index building (16%), and 13 minutes 49 seconds for spatial join (84%).

Effect of Data Compression. For data compression in R*-Tree with chain code based coordinate representation, the storage is reduced by 42%, a significant reduction of I/O during query processing. When the utilization ratio is varied from 70% to 100% in R*-Tree configuration, it saves space by another 2%.

7.2 Performance of Hadoop-GIS

We take the example in Figure 7 to demonstrate the performance of executing spatial queries in Hadoop-GIS. We first present data staging performance and then present query execution performance.

7.2.1 Data Staging

Hadoop-GIS is designed for fast response, and the performance of data staging affects the overall performance. One major factor that affects spatial query performance is the way to store the input data, which consists of many small files with partitioning based image analysis results – with average size of 3.2 MB. HDFS is designed for storing large files, and not suitable for handling small files. Our method on merging small files into large ones and tracking file/tile information as metadata inside the merged files boosts the performance for spatial query processing. The average size of merged files is 361 MB per image.

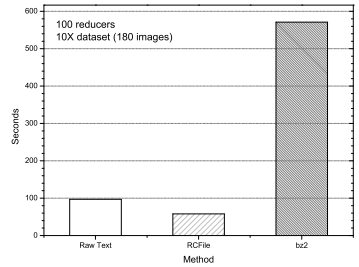
Effect of File Sizes. To test the effect between small files and merged files, we use our spatial join program and run it as Map only MapReduce job on one set of data (18 images). By storing small result files of tiles directly onto HDFS, it takes 199 seconds in the map phase. After merging small files into two large files (one file per result per image), the map phase execution time is reduced to only 56 seconds. The slow performance of small file approach is due to the much overhead from large number of blocks and the large number of map tasks invoked: 4228 tasks versus 125 tasks in file merging approach.

File Merging versus HAR. While HAR provides an approach for handling small files, it takes extra time for loading data. For example, with 18 images, HAR takes 440 seconds, and merging and loading small files only takes 202 seconds, more than 50% reduction of time.

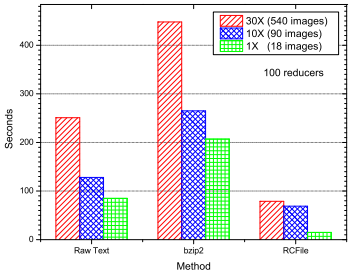
7.2.2 Spatial Queries

We perform two sets of testing for MapReduce based spatial queries, on the small cluster with dedicated resource, and on the medium cluster in a real world environment. Note that the reduce phase dominates the cost as map phase is performing simple record grouping based on keys. Figure 12(a) shows the results on the small cluster with different data sizes: 1X, 3X, 5X, and 10X (18 images) data sets, with varying number of reducers. We can see a continuous drop of time when the number of reducers increases, and the time reaches a steady number when all cores on the cluster are fully utilized. It achieves a nearly linear speed-up, e.g., time is reduced to about half when the number of reducers is increased from 50 to 100. The average querying time per image is 15 seconds for the 1X dataset with all cores, comparing with 22 minutes 12 seconds in a single partition postGIS, and 89 minutes 30 seconds on a single partition SDBMS X. Figure 12(b) demonstrates the loading time and querying time versus data sizes on the small cluster with 180 reducers, for dataset 1X, 3X, 5X and 10X. It shows a nearly linear increase of time versus data sizes.

The medium sized cluster is a shared cluster on futuregrid with many other jobs running. Thus the time is affected by the system load. We test with four datasets: 1X, 5X, 10X, and 30X data sets with varying number of reducers, as shown in Figure 12(c). We can see that the curves are not as smooth as those in the dedicated small cluster due to random background job traffic. We still see a continuous dropping of time, although it is not linear due to the background job traffic.



