SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2383

GISQAF: MapReduce guided spatial query processing and analytics system

Khaled Mohammed Al-Naami[1,*,†], Sadi Evren Seker[2] and Latifur Khan[1]

[1]*Department of Computer Science, The University of Texas at Dallas, Dallas, TX, USA*
[2]*Department of Business, Istanbul Medeniyet University, Istanbul, Turkey*

SUMMARY

The Global Database of Event, Language, and Tone (GDELT) is the only global political georeferenced event dataset with more than 250 million observations covering all countries in the world since January 1, 1979. TABARI and CAMEO are the tools that are used to collect and code events from all international news coverage. To query such big geospatial data, traditional RDBMS can no longer be used, and the need for parallel distributed solutions has become a necessity. MapReduce paradigm has proven to be a scalable platform to process and analyze Big Data in the cloud. Hadoop, as an implementation of MapReduce, is an open-source application that has been widely used and accepted in academia and industry. However, when dealing with Spatial Data, Hadoop is not equipped well and does not perform efficiently. SpatialHadoop is an extension of Hadoop with the support of spatial data. In this paper, we present Geographic Information System Query and Analytics Framework (GISQAF), which has been built on top of SpatialHadoop. GISQAF focuses on two parts: query processing and data analytics. For the query processing part, we show how this solution outperforms Hadoop query processing by orders of magnitude when applying queries on the GDELT dataset with a size of 60 GB. We show the results for various types of queries. For the data analytics part, we present an approach for finding Spatial co-occurring events. We show how GISQAF is suitable and efficient to handle data analytics techniques. Copyright © 2015 John Wiley & Sons, Ltd.

Received 2 September 2014; Revised 21 October 2015; Accepted 9 November 2015

KEY WORDS: big data; MapReduce; Hadoop; spatial query processing; data analytics; spatial co-occurring events

1. INTRODUCTION

Living in the Big Data era, one can see the enormous growth of data that have reached an explosion. The advances in technology have resulted in ease of collecting data from numerous resources. Location-aware and position-based devices generate huge amounts of geospatial datasets, which have led to the need to extract, process, and query such information in a timely and efficient manner. The Global Data of Events, Language, and Tone (GDELT) dataset is a publicly available dataset with more than 250 million observations (called events) with global coverage since 1979 [1]. This dataset contains records and observations from all international news coverage. TABARI [2] and CAMEO [3] are the systems used to collect and prepare the GDELT dataset. The purpose of this widely used dataset is to help understand and uncover trends and behaviors of the social and international system [1].

The challenge is how to query such big data with a satisfactory performance. Traditional relational database management system (RDBMS) can no longer be used [4, 5] because of scalability and query time issues. Furthermore, as storing the GDELT data follows the 'write once read many' concept, there is no need to worry about updates. Thus, parallel distributed solutions based on "Not only

*Correspondence to: Khaled Mohammed Al-Naami, Journals Production Department, John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK.
†E-mail: khaled.al-naami@utdallas.edu

SQL" (NoSQL) have emerged [6]. Examples of these solutions include Hadoop [7], BigTable [8] Cassandra [9], and others. As a matter of fact, this is why many giant corporations have replaced traditional RDBMS with parallel distributed solutions. In their paper [10], prior to 2008, Facebook query processing was built on commercial RDBMS, which became not only expensive but also inefficient to handle the growth of the daily data generated by users, and thus they switched to Hadoop [7].

Google [11] has invented a MapReduce programming model as a parallel data processing paradigm, which has proven to be a scalable and reliable platform. Hadoop [7] is a popular open-source implementation of MapReduce that has been used for years in academia and industry [12]. On the other hand, Hadoop has some limitations. Previous attempts have been made to tune Hadoop. Tan *et al.* [13] presented a solution to fine-tune Hadoop to process input data efficiently. Liao *et al.* [12] proposed an event trigger mechanism to refine the Hadoop system. As studied in [14–16], one of the MapReduce limitations is that it is schema-free and index-free as it requires the evaluation of each record when consuming input, causing performance degradation. Files are distributed into blocks, which are stored in multiple nodes. When querying these files, MapReduce jobs are distributed across nodes in parallel. However, within each node, Map jobs process records in a sequential manner, which takes long processing times. As a result, Hadoop has not been suited to process spatial data that obviously require spatial indexing techniques such as grid index [17] or R-tree [18]. As an alternative, SpatialHadoop [19] is an open-source Hadoop-extended framework for processing spatial datasets efficiently. SpatialHadoop has been developed on top of Hadoop, so for regular MapReduce jobs it runs exactly as Hadoop, but it has the awareness of spatial data when it encounters spatial constructs and operations. Unfortunately, although SpatialHadoop is well-suited for spatial data, it has not been tuned to handle schema-like spatial datasets with such spatial archives. An example is the GDELT dataset, which has events represented by a schema.

In this paper, we introduce the Geographic Information System Query and Analytics Framework (GISQAF) that extends SpatialHadoop [20]. GISQAF achieves two objectives: query processing (QP) and data analytics (DA). The contributions of this paper are as follows.

1. This paper extends the SpatialHadoop framework so it can handle heterogeneous datasets. As a matter of fact, trying to integrate the GDELT dataset into SpatialHadoop by itself will not work as SpatialHadoop does not support schema-like datasets with multiple attributes and longitude/latitude events. In this paper, we convert these events into geometric EventPoint shapes and show how SpatialHadoop has been customized and extended to index, decode, and query the GDELT georeferenced dataset and similar datasets.
2. Geographic Information System Query and Analytics Framework adds new queries to the SpatialHadoop framework. For the QP part and based on the GDELT field parameters passed along with the query type, two types of evaluations will be shown in this study: one for the Apache Hadoop system and one for the modified SpatialHadoop system. Three types of queries have been implemented. The first is the spatial selection query. The second query is a circle-area query, which gives all events in a specific region. Aggregation query is the third type of query we show in the results.
3. We present an approach to find co-occurring events from massive spatial datasets. We show how our framework is well suited to process such tasks. For the DA part, we focus on generating spatial co-occurring events using GISQAF. Finding co-occurring events from big spatial datasets is challenging but useful for Association Rule Mining [21, 22] and Spatio-Temporal Rule Mining [23] to extract patterns and rules. This will help in understanding and uncovering spatial and perceptual trends and behaviors of the social and international system. We show how we generate two co-occurring events and use the pruning technique to generate $3, 4, \ldots, c$ co-occurring events where c is the number of frequent itemsets required.

To the best of our knowledge, this is the first attempt to address scalability for big and complex geospatial datasets that are processed and indexed geospatially over a MapReduce framework with high performance and efficient query response time. We use the GDELT dataset as a case study. Running experiments on a multi-node cluster shows that GISQAF with SpatialHadoop achieves a much better performance than Hadoop query processing.

DOI: 10.1002/spe

The rest of the paper is structured as follows. Section 2 presents a background about Hadoop and SpatialHadoop. Section 3 gives an overview of the Framework. Section 4 presents Experimental Results. Section 5 highlights Related Work. Finally, Section 6 discusses the Conclusion.

2. BACKGROUND

Before we start explaining the GISQAF framework, we briefly introduce the first two layers on which this framework has been built. This section introduces Hadoop [7] and SpatialHadoop [19] applications. Then we describe the GDELT data points.

