SpatialHadoop: Towards Flexible and Scalable Spatial Processing using MapReduce

Ahmed Eldawy
Expected Graduation: December 2015
Supervised by: Mohamed F. Mokbel
Computer Science and Engineering Department
University of Minnesota Minneapolis, MN, USA
{eldawy,mokbel}@cs.umn.edu

ABSTRACT

With the huge volumes of spatial data coming from different sources, there is an increasing demand to exploit the efficiency of Hadoop MapReduce framework in spatial data processing. However, Hadoop falls short in supporting spatial data efficiently as the core is unaware of spatial data properties. This paper describes SpatialHadoop; a full-fledged MapReduce framework with native support for spatial data. SpatialHadoop is a comprehensive extension to Hadoop that injects spatial data awareness in the main layers of Hadoop. It ships with a simple high level language with spatial datatypes and operations. The core contains traditional spatial indexes, Grid, R-tree and R+-tree, which are adapted to the MapReduce environment. SpatialHadoop is already equipped with a dozen of operations, including standard operations, computational geometry and data mining operations. SpatialHadoop is already used as a main component in three live systems MNTG, TAREEG and SHAHED. For efficient processing of spatio-temporal data, Spatio-temporal Hadoop is proposed as an extension to SpatialHadoop.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database ApplicationsSpatial Databases and GIS

Keywords

Hadoop; MapReduce; Spatial; Indexing

1. INTRODUCTION

The recent explosion in the amounts of spatial data urged researchers and practitioners worldwide to take advantage of the MapReduce environment in supporting large-scale spatial data. Most notably, in industry, ESRI has released a suite of GIS tools on Hadoop [1] that work with their flagship ArcGIS product. Meanwhile, in academia, three system prototypes were proposed: (1) Parallel-Secondo [12] as a parallel spatial DBMS that uses Hadoop as a distributed task scheduler, (2) $\mathcal{MD}$-HBase [14] extends HBase [2], a non-relational database runs on top of Hadoop, to support multidimensional indexes, and (3) Hadoop-GIS [6] extends Hive [16], a data warehouse infrastructure built on top of Hadoop with a uniform grid index for range queries and self-join.

A main drawback in the systems discussed above is the lack of integration with the core of Hadoop. For example, HadoopGIS is built on Hive, a data warehousing system for Hadoop, rendering it unuseful for traditional MapReduce programs running directly in Hadoop. On the other hand, SpatialHadoop is *built-in* Hadoop which pushes spatial constructs inside the core of Hadoop making it more efficient with query processing. More importantly, SpatialHadoop introduces standard spatial indexes and MapReduce components that allow researchers and developers to implement new spatial operations efficiently in the system. This is in contrast to Hadoop-GIS and other systems that cannot support such kind of flexibility, and hence they are limited to the functions they ship with. Finally, indexing options in previous systems are limited, e.g., uniform grid index in HadoopGIS, while SpatialHadoop provides more options including Grid File, R-tree and R+-tree, which makes it more useful with skewed data.

SpatialHadoop is available as open source [3] and has been already downloaded more than 75,000 times. SpatialHadoop is accessible through a high level language, named Pigeon [11], which extends Pig Latin with OGC-standard [4] data types and operations. In the core of SpatialHadoop, Grid File, R-tree and R+-tree indexes are adapted to the Hadoop file system (HDFS) by building them as a global index which partition data across nodes and multiple local indexes to organize records inside each node. The new indexes are made accessible to MapReduce programs by introducing two new components, SpatialFileSplitter and SpatialRecordReader which are used to access the global and local indexes, respectively. The new design of spatial indexes allows a myriad spatial operations to be implemented efficiently in SpatialHadoop. We show how to implement three basic operations, range query, kNN and spatial join [10]. In addition, we build CG_Hadoop [9], a suite of computational geometry operations, in SpatialHadoop which makes use of spatial indexes to provide orders of magnitude speedup. SpatialHadoop is also used in three live systems, MNTG [13] a web service for generating traffic data, TAREEG [7], a web-based extraction tool for OpenStreetMap data, and SHAHED, a system for analyzing satellite data from NASA. In order to support spatio-temporal data processing, we propose Spatio-temporal Hadoop as an extension that takes the time dimension into account.