(a) Small Cluster



(b) Medium Cluster

Figure 13: Query Performance of Feature Aggregation

7.2.3 Performance of Feature Queries

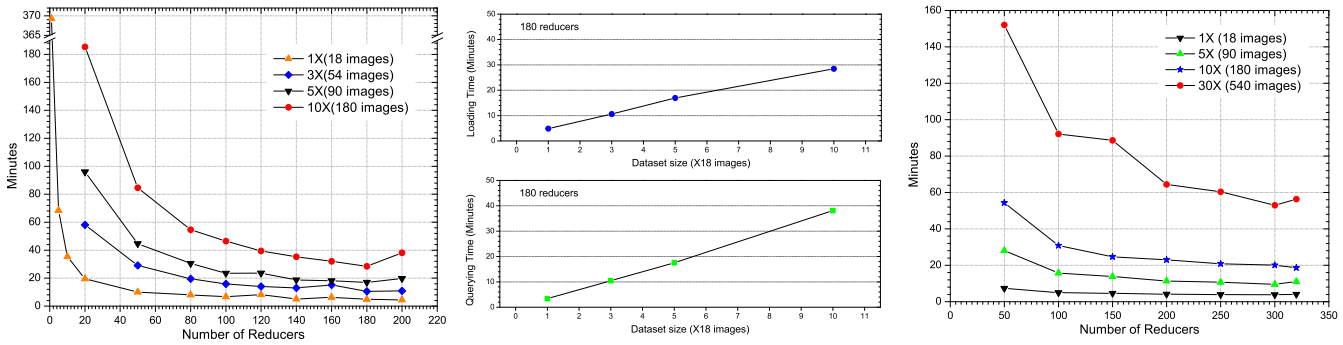
We test the performance of executing feature aggregation queries in Hive. Figure 13(a) shows the performance on the small cluster for 10X dataset with 100 reducers. It takes 58 seconds to run the query with RCFile method, and 97 seconds to run with raw text method. Clearly, RCFile method is the query performance winner, and is ideal when the queries are run multiple times.

Figure 13(b) shows the query performance of Hive on the medium cluster for text, bzip2 and RCFile with different datasets. Hive provides efficient aggregation queries – it takes only a few minutes even for the 30X dataset (with about 20 billion features).

8. RELATED WORK

Scientific databases often come with spatial aspects [10], for example, Large Synoptic Survey Telescope (LSST) generates huge amount of spatially oriented sky image data. Human Atlas projects include [1] and others. Digital microscopy is an emerging technology which has become increasingly important to support biomedical research and clinical diagnosis. There are several projects that target creation and management of microscopy image databases and processing of microscopy images. The Virtual Microscope system [17] developed by our group provides support for storage, retrieval, and processing of very large microscopy images on high-performance systems. The Open Microscopy Environment project [23] develops a database-driven system for managing analysis of biological images, which is not optimized for large scale pathology images.

Pig/MapReduce based approach has been studied in [28] for structural queries for astronomy simulation analysis tasks and compared with IDL and DBMS approaches. In [16], an approach is proposed



(a) Querying Performance on the Small Cluster (b) Loading and Querying Performance on the Small Cluster (c) Querying Performance on the Medium Cluster

Figure 12: Hadoop-GIS Spatial Join Performance

on bulk-construction of R-Trees and aerial image quality computation through MapReduce. In [39], a spatial join algorithm on MapReduce is proposed for skewed spatial data, without using spatial indexes. The approach first produces tiles with close to uniform distributions, then uses a strip based plane sweeping algorithm by further partitioning a tile into multiple strips. Joins are performed in memory, with a duplication avoidance technique to remove duplicates across tiles. In Hadoop-GIS, tiling is produced at image analysis step, and it is a common practice for pathology imaging to discard duplicated objects at tile boundaries, as the final analysis result is a statistical aggregation. We take a hybrid approach on combining partitioning with indexes, and build spatial indexes on the fly, as the index building overhead is significantly reduced to a small faction of the total query time due to the rapid development of CPU speed. Our approach is not limited to memory size, and provides high efficiency with R*-Tree based join algorithm [14]. Our approach relies on implicit parallelization through MapReduce.

The Sloan Digital Sky Survey project (SDSS) [5] created a high resolution multi-wavelength map of the Northern Sky with 2.5 trillion pixels of imaging, and takes a large scale parallel database approach. SDSS provides a high precision GIS system for astronomy, implemented as a set of UDFs. The database runs on GrayWulf architecture [33], with waived license fee from Microsoft.

Partitioning based approach for parallelizing spatial joins is also discussed in [30, 40] where no indexing is used. An R-Tree based spatial join is proposed in [15] with a combined shared virtual memory and shared nothing architecture.

Comparisons of MapReduce and parallel databases are discussed in [31, 20, 32]. Tight integration of DBMS and MapReduce is discussed in [9, 38]. MapReduce systems with high-level declarative languages include Pig Latin/Pig [29, 22], SCOPE [18], and HiveQL/Hive [34]. YSmart provides an optimized SQL to MapReduce job translation and is recently patched to Hive. Hadoop-GIS takes an approach that marries DBMS's spatial indexing and declarative query language into MapReduce.

9. DISCUSSION & FUTURE WORK

Our experiment results demonstrate that Hadoop-GIS provides a scalable and effective solution for querying large scale spatial datasets. Ongoing work includes generalizing existing framework to support more complex spatial query and spatial analysis use cases and utilizing GPU to accelerate spatial queries.