2.1. Hadoop and SpatialHadoop

Hadoop [7] is an open-source MapReduce implementation for processing big volumes of data in a distributed manner on large clusters. Inspired by Lisp and proposed by Google [11], MapReduce is a programming model that follows a certain functional style to process large datasets. This functional style hides the details of data distribution, task scheduling, failure handling, and communication from the user. Basically, the user writes map and reduce functions to achieve a particular task. Depending on the number of jobs associated with the program, jobs are submitted for execution to the master node. Assuming data has been already distributed into chunks between slave nodes using Google File System (GFS) or Hadoop Distributed File System (HDFS) in case of Hadoop, the master node divides the job into tasks, each of which is associated with particular chunks or blocks, and – by following pull scheduling strategy – assigns each task to a slave node. Slave nodes run map and reduce functions and report to the master node when finished. Each map function expects key-value pairs as input and produces intermediate key-value pairs as well. The intermediate key-value pairs are partitioned, and each unique key is sent to a single reducer along with its list of values emitted by multiple mappers. Reduce function, usually iterates over a list of values for a specific key, processes the computation assigned to it, and emits the final output as a key-value pair, which is written back to the distributed file system.
SpatialHadoop [19] is an open-source Hadoop-extended geospatial framework for processing massive spatial datasets efficiently. SpatialHadoop is an extension to Hadoop, and the aim of this project is to process large spatial datasets on Hadoop with the native support for spatial data. It is available at [24] as an open-source so contributors can extend its functionality and modify it to process different spatial queries and operations. Programs in this application run the same way as in Hadoop, implementing map and reduce functions with the awareness of spatial operations. The basic idea of SpatialHadoop is to index georeferenced datasets using Grid file [17], R-Tree [18], or R+-Tree [25] indexing techniques. The spatial indexing is carried out as a MapReduce job and stored in the HDFS. Two-level indexing is used, global and local.

In general, the global index holds information about each cell or Minimum Bounding Rectangle (MBR) that point to an individual file that contains all spatial objects (Point, Rectangle, or Polygon) in that cell. Following the trend of Hadoop, the local index is responsible for distributing files in each slave node.

SpatialHadoop constructs the index in three phases, partitioning, local indexing, and global indexing. In the partitioning phase, the input file is divided into spatial partitions of 64MB in size such that each partition is contained in a rectangle. This is based on the type of indexing used (Grid index, R-Tree, or R+-Tree). In the local indexing phase, an in-memory local index is constructed for each partition and sent to HDFS as a 64MB block. The global indexing phase concatenates all local indexes and builds one big global index file that holds the MBRs and partition names. While the local indexes reside in the slave nodes, the global index resides in the master node.

When submitting spatial queries such as range queries, SpatialHadoop applies a pruning step before starting MapReduce jobs. The pruning or filtering step loads only partitions that intersect with the query MBR and prunes the ones that do not. This ensures that only shapes contained in the MBR are processed. This leads to a significant performance improvement as compared with Hadoop, which loads all entries from all partitions. A shape might fall in more than one partition. To avoid reporting the shape twice to the query result, SpatialHadoop uses Duplicate Avoidance Technique [26]. In a recent study [27], different computational geometry problems have been implemented successfully using SpatialHadoop.

DOI: 10.1002/spe

2.2. *Global Data of Events, Language, and Tone data points*

Handled by the TABARI system, each event (data point) in GDELT is georeferenced by latitude and longitude locations. Events happen between two actors or parties that usually represent two countries. Figure 1 and Table I show an example of an event that took place on September 2, 2013. GDELT is concerned about how to translate this textual context to a georeferenced CAMEO-coded observation. This is performed through crawling international news sources and implementing spatial archiving techniques to get the 58 coded fields for each event using the TABARI system and other software. In the CAMEO geocoding system, each event in the GDELT dataset has two Actors. These are the two main parties of each political observation. Each event is supposed to happen between Actor1 and Actor2. In some cases, Actor1 is the sole party of the event. As Figure 1 and the Actor Attributes Fields in Table I show, Actor1 in this event is Russia and Actor2 is Syria. The news talks about a summit in Russia that discussed the conflict in Syria in which chemical weapons have been used against civilians and the possible UN international response to that. Actor1Geo_Lat, Actor1Geo_Long, Actor2Geo_Lat, and Actor2Geo_Long are the location fields that hold latitude and longitude for Russia and Syria, respectively. As shown in Figure 1 and Event Geography Fields in Table I, the latitude/longitude values of Russia and Syria are of 59.8944/30.2642 and 35.0000/38.0000, respectively.

From the record example in Table I, data fields or attributes used in GDELT events represent the following:

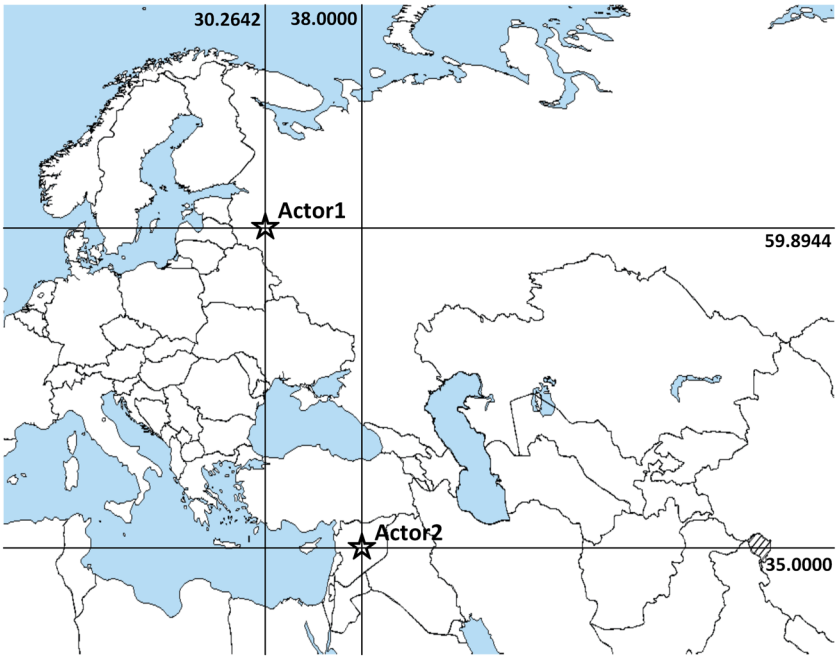

Figure 1. A GDELT event between two actors.

Table I. An example of a GDELT dataset event.

FIELDS	VALUES
EVENTID AND DATE ATTRIBUTES	266765660, 20130902, 201309, 2013, 2013.6630
ACTOR ATTRIBUTES	RUS ST PETERSBURG RUS, SYR SYRIA SYR
EVENT ACTION ATTRIBUTES	1, 036, 036, 03, 1, 4.0, 6, 2, 6, 0.816326530612245
EVENT GEOGRAPHY	4, Petersburg, Sankt-Peterburg, Russia, RS, RS66, 59.8944, 30.2642, -2996338, 1, Syria, SY, SY, 35, 38, SY, 1, Syria, SY, SY, 35, 38, S
DATA MANAGEMENT FIELDS	20130909, http://www.newkerala.com/news/story/65045/world-should-respond-if-syria-used-chemical-weapons-ban.html

DOI: 10.1002/spe

1. EVENTID AND DATE ATTRIBUTES. The first few fields represent a unique identifier and information about the event date (day, month, and year).
2. ACTOR ATTRIBUTES. The next fields describe the two actors. An example of these attributes is Actor1Code, which is represented by three alphabetic CAMEO-coded letters to describe the first party of the event. Other fields include Actor1 name, ethnic code, religion code, and others. The same fields are repeated for Actor2 as well.
3. EVENT ACTION ATTRIBUTES. These fields offer the importance or impact of the event. An example is IsRootEvent, which helps to track the stream or chain of events. Another example is EventCode, which describes the action that Actor1 performed against Actor2. The QuadClass integer parameter specifies the primary classification of the event type as follows: 1 means Verbal Cooperation, 2 means Material Cooperation, 3 means Verbal Conflict, and 4 indicates Material Conflict. The Goldstein Scale numeric coded field describes the theoretical effect of this event on the stability of a country.
4. EVENT GEOGRAPHY. These are the attributes that georeference the GDELT. Each event contains a longitude/latitude point. In these fields, Actor1 and Actor2 locations are represented. As mentioned earlier, some of the fields include Actor1Geo_Lat, Actor1Geo_Long, Actor2Geo_Lat, and Actor2Geo_Long, which represent Actor1 latitude, Actor1 longitude, Actor2 latitude, and Actor2 longitude, respectively.
5. DATA MANAGEMENT FIELDS. The last set of fields provides data management information for the event. This is useful for database management purposes. DATEADDED is an integer field that tells when the event was added to the database. Another field is the SOURCEURL field, which adds the URL of the news source. If multiple news sources are used, only one is recorded. This field was added to the GDELT collection on April 1, 2013.