This work is supported in part by the National Science Foundation, USA, under Grants IIS-0952977 and IIS-1218168.

SIGMOD'14 PhD Symposium, June 22, 2014, Snowbird, UT, USA

http://dx.doi.org/10.1145/2602622.2602625.

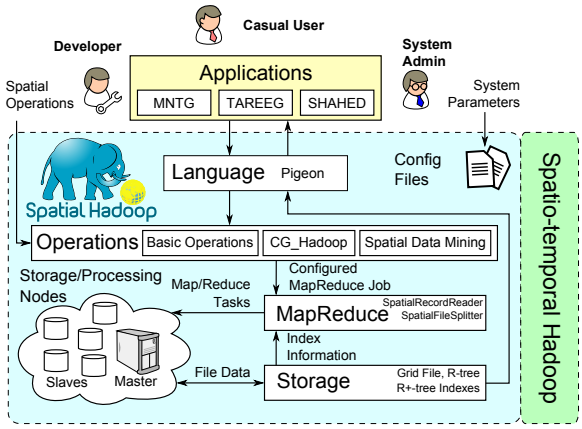


Figure 1: Overview of SpatialHadoop

2. OVERVIEW OF SpatialHadoop

Figure 1 gives an overview of SpatialHadoop. A SpatialHadoop cluster contains one master node that breaks a MapReduce job into smaller tasks, carried out by slave nodes. There are three types of users who interact with SpatialHadoop. *Casual users* deal with the system through the available applications. *Developers* are able to create new operations via MapReduce. *System admins* have deep understanding of system issues and can tune up the system through the configuration files. The core of SpatialHadoop consists of four layers described briefly below.

(1) The **Language** layer contains **Pigeon** [11], a high level language with OGC-compliant spatial data types and functions. Pigeon is discussed in Section 3. (2) The **Storage** layer contains three spatial indexes, Grid File, R-tree and R+-tree, all implemented inside the Hadoop Distributed File System (HDFS) allowing spatial operations to run much faster as opposed to heap files in Hadoop. Indexes are organized in two-layers, one global index that partitions data across nodes, and multiple local indexes to organize records inside each node. (3) We enrich the **MapReduce** layer with two new components, namely, SpatialFileSplitter and SpatialRecordReader, that allow spatial operations to access the constructed indexes. The spatial indexes and new MapReduce components are described in Section 4. (4) The **Operations** layer encapsulates the spatial operations supported by SpatialHadoop. *Basic operations* contains three standard spatial operations, range query, kNN and spatial join. *CG_Hadoop* [9] is a suite of fundamental computational geometry operations. More advanced spatial data mining techniques are also available for scalable spatial analysis of big data. The supported operations are discussed in Section 5.

The core of SpatialHadoop is designed to serve as a backbone for applications that deal with large scale data processing. In this paper, we show three case studies of real systems that use SpatialHadoop as a main component. (1) MNTG [13], a web-based traffic generator for road networks. (2) TAREEG [7], a web service for extracting spatial datasets from OpenStreetMap. (3) SHAHED, a tool for analyzing and processing large scale remote sensing data from NASA. The three applications are briefly reviewed in Section 6.

Motivated by the increased importance of the time dimension in the applications supported by SpatialHadoop, we propose *Spatio-temporal Hadoop* as an extension which adds efficient support of temporal data by following a similar approach. It adds a spatio-temporal index in the storage layer which partitions the data based on both space and time. It goes all the way up through the different layers to support efficient spatio-temporal functionality to end users. Section 7 provides a discussion of spatio-temporal indexing.