9.1 Complex Spatial Queries and Spatial Data Analysis

Spatial Proximity between Micro-anatomic Objects. Micro-anatomic objects with spatial proximity often form groups of cells that are close in both physical space and gene expression space. For example, a shortest distance query will provide summary statistics of the proximity of stem cells to these regions of interest: for each stem cell, find hypoxic and angiogenesis regions that are closest to the cell and compute the mean distance and standard deviation for all stem cells in an image. This query will involve millions of cells for a single image. Another example query which involves spatial proximity is to identify nearest blood vessels for each cell and return local density measurement of blood vessels around a cell.

Global Spatial Pattern Discovery in Images. The tumor growth comes with necrosis and vascular proliferation which often forms spatial patterns during different stage of tumor growth. For example, glioblastoma, the most common brain tumor, often appears as ring-enhancing lesions where the rings have much higher concentration of cells than adjacent cells. By analyzing the spatial distribution patterns of cells or nuclei, it is possible to automate the identification of tumor subtypes and their characteristics.

Spatial Clustering. Spatial Clustering is a process of grouping spatial objects into clusters so that objects in the same cluster have higher similarity to one another. Such a grouping can be used to characterize tissue regions and region based features can be further applied to correlate with genetic information and disease outcome.

Preliminary work has been done on supporting nearest neighbor queries and spatial pattern discovery (high density regions) by extending the RESQUE query engine with additional query processing methods with extended access methods. Partitioning boundaries are also considered in some queries such as nearest neighbor queries.

9.2 GPU Accelerated Spatial Queries

Heavy geometric computation is among the major cost of many spatial queries. GPU has emerged as a cost-effective and powerful computing device for massively data-parallel general purpose computations. We are exploiting massive data parallelism by developing GPU aware parallel spatial computation algorithms such as spatial intersection and execute them on GPUs. Preliminary work has demonstrated promising results on reducing the bottleneck of spatial queries, which will be reported separately.

10. CONCLUSION

The big data from medical imaging "GIS" – the vast amount spatially derived information generated from pathology image analysis – shares similar requirements for high performance and scal-

ability with enterprise data, but in a unique way. Experimental oriented scientific data demands quick query response for discovery and evaluation of preliminary results, and is often computational intensive. The unique users – biomedical researchers – prefer declarative query interfaces for high usability. We study the feasibility of applying and extending the software stack (Hadoop/Hive) from the domain of enterprise big data analysis to the unique problems of scientific data analysis, and develop a system Hadoop-GIS to bridge the gap and meet the unique requirements for analytical pathology imaging. Hadoop-GIS provides an efficient and generic spatial query engine RESQUE, one core component to support real-time based spatial queries. With a combined partitioning and spatial indexing based approach, and implicit parallelization, complex spatial queries are mapped into MapReduce based applications. The hybrid approach on combining a full-fledged spatial query engine with MapReduce enables cost effective, efficient and scalable queries on commodity clusters. By integrating spatial querying capabilities into Hive, we deliver a single unified system to support both spatial queries and feature queries in a declarative language. Hadoop-GIS is deployed at Emory University to query results from 14,000 whole slide images for about 100TB of data, and is built as an open source software for medical imaging community. Our work is general and could potentially be applied to support similar "GIS" oriented applications.

11. REFERENCES

[1] The allen reference atlas. http://www.brain-map.org; http://mouse.brain-map.org/api/.

[2] Hadoop archives. http://hadoop.apache.org/common/docs/current/hadoop_archives.html.

[3] Large synoptic survey telescope. http://lsst.org/lsst/overview.

[4] Postgis. http://postgis.refractions.net.

[5] The sloan digital sky survey project (sdss). http://www.sdss.org.

[6] Spatial index library. http://libspatialindex.github.com.

[7] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[8] Pathology analytical imaging standards. https://web.cci.emory.edu/confluence /display/PAIS, 2011.

[9] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proc. VLDB Endow.*, 2:922–933, August 2009.

[10] A. Ailamaki, V. Kantere, and D. Dash. Managing scientific data. *Commun. ACM*, 53, June 2010.

[11] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, 1990.

[12] J. V. d. Bercken and B. Seeger. An evaluation of generic bulk loading techniques. In *VLDB*, pages 461–470, 2001.

[13] S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. In *SIGMOD*, 2010.

[14] T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD*, pages 237–246, 1993.

[15] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel processing of spatial joins using r-trees. In *ICDE*, 1996.