3. FRAMEWORK

3.1. Architecture

GISQAF is an extension to SpatialHadoop. Figure 2 shows how it has been built on top of SpatialHadoop. Layer 1 (bottom layer) of this design is the HDFS storage layer. Layer 2 (middle) consists of the extended SpatialHadoop and Hadoop. The terms SpatialHadoop and Extended SpatialHadoop will be used interchangeably through the rest of the paper. Layer 3 is responsible for the following:

1. **Preprocessing the datasets.** Because SpatialHadoop expects shape files as an input, each data point in the required-to-be-indexed dataset has to be converted to a form of a shape object depending on the data point geometry type (i.e., Point, Rectangle, or Polygon). In GDELT, a data point represents a longitude/latitude location so each event is represented as an EventPoint, which is an extension of the Point shape.
2. **Spatial Indexing.** Layer 3 communicates with the Extended SpatialHadoop in layer 2 to index the dataset. The Extended SpatialHadoop communicates with the storage layer (layer 1) to construct the index and save it in the HDFS.
3. **Query Processing.** Here, layer 3 interacts with the client to process the queries. Depending on the query type, the Extended SpatialHadoop is called to deal with spatial queries while Hadoop is called to handle non-spatial queries and regular MapReduce jobs.
   More details are presented in the following sections.

3.2. Global Data of Events, Language, and Tone preprocessing and spatial indexing

The GDELT spatial dataset has to be preprocessed and transformed into a format with which SpatialHadoop can deal. SpatialHadoop is able to index shapes (Points, Rectangles, and Polygons) and move them to the HDFS storage layer. Following a bottom-up fashion, local indexes are constructed first followed by the global index. An event in the GDELT dataset has 58 fields including the location. Figure 3 depicts this process as follows:

DOI: 10.1002/spe

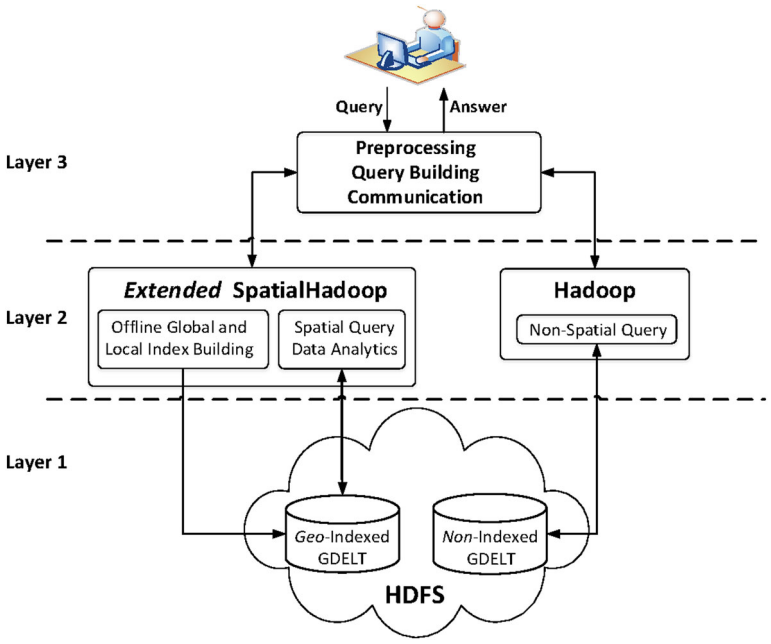

Figure 2. GISQAF Architecture.

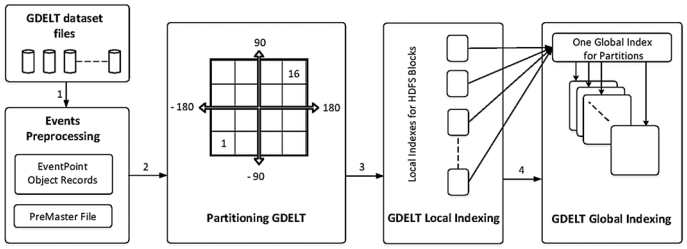

Figure 3. Global data of events, language, and tone preprocessing and spatial indexing.

1. We extract the longitude/latitude in a way that converts each event to an *EventPoint* Object that contains x and y geometry points. In the mean time, this object has to keep all the CAMEO-coded fields of the event in order to be used in the spatial queries. A PreMaster file is prepared to be used by the Extended SpatialHadoop for the partitioning phase. Each dataset in the GDELT collection is represented by a record (one line) in the PreMaster file. This record has two components: the MBR of all the data points in the file and the file name. A rectangle is represented by two points, (x1, y1) as the left lower corner and (x2, y2) as the dimensions. Thus, in the PreMaster file, the MBR's two points are (−180, −90) and (180, 90), which considers the longitude and latitude of the whole geographic map area. This step is shown in lines 1 to 8 in the pseudo code in Algorithm 1. As mentioned in Section 2.2, in some cases Actor1 may be the sole party of the event. This is why we extract the point of Actor1 in Algorithm 1, line 4. As shown in line 10 in Algorithm 1, the preprocessed GDELT datasets along with the PreMaster File are saved to the HDFS. This is the non-indexed GDELT shown in Figure 2.
2. This step partitions the GDELT by dividing it into spatial partitions. This is based on the type of indexing used (Grid index, R-Tree, or R+−Tree). For clarity, and as an example, Figure 3 shows 16 uniform partitions. Partition 1 or rectangle 1 is represented by the left lower corner (−180, −90) and dimensions (−90, −45). Partition 2 is represented by (−90, −90) and (0, −45), and partition 16 is represented by (90, 45) and (180, 90), and so on. This is represented in line 12 in Algorithm 1.

DOI: 10.1002/spe

Algorithm 1 Preprocessing and Spatial Indexing

```
1: preparePreMasterFile();
2: for each GDELT dataset file do
3:     for each event do
4:         EventPoint ← extractActor1LongitudeLatitude(event);
5:         EventPointRecord ← convertToCartesianCoordinates(EventPoint);
6:         updateEvent(EventPointRecord);
7:     end for
8: end for
9: /* End Processing all GDELT dataset files */
10: storeNonIndexGDELT();
11: /* Start Spatial Indexing */
12: partitionGDELT();
13: buildLocalIndex();
14: buildGlobalIndex();
15: storeIndexedGDELT();
```

3. Local indexing is constructed for each partition, prepared in step 2, and sent to HDFS as a 64 MB block. Each local index file has meta data for some blocks and their MBRs. See line 13 in Algorithm 1.
4. All local indexes are concatenated into one big global index stored in the main memory of the master node. Basically, each line in the global index file has the partition name along with the MBR or cell that contains the relative EventPoint shapes. Line 14 in Algorithm 1 depicts this step.
   Building a spatial index is a bottom-up process as local indexes are built first followed by the global index. On the other hand, querying is a top-down process as the global index is called first followed by the local index. We will see that in the following section.
   Finally, line 15 in Algorithm 1 shows how the indexed GDELT is stored in the HDFS.