3. LANGUAGE LAYER: PIGEON

As map-reduce-like paradigms require huge coding efforts [8,15, 17], a set of declarative SQL-like languages have been proposed, e.g., HiveQL [16], Pig Latin [15], and SCOPE [8, 17]. SpatialHadoop does not provide a completely new language. Instead, it provides, Pigeon [11], an extension to Pig Latin language [15] by adding spatial data types, functions, and operations that conform to the Open Geospatial Consortium (OGC) standard [4]. In particular, we add the following:

1. Pigeon adds support for OGC-compliant spatial **data types** including, `Point`, `LineString`, and `Polygon`. Since Pig Latin does not allow defining new data types, Pigeon overrides the `bytearray` data type to define spatial data types. Conversion between `bytearray` and `geometry` is done automatically on the fly which makes it transparent to end users.

2. We add **basic spatial functions** which are used to extract useful information from a single shape; e.g., `Area` calculates the area of a polygonal shape.

3. Pigeon supports OGC standard **spatial predicates** which return a Boolean value based on a test on the input polygon(s). For example, `IsClosed` tests if a linestring is closed while `Touches` checks if two geometries touch each other.

4. Spatial analysis functions perform some spatial transformations on input objects such as calculating the `Centroid` or `Intersection`. These functions are usually used to performs a series of transformations on input records to produce final answer.

5. Spatial aggregate functions take a set of spatial objects and return a single value which summarizes all input objects; e.g., the `ConvexHull` returns one polygon that represents the minimal convex polygon that contains all input objects.

In addition to the functions in Pigeon, we do the following changes to the language.

1. KNN Keyword. A new keyword `KNN` is added to perform a k-nearest neighbor query.

2. FILTER. To support a range query, we override the Pig Latin selection statement `FILTER` to accept a spatial predicate as an input and calls the corresponding procedure for range queries.

3. JOIN. To support spatial joins, we override the Pig Latin join statement `JOIN` to take two spatial files as input. The processing of the `JOIN` statement is then forwarded to the corresponding spatial join procedure.

4. SPATIAL INDEXING

SpatialHadoop adds new spatial indexes that are adapted to the MapReduce environment. These indexes overcome a limitation in Hadoop, which supports only non-indexed heap files. There are two challenges when using traditional spatial indexes as-is in Hadoop. First, traditional indexes are designed for the procedural programming paradigm while SpatialHadoop uses the MapReduce programming paradigm. Second, traditional indexes are designed for local file systems while SpatialHadoop uses the Hadoop File System (HDFS), which is inherently limited as files can be written in an append only manner, and once written, they cannot be modified. To overcome these challenges, SpatialHadoop organizes its index in two layers, *global* and *local* indexing. The *global* index partitions data across nodes in the cluster while the *local* indexes organize data within each node. The separation of global and local indexes lends itself to the MapReduce programming paradigm where the global index is used for preparing the MapReduce job while the local indexes are used for processing map tasks. Breaking the file into smaller partitions allows indexing each partition separately in memory and writing it to a file in a sequential manner.