[16] A. Cary, Z. Sun, V. Hristidis, and N. Rishe. Experiences on processing spatial data with mapreduce. In *SSDBM'2009*, pages 302–319, 2009.

[17] Ü. V. Ç, M. D. Beynon, C. Chang, T. M. Kurç, A. Sussman, and J. H. Saltz. The virtual microscope. *IEEE Transactions on Information Technology in Biomedicine*, 7(4):230–248, 2003.

[18] R. Chaiken, B. Jenkins, P. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and efficient parallel processing of massive data sets. *PVLDB*, 1(2):1265–1276, 2008.

[19] L. A. D. Cooper, J. Kong, D. A. Gutman, F. Wang, J. Gao, C. Appin, S. Cholleti, T. Pan, A. Sharma, L. Scarpace, T. Mikkelsen, T. Kurc, C. S. Moreno, D. J. Brat, and J. H. Saltz. Integrated morphologic analysis for the identification and characterization of disease subtypes. *J Am Med Inform Assoc.*, Jan. 2012.

[20] J. Dean and S. Ghemawat. Mapreduce: a flexible data processing tool. *Commun. ACM*, 53(1):72–77, 2010.

[21] D. J. Foran, L. Yang, W. Chen, J. Hu, L. A. Goodell, M. Reiss, F. Wang, T. M. Kurç, T. Pan, A. Sharma, and J. H. Saltz. Imageminer: a software system for comparative analysis of tissue microarrays using content-based image retrieval, high-performance computing, and grid technology. *JAMIA*, 18(4):403–415, 2011.

[22] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a highlevel dataflow system on top of MapReduce: The Pig experience. *PVLDB*, 2(2):1414–1425, 2009.

[23] I. Goldberg and C. A. et. al. The open microscopy environment (ome) data model and xml file: Open tools for informatics and quantitative analysis in biological imaging. *Genome Biol.*, 6(R47), 2005.

[24] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu. Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems. In *ICDE*, pages 1199–1208, 2011.

[25] L. C. J. Kong, C. Moreno, F. Wang, T. Kurc, J. Saltz, and D. Brat. In silico analysis of nuclei in glioblastoma using large-scale microscopy images improves prediction of treatment response. In *EMBC*, 2011.

[26] J. Kong, L. Cooper, F. Wang, C. Chisolm, C. Moreno, T. Kurc, P. Widener, D. Brat, and J. Saltz. A comprehensive framework for classification of nuclei in digital microscopy imaging: An application to diffuse gliomas. In *ISBI*, 2011.

[27] R. Lee, T. Luo, Y. Huai, F. Wang, Y. He, and X. Zhang. Ysmart: Yet another sql-to-mapreduce translator. In *ICDCS*, 2011.

[28] S. Loebman, D. Nunley, Y.-C. Kwon, B. Howe, M. Balazinska, and J. Gardner. Analyzing massive astrophysical datasets: Can pig/hadoop or a relational dbms help? 2009.

[29] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, 2008.

[30] J. M. Patel and D. J. DeWitt. Partition based spatial-merge join. In *SIGMOD*, pages 259–270, 1996.

[31] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD*, pages 165–178, 2009.

[32] M. Stonebraker, D. J. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.

[33] A. S. Szalay, G. Bell, J. vandenBerg, A. Wonders, R. C. Burns, D. Fay, J. Heasley, T. Hey, M. A. Nieto-Santisteban, A. Thakar, C. v. Ingen, and R. Wilton. Graywulf: Scalable clustered architecture for data intensive computing. In *HICSS*, pages 1–10, 2009.

[34] 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. *PVLDB*, 2(2):1626–1629, 2009.

[35] 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.

[36] F. Wang, J. Kong, L. Cooper, T. Pan, K. Tahsin, W. Chen, A. Sharma, C. Niedermayr, T. W. Oh, D. Brat, A. B. Farris, D. Foran, and J. Saltz. A data model and database for high-resolution pathology analytical image informatics. *Journal of Pathology Informatics*, 2(1):32, 2011.

[37] F. Wang, J. Kong, J. Gao, C. Vergara-Niedermayr, D. Alder, L. Cooper, W. Chen, T. Kurc, and J. Saltz. High performance analytical pathology imaging database for algorithm evaluation. In *MICCAI/MICCAI-DCI*, 2011.

[38] Y. Xu, P. Kostamaa, and L. Gao. Integrating hadoop and parallel dbms. In *SIGMOD*, pages 969–974, 2010.

[39] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. Sjmr: Parallelizing spatial join with mapreduce on clusters. In *CLUSTER*, 2009.

[40] X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. *Geoinformatica*, 2:175–204, June 1998.