3.3. Global Data of Events, Language, and Tone querying

As the system is now aware of spatial data, when submitting the spatial queries to GISQAF, EventPoint partitions that do not contribute to the final answer will not be examined. As mentioned earlier, this is performed in a top–down fashion by calling the global index first and then the local indexes. Figure 4 illustrates the query execution in GISQAF as follows:

1. Framework passes the query along with the MBR filter, which might be of any type of geometric shape (i.e., Point, Rectangle, Circle, or Polygon). In this step also, the framework

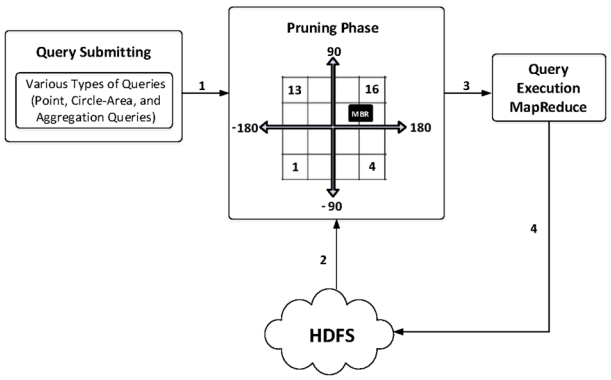

Figure 4. Spatial-indexed GDELT query execution.

DOI: 10.1002/spe

distinguishes between spatial and non-spatial queries. In Figure 4, we consider the case of spatial query. This step is shown in lines 1 through 5 in the pseudo code of Algorithm 2.

Algorithm 2 Query Processing Pseudo Code

```
1:  passQueryAndMBR();
2:  if spatialQuery then
3:      /* Extended SpatialHadoop Data Handling */
4:      getIndexedGDELT();
5:      getQueryMBR();
6:      BP ← findAppropriatePartitionsUsingGlobalIndex(); /* Big Partitions */
7:      SB ← findSmallerBlocksUsingLocalIndex(BP); /* Smaller Blocks */
8:      runMapReduceExtendedSpatialHadoop(SB); /* Algorithm 3 */
9:  else
10:     /* Hadoop Query Processing */
11:     getNonIndexedGDELT();
12:     runMapReduceHadoop();
13: end if
```

2. According to the MBR of the filter shape passed, step 2 gets only partitions that contribute to the query result. The global index, which is kept in the master node, is used to determine the partitions that overlap with the MBR filter. This is performed using SpatialFileSplitter in SpatialHadoop [19]. For clarity, we assume there are 16 partitions as Figure 4 shows. Partition numbers 11 and 12 overlap with the query MBR filter (shown as a black box). Accordingly, only these two partitions are to be picked to be processed. Algorithm 2, line 6 shows this global index pruning step.
3. Before executing the map function, the local index is utilized to load records using SpatialRecordReader [19]. Map function processes these EventPoints according to the query requirements. Local index use and MapReduce call are shown in lines 7 and 8 of the pseudo code of Algorithm 2. GDELT coded field awareness has been injected to the map and reduce functions to choose the particular fields passed in the query line to get the final results.
4. The results are written back to the HDFS storage layer for the client to view them.

3.4. Query types

From GDELT, extracting and analyzing events that happen in specific regions of the globe are highly demanded. This helps decision makers implement policies and uncover patterns of social evolution. For example, selection queries of protest events in specific locations can be mapped to understand global protest trends [1]. In this study, three types of queries have been implemented as follows. The first is the spatial selection point query. This query simply selects observations and events that are associated with any latitude/longitude location. The latitude/longitude location is transformed to be dealt with as an x-y geometry point. Every GDELT event has particular fields for latitude/longitude locations that show where the event has taken place. For instance, we might be interested to select all events happening in the capital of Iraq, Baghdad. Query latitude/longitude parameters would be 33.3335/44.3521. As this is a selection query, map tasks simply emit the selected events and no reducer is needed.

The second query is a circle-area query which extracts all events in a specific region surrounding a point of interest. In this query, geometry circle class has been implemented in a similar way to the SpatialHadoop range query. However, dataset schema has been considered to parse events in GDELT. Besides, the query region is passed to the framework as x, y, and r, where x and y are the coordinates of the center of attention and r is the radius of the coverage. For instance, the center of attention may be the capital of South Sudan, Juba, with the latitude/longitude value of 4.8455/31.5859, and all events around this point with any particular radius. Similar to query 1, no reducer is needed in this query as well.

DOI: 10.1002/spe

Algorithm 3 Aggregation Count Query Pseudo Code

```
1:  procedure MAP(k, v) /* Input to MAP is SB, Smaller Blocks Records */
2:      /* k is the queryMBR Rectangle shape
3:      v is the EventPoint shape */
4:      groupByClause ← YearMonthQuadClass; /* From EventPoint */
5:      Actor1Code ← "CHN"; /* Job Configuration */
6:      Actor2Code ← "TWN"; /* Job Configuration */
7:      if v.isIntersected(k) then
8:          if EventActor1.equals(Actor1Code) & EventActor2.equals(Actor2Code) then
9:              emit(groupByClause, 1);
10:         end if
11:     end if
12: end procedure
13: procedure REDUCE(k, v)
14:     /* k is the group by clause
15:     v is the list of counts */
16:     for each value in v do
17:         count += value;
18:     end for
19:     emit(k, count);
20: end procedure
```

Aggregation query is the third type of query we show in the results. In this query, we show a query of interest when dealing with GDELT data points. We explain this query with an example. The aggregate function of interest here is to count the number of events between any two actors, such as China and Taiwan, with the parameters Actor1Code = 'CHN' and Actor2Code = 'TWN' and apply 'group by' clause for the year, month, and QuadClass of the event. This type of query can be used to explore evolving situations between two parties [1]. CHN and TWN will be translated into georeferences to get the MBR in order for the search to be more efficient using spatial filtering and pruning. Algorithm 3 shows the MapReduce pseudo code for the third query. The map function key is the MBR (Rectangle shape) of China and Taiwan calculated from the geographic world map. The map function value is the EventPoint shape, which is basically a GDELT data point that holds the x-y location and all the CAMEO-coded fields. See lines 1 through 3 in Algorithm 3. Notice that there is a pruning step before this MapReduce job as explained earlier in lines 6 and 7 of the pseudo code in Algorithm 2, which shows Smaller Blocks (SB) as the blocks that the map function will consume. As represented in lines 7 and 8 of Algorithm 3, each EventPoint shape from the SBs is examined to check if intersected with the MBR shape (query filter). If intersected, then Actor1 and Actor2 are checked to make sure they match China and Taiwan passed by query condition. If matched, then the EventPoint record count of one is emitted by the map function with the group by clause as the intermediate key and one as the intermediate value. See line 9 in Algorithm 3. In this query, the combination of Year, Month, and QuadClass forms the intermediate key. This ensures that the reducer gets a unique key with a list of ones as the value. After summing up these ones, reduce emits the group by clause as the final key and the total count as the final value. The reduce function is shown in lines 13 through 20 in Algorithm 3. Other types of queries are possible but we focused on these three types in this study as they can be extended to include other query requirements. For instance, queries can include time as well.

Comparing SpatialHadoop with Hadoop, as we will show in the experiments section, Hadoop will not work efficiently with such queries as no indexing is being followed.

3.5. Finding co-occurring events approach

3.5.1. Problem statement. To develop marketing strategies, discovering association rules between purchased items from a large collection of transactions, called *market-basket data*, was introduced

DOI: 10.1002/spe

Table II. Market-basket transactions.

TID	Items
100	MILK, BREAD, EGGS
200	BREAD, PANCAKES, SUGAR, SYRUP
300	BREAD, CEREAL, MILK
400	MILK, BREAD, SUGAR, EGGS
500	SYRUP, MILK, CEREAL, PANCAKES
600	PANCAKES, SYRUP
700	CEREAL, MILK

in [21, 22]. As an example, Table II shows a set of transactions. Each transaction has a set of purchased items. The idea is to understand customer buying habits by finding association rules that will predict occurrences of purchased items. Each rule has two parts, an *if* antecedent and a *then* consequent. For instance, from these transactions, the following rules may be generated.

$$\{BREAD, MILK\} \Rightarrow \{EGGS\}$$
$$\{PANCAKES\} \Rightarrow \{SYRUP\}$$
$$\{CEREAL\} \Rightarrow \{MILK\}$$

The first rule implicates that customers who buy bread and milk tend to buy eggs as well. Notice that the implication in the previous three rules means co-occurrence and not causality.