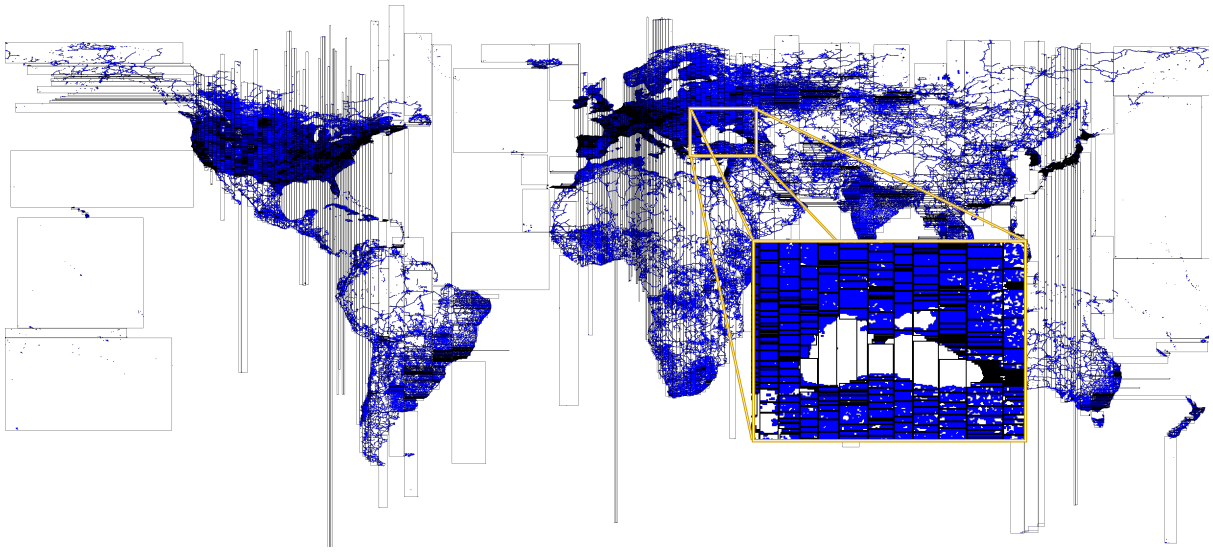


Figure 2: R+-tree index of a 400 GB OpenStreetMap dataset representing all map objects (Best viewed in color)

Figure 2 shows an example of an R+-tree index built in SpatialHadoop for a 400 GB dataset of all map objects in the world extracted from OpenStreetMap. Blue lines represent data while black rectangles represent partition boundaries of the global index. As shown in this example, SpatialHadoop adjusts the size of each partition based on data distribution such that the contents of each partition are 64MB which ensures load balancing. Records in each partition are stored together as one HDFS block in one machine.

This design is used to support three classes of indexes in SpatialHadoop, namely, Grid File, R-tree and R+-tree. Grid File is used for uniformly distributed data while R-tree and R+-tree are used for skewed data. In R-tree, records are not replicated which causes partitions to overlap. This makes it more efficient for range queries where partitions that are completely contained in query range can be copied to output and no deduplication step is required. R+-tree ensures that partitions are disjoint but some records need to be replicated. R+-tree is more efficient with spatial join where disjoint partitions allow each one to be processed independently.

The proposed index is constructed using a MapReduce job that runs in three steps. (1) The partitioning step uses our spatial-aware partitioner which distributes the data among computation nodes such that each partition fits within a 64MB HDFS block. For uniformly distributed data a uniform grid is used while for skewed data an R-tree-based partitioning is used. A record (e.g., polygon) may be replicated if it overlaps multiple partitions while the replication is handled in the operations layer to produce a correct answer. (2) In the local indexing step, each partition is locally indexed as an R-tree and stored in a separate file. (3) The global indexing step combines all files in one file and creates a global index for the partitions which is kept in the main memory of the master node.

This index is made accessible to MapReduce programs by introducing two new components in the MapReduce layer, namely, *SpatialFileSplitter* and *SpatialRecordReader*. The SpatialFileSplitter takes a spatially indexed input file and a user-defined *filter function* and it exploits the global index in the input file to prune partitions that do not contribute to answer. The SpatialRecordReader takes a locally indexed partition returned by the filter function and exploits its local index to retrieve the records that match the user query. These two components allow developers to implement many spatial operations efficiently as shown in the next section.

5. OPERATIONS

The combination of the spatial indexing with the new spatial functionality in the MapReduce layer gives the core of SpatialHadoop that enables the possibility of efficient realization of many spatial operations. In this section, we describe how developers can use the core of SpatialHadoop to implement basic spatial operations and computational geometry operations. This shows the power and flexibility of SpatialHadoop for support spatial operations.

5.1 Basic Operations

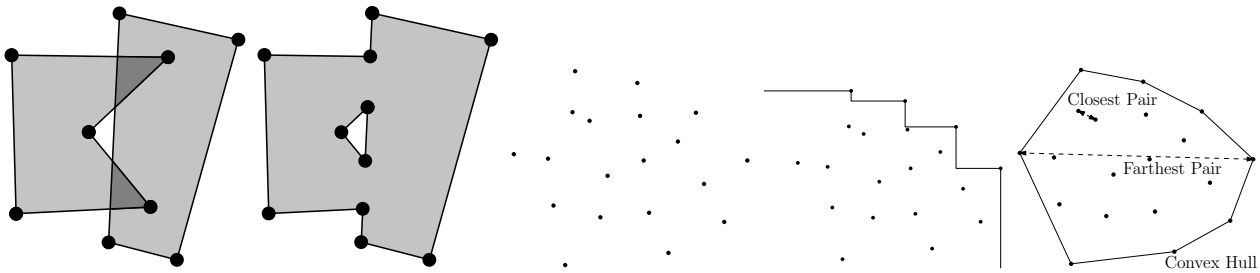
Among the available spatial operations available, we chose three basic operations, namely, range query, k-nearest neighbor and spatial join, to implement in SpatialHadoop due to their wide use. We describe range query and spatial join in this section.