To achieve this, *Association Rule Mining* algorithms find frequent itemsets and extract rules from these itemsets. Frequent itemsets are the k-itemsets that satisfy a predefined minimum support. From the previous table, if we take the minimum support as 40%, then {CEREAL, MILK} is one example of a 2-itemset as it has the support 3/7, which is greater than 40%. Similarly, we can find all the $< 1, 2, 3, \ldots >$-itemsets. From these itemsets, rules that satisfy a certain minimum confidence can be generated. The confidence of a rule $X \Rightarrow Y$ is defined as the ratio of the number of times items in Y appear in transactions that contain X to the number of times items in X occur.

For instance, from the frequent 2-itemset {CEREAL, MILK}, the number of times cereal and milk happen together is 3 and the number of times milk happens is 5. So for the rule {CEREAL} $\Rightarrow$ {MILK} to be considered, the confidence threshold should be at most 60% or 3/5.

Finding frequent itemsets from a large database is expensive. If we have 100 purchased items, the number of candidate itemsets is 2^{100}, which is very large. Frequent itemset generation requires efficient algorithms like Apriori [22], which prunes all parent itemsets that have infrequent children itemsets. For instance, {BREAD, MILK, EGGS} is to be evaluated only if each of the subsets (parent nodes) has a support that is greater than or equal to minimum support. If {MILK, EGGS} has a support less than the minimum support, all children nodes of {MILK, EGGS} including {BREAD, MILK, EGGS} will be pruned. This ensures to reduce the running time complexity tremendously.

In our case, instead of having purchased items, we have geo-spatial political events in the GDELT dataset. We are interested in generating rules such as {PROTEST} $\Rightarrow$ {VIOLENCE AGAINST CIVILIANS}, {VIOLENCE AGAINST CIVILIANS} $\Rightarrow$ {MIGRATION}, and {TERRORIST ATTACKS} $\Rightarrow$ {MIGRATION} and see how they are associated over time around the globe. For example, the migration of Syrian refugees, due to the civil war, through the European countries Greece, Macedonia, Serbia, Hungary, Austria, and Germany can be analyzed to generate such associated rules. We can pick the month of August 2015 and the MBR of Eastern Europe to extract meaningful migration patterns. We show how GISQAF is very efficient to handle such tasks. Similar to the general rule mining technique described above, finding frequent itemsets (eventsets) is important to extract meaningful rules. We define frequent events as co-occurring events (i.e., events located next to each other that happen at a particular time window). Given a large collection of events like the GDELT dataset, the problem this paper addresses is as follows. From the spatial database tuples (or events) collection, find all $2, 3, 4, \ldots, c$ conflicting co-occurring events (frequent itemsets or eventsets) that happen in a particular geographical region (denoted by MBR) considering a specific time window and a particular geographical radius.

To generate co-occurring events from big datasets like GDELT, we need to use an efficient spatial-aware framework. We show how GISQAF is an efficient geospatial framework using this scenario

DOI: 10.1002/spe

as a case study. After finding frequent itemsets, Spatio-Temporal Rule Mining [23] can be used to generate rules to enable policy makers to develop strategies. The terms co-occurring events and frequent itemsets/eventsets will be used interchangeably in this paper. The problem can be formulated as follows.

Given

$$R(t_1, t_2, t_3, \ldots, t_n);$$
$$S(t_1, t_2, t_3, \ldots, t_m);$$
$$\chi = \{< t_i, t_j >, < t_i, t_j, t_k >, \ldots, < t_i, t_j, t_k, \ldots, t_m >\};$$
$$\delta;$$
$$M;$$
$$r;$$
$$c.$$

find all co-occurring tuples

$$\zeta = \{< t_i, t_j >, < t_i, t_j, t_k >, \ldots, < t_i, t_j, t_k, \ldots, t_c >\}.$$

such that

$$\begin{array}{l} S \subseteq R, \textit{ and hence } m \leqslant n; \\ \zeta \subseteq \chi, \textit{ and hence } c \leqslant m; \\ \forall t,\ t \in S :\ t.\textit{LatLongPoint} \in\ M \textit{ and } t.\textit{time} \in \delta; \\ \forall < t_a, \ldots, t_b > \ \in\ < t_i, t_j, t_k, \ldots, t_c >:\ < t_a, \ldots, t_b > .\textit{radius} \leqslant r \end{array} ,$$

where

R: Collection of spatial database events in a time window δ;
S: Subset of R;
χ: All possible candidate itemsets in S;
δ: Time window;
M : MBR;
r : Radius;
c : Number of frequent itemsets required;
ζ : All $2, 3, 4, \ldots, c$-itemsets (co-occurring events);
$< t_i, t_j >$: All two co-occurring events;
$< t_i, t_j, t_k >$: All three co-occurring events;
$< t_i, t_j, t_k, \ldots, t_c >$: All c co-occurring events;
LatLongPoint: Latitude and Longitude of an event.

3.5.2. Example 1. To clarify, we present an example using the GDELT dataset. Given the year 2012 and month of December as the time window δ and the MBR $M = (36.6723, 36.5979, 38.1665, 37.1078)$ as longitude1, latitude1, longitude2, and latitude2, which is located at the borders of Syria and Turkey, we are interested in finding all 2, 3, 4 co-occurring events within a 5-mile radius so $c = 4$ and $r = 0.09°$, which is equivalent to 5 miles using the Haversine Formula [28] to convert from miles to longitude/latitude distance. Applying range query produces $m = 242$ events. All candidate co-occurring events in χ are shown in Figure 5.

The number of all candidate itemsets is $2^m = 2^{242}$, which gives a very large number. Figure 5 shows how pruning excludes all super sets of $< t_1, t_2 >$ as the two events do not co-occur (not within a five-mile radius of each other). In this example we set $c = 4$ to get the subset ζ. We will show how the algorithm works in the coming sections. After applying the algorithm to the GDELT dataset and getting the 242 events, we get 21796 two co-occurring, 356461 three co-occurring, and 9347136 four co-occurring events. As an example, we take one of the 21796 two co-occurring events $< t_i, t_j > = < t_{28}, t_{109} >$. The latitude/longitude values of t_{28} and t_{109} are 36.6439/37.0875 and 36.7161/37.115, respectively.

DOI: 10.1002/spe

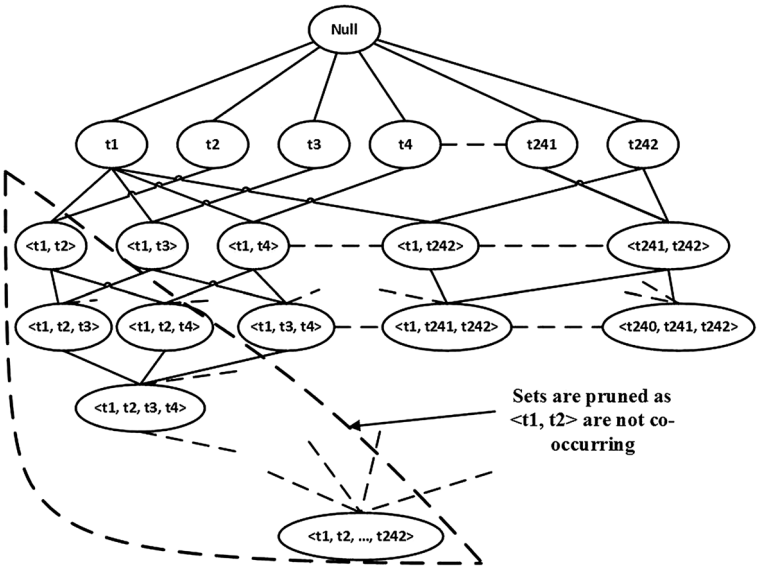

Figure 5. All candidate co-occurring tuples (events) in χ.

Algorithm 4 Pseudocode to find Spatial Co-occurring Events

Input:
R: Collection of spatial database events in a time window δ;
M: Minimum Bounding Rectangle (MBR);
r: Radius;
c: Number of frequent itemsets required;

Output: All 2, 3, ..., c co-occurring events.

1: $Indexed\_R \leftarrow$ Geo-Index(R)

2: $S \leftarrow$ Extended-Range-Query($Indexed\_R, M$)

3: **for** *each tuple* t_i *in* S **do**
4: $CQ = CQ +$ Circle-Query(t_i, r)
5: **end for**

6: $B \leftarrow$ Build-Matrix(CQ)

7: **for** $d = 3 \rightarrow c$ **do**
8: $\zeta \leftarrow$ Find-d-Co-Occurring-Events(B)
9: **end for**

3.5.3. An approach to find spatial co-occurring events. In this section we show how we find the spatial co-occurring tuples or events. We use GISQAF to accomplish this task. Algorithm 4 and Figure 6 explain this process as follows.

1. As Spatial Data requires spatial indexing techniques such as grid index, or R-tree, and for fast query response (Range Query and Circle Query used in the following steps), using GISQAF, we first index GDELT events that happened in the time window, denoted earlier as $R(t_1, t_2, t_3, ..., t_n)$, to build the geo-spatial indexes as shown in line 1 of Algorithm 4. Besides,

DOI: 10.1002/spe

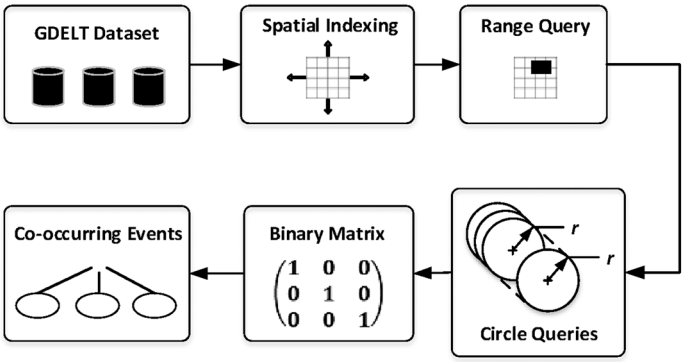

Figure 6. Co-occurring events block diagram.

as the problem specifies a Minimum Bounding Rectangle (MBR) region M, geo-indexing fastens query response by fetching HDFS blocks that contribute to the query result.

2. As the system is now aware of spatial data, when submitting the spatial queries to GISQAF, partitions that do not contribute to the final answer will not be examined. This step, Algorithm 4, line 2, applies Range Query (Extended SpatialHadoop Range Query to incorporate GDELT events) where the framework passes the query along with the MBR M filter. The result is the Database $S(t_1, t_2, t_3, ..., t_m)$ where $S \subseteq R$. Each tuple t_i in S contains the GDELT event ID and the event details. Notice that up to this point and as the example in Figure 5 shows, all possible co-occurring events (candidate itemsets) $\chi = \left\{< t_i, t_j >, < t_i, t_j, t_k >, \ldots, < t_i, t_j, t_k, ..., t_m >\right\}$, can be built from S.
3. For each tuple $t_i \in S$, we execute a Circle Query (Extended SpatialHadoop MapReduce Query) to get all the 2 co-occurring events. The query in Algorithm 4, line 4, extracts all events in a specific region surrounding a point of interest. The query region is passed to the framework as $(x,\ y,\ r)$ where x and y are the coordinates of the center of attention which is t_i and r is the radius of the coverage. Then the Circle Query finds all two co-occurring events, $< t_i, t_j >$ where t_j represents any tuple that is within the Radius r from t_i. As the number of Circle Queries is equal to $|S|$, Extended SpatialHadoop performs much faster than Hadoop as will be explained later in the Experiments Section.
4. After finding all two co-occurring events, line 6 of Algorithm 4 builds a Binary Matrix B that contains only zeros and ones. The form of the Binary Matrix B is represented as follows:

$$b_{ij} = \begin{cases} 1 & \text{if } < t_i, t_j > \ \ co-occur \\ 0 & \text{otherwise} \end{cases}$$

Notice that all values in the diagonal of B are ones as each event co-occurs with itself by default (i.e., $< t_i, t_i >$ co-occur intuitively). Besides, B is a symmetric matrix around the diagonal as $b_{ij} = b_{ji}$.

5. From the binary matrix B, to find $\zeta = \{< t_i, t_j >, < t_i, t_j, t_k >, \ldots, < t_i, t_j, t_k, \ldots, t_c >\}$ (lines 7 - 9, Algorithm 4), we define $< t_i, t_j, t_k, t_l, \ldots, t_d >$ co-occurring events where $3 \leqslant d \leqslant c$. For each d, to decide whether $< t_i, t_j, t_k, t_l, \ldots, t_d >$ co-occur or not, we check if each two events co-occur by examining the binary matrix B. For each $< t_a, t_b, \ldots, t_z >$ in $< t_i, t_j, t_k, t_l, \ldots, t_d >$, if $b_{ab} = 1, \ldots, b_{az} = 1, \ldots, b_{bz} = 1, \ldots$, then all events in $< t_i, t_j, t_k, t_l, \ldots, t_d >$ co-occur. If any subset of $< t_i, t_j, t_k, t_l, \ldots, t_d >$ does not co-occur, then there is no need to continue as definitely there is no co-occurrence for the tuples in $< t_i, t_j, t_k, t_l, \ldots, t_d >$. Thus, as discussed earlier in Figure 5, if tuples in a parent node do not co-occur, then this node and all descendent nodes will be pruned.

The pruning step in our algorithm is different than the one in the Apriori algorithm [22]. In Apriori, pruning of a node happens if any of its parents has a support lower than a predefined threshold. Here, the node pruning takes place if events do not co-occur in any of its parents.

DOI: 10.1002/spe

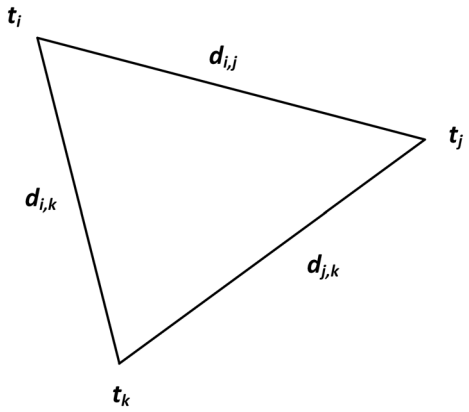

Figure 7. Three co-occurring events.

Lines 1 through 5 in Algorithm 4 present our framework solution to find co-occurring events. The rest of Algorithm 4 is simply implemented in a single machine as this does not require a distributed system.

3.5.4. Example 2. In this section, we give an example for finding Spatial Co-occurring Events. Although this is not the case in practice, for the sake of clarity we assume a small number of tuples which is $m = 6$ as the result of the Range Query with M as our MBR. So we have $S(t_1, t_2, t_3, t_4, t_5, t_6)$. All events in S happened in Dec 2012 which is our Time Window δ. All possible co-occurring events in S are 2, 3, 4, 5, 6 co-occurring events which gives us $\chi = \{< t_i, t_j >, < t_i, t_j, t_k >, < t_i, t_j, t_k, t_l >, < t_i, t_j, t_k, t_l, t_o >, < t_i, t_j, t_k, t_l, t_o, t_m >\}$. For each tuple $t_i \in S$, we execute the Circle Query to get all 2 co-occurring events and build our Binary Matrix B as follows.

$$B = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Let us assume that $c = 4$ as we are interested in finding all 2, 3, 4 co-occurring events so $\zeta = \{< t_i, t_j >, < t_i, t_j, t_k >, < t_i, t_j, t_k, t_c >\}$. All 2 co-occurring events $< t_i, t_j >$ have been already obtained in the matrix B. To find all 3 co-occurring events $< t_i, t_j, t_k >$ where $1 \leqslant i, j, k \leqslant 6$, from B, we check if $< t_i, t_j >, < t_i, t_k >$, and $< t_j, t_k >$ co-occur by examining if $b_{ij} = b_{ik} = b_{jk} = 1$. If yes, then t_i, t_j, and t_k co-occur and we add $< t_i, t_j, t_k >$ to the frequent eventsets.