5.1.1 Range Query

A range query takes a set of spatial records R and a query area A as input, and returns the records that overlap with A. In SpatialHadoop, the SpatialFileSplitter reads the global index and uses a filter function to select the partitions that overlap the query area. Partitions that are completely outside the query area are pruned as they do not contribute to answer. Each matching partition is processed in a map task, where the SpatialRecordReader extracts its local index and processes it with a range query to return records matching the query area. Finally, a duplicate avoidance step filters out duplicate results caused by replication in the index.

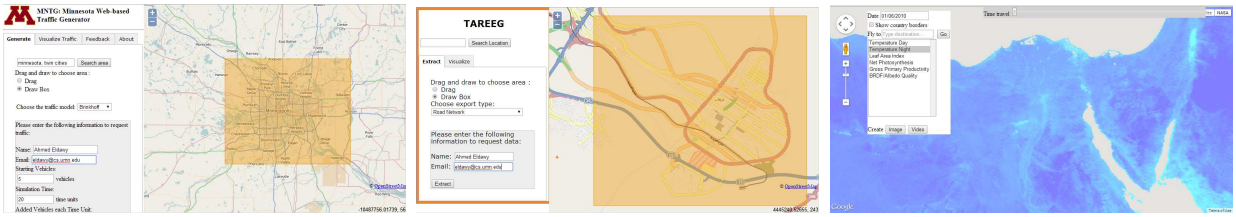
5.1.2 Spatial Join

A spatial join takes two sets of spatial records R and S and a spatial join predicate θ (e.g., `touches`, `overlaps`, or `contains`) as input, and returns the set of all pairs $\langle r, s\rangle$ where $r \in R$, $s \in S$, and the join predicate θ is true for $\langle r, s\rangle$. To implement this operation in SpatialHadoop, we feed the SpatialFileSplitter with a filter function that selects every pair of overlapping partitions. The SpatialRecordReader processes every pair of overlapping partitions and exploit their local indexes with a traditional spatial join operation to return all overlapping pairs. Finally, a duplicate avoidance step uses the reference point technique to eliminate duplication in answer caused by replication in the index.



(a) Input Polygons (b) Union (c) Input Points (d) Skyline (e) Other Operations

Figure 3: Computational Geometry Operations covered by CG_Hadoop



(a) MNTG (b) TAREEG (c) SHAHED

Figure 4: Applications built on-top of SpatialHadoop

5.2 CG_Hadoop

CG_Hadoop [9] is a suite of computational geometry operations for MapReduce. As illustrated in Figure 3, it supports five fundamental computational geometry operations, namely, polygon union, skyline, convex hull, farthest pair, and closest pair, all implemented as MapReduce. Each operation has two implementations, one for Hadoop which deals with heap files and a more efficient implementation for SpatialHadoop which utilizes the spatial index. In this section, we give the general scheme of how the operations are implemented in both Hadoop and SpatialHadoop while interested readers can refer to [9] for more details.

In Hadoop, a computational geometry operation runs in three steps. (1) The **partition** step randomly partitions input records across nodes using the default Hadoop non-location-aware partitioner. (2) The **local process** step processes each partition independently and produces a partial answer stored as intermediate result. For example in skyline, this step extracts the points on the skyline of each partition and produces them as a partial answer. (3) In the **global process** step, the partial answers are collected in one machine which computes the final answer and writes it output. For example in skyline, the points returned by all machines are collected in one machine which computes the skyline for them and returns the answer.