Figure 7 shows three co-occurring tuples. Notice that $(d_{i,j} \leqslant r)$ is the distance between the two geospatial events t_i and t_j. Similarly, we find all 4-co-occurring events.

After running the algorithm for this example, the 3 co-occurring events are $< t_1, t_2, t_3 >$, $< t_1, t_2, t_5 >$, $< t_1, t_3, t_5 >$, $< t_2, t_3, t_5 >$, $and < t_3, t_4, t_5 >$. There is only one 4 co-occurring event which is $< t_1, t_2, t_3, t_5 >$. Notice that event t_6 is an outlier as it does not co-occur with any other event.

The summary of this example is as follows. The number of events in the MBR M is 6, the number of two co-occurring events is 21, the number of triple co-occurring events is 5, and the number of 4 co-occurring events is 1.

4. EXPERIMENTS

In this section we study the performance of GISQAF. We execute queries in two platforms, one using GISQAF which uses SpatialHadoop, and the other using Hadoop. We show the results when using a single-node cluster and a multi-node cluster. Both Hadoop and SpatialHadoop clusters

DOI: 10.1002/spe

run Apache Hadoop 1.2.1 and Java 1.6. All experiments have been implemented using internal university machines. The dataset used is the GDELT dataset explained earlier.

4.1. Experimental setup

Single-node cluster This single machine has a 24GB RAM, HDD of 2.2TB, and a 16 core CPU of 2.40GHz with Ubuntu 12.04 operating system.

Multi-node cluster The number of nodes in the cluster is nine. All machines are homogeneous. Every machine has 16GB of RAM, 256GB HDD, and a 4 core CPU of 2.2GHz. The operating system is CentOS 6.4.

4.2. General results

Table III summarizes the query types, the number of runs for each query in the experiments, and the query samples. Average running time in seconds will be shown in the following tables and figures. Hadoop and the Extended SpatialHadoop emit the same query results and number of records. The terms SpatialHadoop and Extended SpatialHadoop will be used interchangeably in this section. This section shows the performance, in terms of time, of both the Extended SpatialHadoop and Hadoop using single-node and multi-node clusters. Table IV gives the average running time in seconds when processing the GDELT with a fixed 60GB size. In both SpatialHadoop and Hadoop, the multi-node cluster achieves much better performance than the single-node cluster in all queries. This comes as a result of the MapReduce distributed query processing. As shown in Table IV, SpatialHadoop is much faster than Hadoop in all queries whether on a single-node machine or a multi-node cluster.

While SpatialHadoop uses spatial indexing to process only partitions overlapped with the query MBR filter, Hadoop processes every record in all partitions sequentially which leads to performance degradation. The pruning function is the extra step that SpatialHadoop processes and the rest of the map and reduce functions are very similar. Indeed, this is illustrated in more detail in Table V as the number of MapReduce tasks is shown for the Multi-node cluster experiments. While Hadoop has a fixed large number of map tasks as it iterates over all records in the GDELT dataset, SpatialHadoop has fewer number of map tasks as a result of the pruning step. Point and Circle-Area queries are faster in SpatialHadoop than aggregation query as the latter query involves implementing the group by functionality to get the final counts which requires a reducer.

In Point and Circle-Area queries, few partitions (rectangles) are chosen as the query MBR is small (a point and a small radius). This results in a small number of Map tasks. This is why, as shown in Table IV, the reduction is just 2x when increasing machines by 9x (Single-node and Multi-node).

Table III. Query types.

Query Type	No. of runs	Query Sample (*Using Extended SpatialHadoop Jar file*)
Point Query	5	Select events for a given Point *pointquery input output* [*long*, *lat*]
Circle-Area Query	5	Select all events surrounding a Point with a given radius *circlequery input output* [*long*, *lat*, *radius*]
Aggregation Query	5	Select events aggregate count between two Actors given a specific region *countquery input output actor1 actor2* [*long1*, *lat1*, *long2*, *lat2*]

Table IV. Experiments on GDELT dataset.

	SpatialHadoop (sec)		Hadoop (sec)	
	Single-node	Multi-node	Single-node	Multi-node
Point Query	20.938	**11.744**	2459.310	280.226
Circle-Area Query	28.954	**12.747**	2471.257	288.716
Aggregation Query	344.660	**84.327**	2118.506	391.511

DOI: 10.1002/spe

Table V. Number of tasks for multi-node cluster experiments.

	SpatialHadoop		Hadoop	
	Map	Reduce	Map	Reduce
Point Query	3	0	980	0
Circle-Area Query	12	0	980	0
Aggregation Query	199	1	980	1

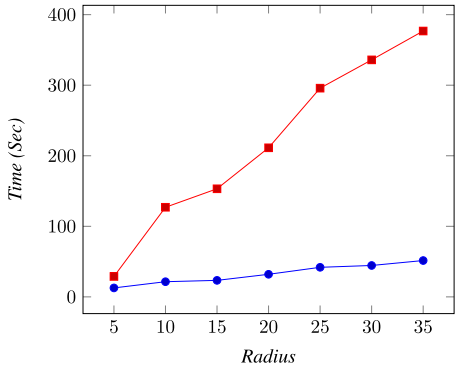

Figure 8. Extended SpatialHadoop circle-area query: *Single-Node*; *Multi-Node*.

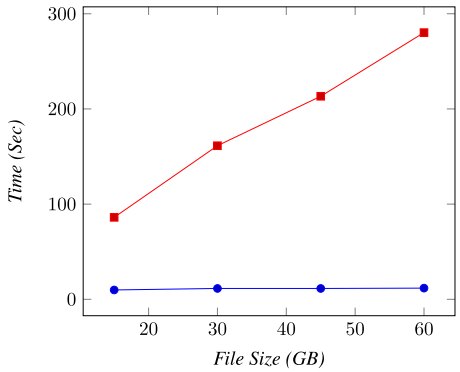

Figure 9. Point query: *Extended SpatialHadoop*; *Hadoop*.

The number of Map tasks are 3 and 12, as shown in Table V, for Point and Circle-Area queries, respectively. This is why there are 199 Map tasks in Aggregation query as the query MBR covers both China and Taiwan. The number of Map tasks is proportional to the size of the query MBR. This is illustrated in Figure 8 where Circle-Area queries have been executed considering different radii for both SpatialHadoop single-node and multi-node clusters. Here we can see clearly how the multi-node cluster outperforms the single-node cluster when increasing the radius (i.e., Query MBR). In Point and circle-area queries, there is no reducer required as the mappers emit the selected events as the final output. This means there is no reshuffling phase in these two queries. However, the aggregation query requires a reducer which involves an expensive reshuffling phase where data will be shipped over the network from mappers to the reducer. One reducer is sufficient as the number of distinct group by clauses which are the keys to the reducer is just a few hundred. Besides, as a result of passing China and Taiwan in the aggregation query parameters, the MBR area for this query is much bigger than the areas of the other queries. We discuss scalability in the coming subsections.