The drawback of Hadoop algorithms is that they need to scan the whole dataset. In SpatialHadoop, we make two modifications to overcome this limitation. (1) we use the *location-aware* partitioner provided by SpatialHadoop in the partition step which groups nearby points in one partition. This allows us to run an extra **pruning step** before the local process step. In the pruning step, partitions that do not contribute to answer are early pruned without processing. For example in skyline, we prune partitions in which all points are guaranteed to be not on the skyline. Only the remaining non-pruned partitions are processed by the local process step which makes it much faster as the input size decreases significantly. Furthermore, the global process step also becomes more efficient as the size of its input (i.e., intermediate partial result) decreases.

6. APPLICATIONS

The core of SpatialHadoop can be used to build scalable applications which deal with tons of spatial datasets. This section describes three key examples of systems that use SpatialHadoop as a powerful backend to handle spatial data processing. (1) MNTG: A web-based traffic generator for road networks, (2) TAREEG: a web service for extracting spatial datasets from OpenStreetMap, and (3) SHAHED: a tool for analyzing and processing large scale remote sensing data from NASA.

6.1 MNTG

MNTG [13], available at http://mntg.cs.umn.edu/, is a web-based traffic generator based on a real road network for the whole world. Figure 4(a) shows the interface of MNTG where users select an area on the map, specify a generation model and its parameters then the system generates the traffic data on the backend and email back the user when the job is done. One challenge that MNTG faces is extracting the road network of the selected area before sending it to the generator. As the total size of the road network dataset is around 100 GB, a full scan would be tedious to do with every request. To overcome this problem, SpatialHadoop is used to construct an R+-tree index and use this index to speed up range queries on selected areas. In MNTG, we construct an index of around 10,000 partitions with an average partition size of 12 MB. This was experimentally found to be the best based on typical request sizes.

6.2 TAREEG

TAREEG [7] is a web service for extracting spatial datasets from OpenStreetMap, available online at http://tareeg.org/. A screenshot of TAREEG is depicted in Figure 4(b). The interface is similar to MNTG where a user selects an area on the map, chooses a dataset on the left and submits an extraction request. On the backend, the server performs a range query on the selected dataset and returns the extracted data in several formats including Google KML format and ESRI Shapefile. All the available datasets are extracted from

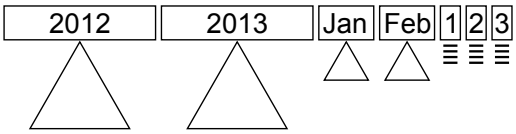


Figure 5: Spatio-temporal index

OpenStreetMap using a MapReduce extractor that runs in SpatialHadoop. A Pigeon script is used to create points and connect them to form lines which form the shapes of the data (e.g., lakes and roads). After generating the datasets, SpatialHadoop is used to build R+-tree indexes, one per dataset, in order to speed up range queries. Total size of all datasets is around 400 GB.

6.3 SHAHED

SHAHED is a tool for analyzing and exploring remote sensing data publicly available by NASA in a 500 TB archive [5]. SHAHED provides a web interface where users navigate through the map and the system displays satellite data for the selected area. For example, Figure 4(c) displays the heat map of temperature on the night of the selected date. In addition, users can select an area and ask the system to display the change of temperature over a selected time period as a video [2]. A user can also select an area and perform an analysis task on that area. For example, find anomalous patterns of vegetation in the selected area. This system uses SpatialHadoop to pre-compute the heat maps for available datasets and makes them available to the web browser for map navigation. The data mining module in the operations layer is also used to perform data analysis tasks issued by user.

7. SPATIO-TEMPORAL HADOOP

As we are working on the applications and datasets discussed earlier, we had an increased need for efficient support of the time domain. For example in SHAHED, we need to run analysis tasks on a specific time range. To support this kind of queries, we need to add a spatio-temporal index which can be used to filter underlying data by both time and space. In the first prototype, we build a temporal index on top of the spatial index. In other words, we first partition the data using the the temporal dimension, then we build a spatial index inside each partition. As the number of time partitions increase, we consolidate some old partitions into larger ones in order to decrease the number of partitions that need to be processed by a query.

Figure 5 illustrates how a spatio-temporal index looks like at a time point on March 3rd, 2014. As illustrated, the most recent partition corresponds to the current date where all new data is appended to that partition. Older partitions have been consolidated in months and even older partitions have been grouped by year. Each partition is associated with an R-tree index to speed up spatial queries inside each partition. Most recent partitions are stored as non-indexed heap files as they are too small and an index would be too much overhead.

There are three advantages to this design of spatio-temporal index. (1) It allows us to reuse existing spatial indexes which minimizes the work effort. (2) It gives the flexibility to choose the time and spatial partitioning separately based on the underlying dataset. For example, we can use a grid index for uniform data and an R-tree index for skewed data. (3) It allows data to be added incrementally to the index in a simple way as new records can be appended to the most recent partition.

[2]Please refer to an example at http://youtu.be/hHrOSVAaak8

Spatio-temporal operations can employ this index by first using the temporal partitioning then the spatial indexes. For example, a range query would run in three steps. First, a temporal filter selects the partitions that overlap the time range. Second, a spatial filter works on the matched partitions and uses their spatial indexes to select spatial partitions matching the spatial range. Finally, a refine step scans all records in matched partitions and returns records matching the spatio-temporal range.

8. CONCLUSION

This paper introduces SpatialHadoop, a full-fledged MapReduce framework with native support of spatial data available as free open-source. It enriches the core of Hadoop with native spatial data support including a spatial high level language, spatial indexes, spatial MapReduce components, and a dozen of spatial operations. We showed that many spatial operations can be realized efficiently using the built-in components. Three live applications are presented as examples of how SpatialHadoop can be used by end users. Finally, we proposed Spatio-temporal Hadoop as an extension which adds temporal support to the core of SpatialHadoop.

9. REFERENCES

[1] `http://esri.github.io/gis-tools-for-hadoop/`.
[2] `http://hbase.apache.org/`.
[3] `http://spatialhadoop.cs.umn.edu/`.
[4] `http://www.opengeospatial.org/`.
[5] `https://lpdaac.usgs.gov/sites/default/files/public/modis/docs/MODIS_LP_QA_Tutorial-1.pdf`.
[6] 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. In *VLDB*, 2013.
[7] L. Alarabi, A. Eldawy, R. Alghamdi, and M. F. Mokbel. TAREEQ: A MapReduce-Based Web Service for Extracting Spatial Data from OpenStreetMap. In *SIGMOD*, 2014.
[8] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: Easy and Efficient Parallel Processing of Massive data Sets.
[9] A. Eldawy, Y. Li, M. F. Mokbel, and R. Janardan. CG_Hadoop: Computational Geometry in MapReduce. In *SIGSPATIAL*, 2013.
[10] A. Eldawy and M. F. Mokbel. A Demonstration of SpatialHadoop: An Efficient MapReduce Framework for Spatial Data. In *VLDB*, 2013.
[11] A. Eldawy and M. F. Mokbel. Pigeon: A Spatial MapReduce Language. In *ICDE*, 2014.
[12] J. Lu and R. H. Guting. Parallel Secondo: Boosting Database Engines with Hadoop. In *ICPADS*, 2012.
[13] M. F. Mokbel, L. Alarabi, J. Bao, A. Eldawy, A. Magdy, M. Sarwat, E. Waytas, and S. Yackel. MNTG: An Extensible Web-based Traffic Generator. In *SSTD*, 2013.
[14] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. $\mathcal{MD}$-HBase: Design and Implementation of an Elastic Data Infrastructure for Cloud-scale Location Services. *DAPD*, 31(2):289–319, 2013.
[15] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-so-foreign Language for Data Processing. In *SIGMOD*, 2008.
[16] A. Thusoo. *et. al.* Hive: A Warehousing Solution over a Map-Reduce Framework. *PVLDB*, 2009.
[17] J. Zhou, N. Bruno, M.-C. Wu, P.-Å. Larson, R. Chaiken, and D. Shakib. SCOPE: Parallel Databases Meet MapReduce.