DOI: 10.1002/spe

4.3. Point query

In this part, we run the point query considering different sizes of the GDELT dataset to show scalability. In Figures 9, 10, and 11, the x-axis is the file size in GB and the y-axis is the running time in seconds. Again, as SpatialHadoop is aware of spatial data and uses spatial indexing to prune the data, it achieves a much better performance than Hadoop as shown in Figure 9. The results show that SpatialHadoop achieves up to 9×, 16×, 20×, and 30× better performance than Hadoop using 15GB, 30GB, 45GB, and 60GB, respectively. While Hadoop values increase linearly as data grow, SpatialHadoop values do not. Actually they increase but very slightly. The reason is that when pruning data, the number of selected partitions stays the same even if the size of the data grows. This lends itself to the fact that the same number of cells is used when indexing. The only thing that grows is the number of records in each of the selected partitions. Hence, as data size grows, the time increases unnoticeably (i.e., fraction of seconds). This lets us conclude that SpatialHadoop scales up very well with data growth.

4.4. Circle-area query

Circle-Area query is very similar to point query when it comes to discussing the results. As Figure 10 shows, the running time is very similar to that in Figure 9 with almost the same performance. SpatialHadoop achieves a better performance than Hadoop as well.

4.5. Aggregation query

Figure 11 demonstrates aggregation query processing differences between SpatialHadoop and Hadoop. As mentioned earlier, SpatialHadoop implements data pruning to select related partitions based on the global and local indexes built beforehand and thus this leads to a better performance

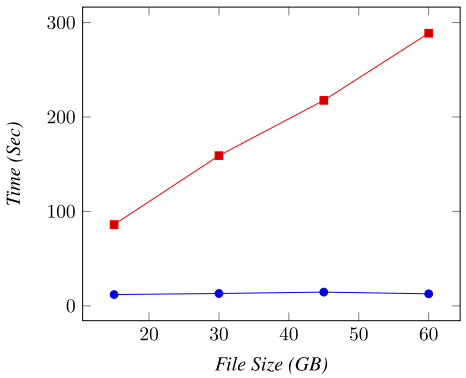

Figure 10. Circle-Area Query: *Extended SpatialHadoop*; *Hadoop*.

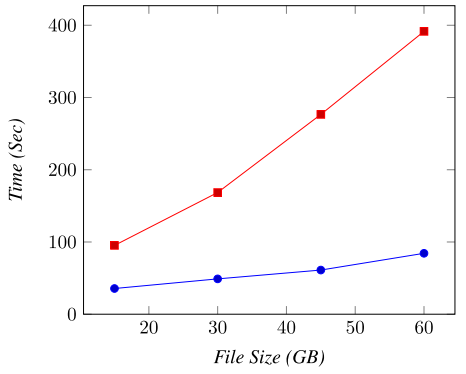

Figure 11. Aggregation Count Query: *Extended SpatialHadoop*; *Hadoop*.

DOI: 10.1002/spe

Table VI. Results of finding 2, 3, 4 co-occurring events.

Number of events in MBR M	242
Number of $< t_i, t_j >$	21796
Number of $< t_i, t_j, t_k >$	356461
Number of $< t_i, t_j, t_k, t_l >$	9347136
2 co-occurring events execution time (242 Circle Queries)	(GISQAF: 48 min) (Hadoop: 20 hours)
3 and 4 co-occurring events time (using B)	126 sec

than processing all records in all partitions. Different than point and circle-area queries that are illustrated in Figures 9 and 10, here SpatialHadoop shows a noticeable increase in the running time. Although the number of partitions selected by SpatialHadoop stays the same even when data grows, the running time increases and that can be seen clearly. The reason is that aggregation query needs more time for implementing the count and grouping results by the required fields.

4.6. Co-occurring events

In this section, we discuss the results of finding $2, 3, 4$ ($c = 4$) co-occurring events as an implementation to the example discussed in Section 3.5.2. As shown in Table VI, the number of geospatial events that are located in the MBR $M = (36.6723, 36.5979, 38.1665, 37.1078)$ is $m = 242$. The table shows the number of two co-occurring events $< t_i, t_j >$, three co-occurring events $< t_i, t_j, t_k >$, and four co-occurring events $< t_i, t_j, t_k, t_l >$. Each subset co-occurs in a radius $r = 0.09°$.

As mentioned in Section 4.4, to find all two co-occurring events, we execute the Circle Query against each of the $m = 242$ events. Each Circle Query takes 12 seconds to execute on average as shown in Table IV using SpatialHadoop on the multi-node cluster. Table VI shows the time needed to execute 242 Circle Queries which is less than an hour. Notice that Hadoop will take around 20 hours to execute these Circle Queries. The table shows also the running time for finding 3 and 4 co-occurring events from the Binary Matrix B.

5. RELATED WORK

Existing non-distributed traditional RDBMS applications for querying GDELT have been used for a while [1, 2]. In [29], a geospatial DBMS approach was used but lacked the ability for efficient spatial partitioning and boundary check. Besides, those applications experience scalability and performance issues.

A spatio-temporal indexing structure built on top of a column-family distributed solution has been suggested in [4] by Fox *et al.* They present a novel spatio-temporal geohashing index structure running on Apache Accumulo [30] which is a distributed key-value store based on Google's BigTable design. Apache Accumulo is built on top of Apache Hadoop, Zookeeper, and Thrift. However, the issue with this solution is that it does not support some types of queries like aggregation queries.

Some other scalable NoSQL based solutions such as GeoCouch [31] and neo4j-spatial [32] have implemented Spatial operations but they only consider limited queries with no support for data analytics.

Aji *et al.* presented Hadoop-GIS [33], a MapReduce spatial data warehousing system. Hadoop-GIS uses a two-level index structure, local and global. It supports various types of queries including aggregation queries. However, Hadoop-GIS focuses mainly on pathology analytical imaging of medical data.

SpatialHadoop is a MapReduce framework built on top of Hadoop for processing spatial datasets efficiently [19]. Similar to Hadoop-GIS, SpatialHadoop exploits a two-level index structure, local and global. Programs in SpatialHadoop run as in Hadoop MapReduce with the awareness of spatial operations. It is open source and available for researchers to use and enhance. However, SpatialHadoop considers simple datasets and does not deal with data analytics and some complex queries like Aggregation Queries.

DOI: 10.1002/spe

This paper presents GISQAF (Geographic Information System Query and Analytics Framework). The framework customizes and extends SpatialHadoop to index, decode, and query the GDELT georeferenced dataset and similar datasets. GISQAF extends the work in [20] which does not consider any approach to implement spatial Data Analytics. Therefore, this paper not only considers the Query Processing (QP), but it also considers the Data Analytics (DA). It addresses extracting geospatial co-occurring tuples (political spatial events) that happen in a particular geographical region (Minimum Bounding Rectangle) by considering a specific time window and a particular geographical radius.

6. CONCLUSION

Processing massive spatial data like the Global Data of Events, Language, and Tone (GDELT) dataset becomes an issue as traditional RDBMS can no longer be used. The MapReduce paradigm introduces alternatives to deal with big data. Unless the MapReduce framework is well-equipped to process spatial data, it will not achieve better performances.

In this paper, Geographic Information System Query and Analytics Framework (GISQAF) has been presented to process queries and implement data analytics for the GDELT dataset. The framework has been built on top of SpatialHadoop which is a Hadoop-extended framework to deal with spatial data. Experimental results using a single-node cluster and a multi-node nine-machine cluster show that GDELT query processing with GISQAF achieves up to 30x better performance than the traditional Hadoop query system.

Incorporating more query types into GISQAF is an avenue of future work. Another avenue is to make the preprocessing and spatial indexing phase of the queried dataset require less programming intervention. This will facilitate client interaction with the framework.

In the Data Analytics part, classification and clustering can be explored as well. In addition, more optimization techniques may be examined. Some infrastructure to perform such optimizations has been already implemented in GISQAF.

As the GISQAF framework runs in an offline batch processing fashion, it does not consider online streaming preprocessing and spatial indexing. Incorporating GISQAF into a big data streaming tool is a promising future work. This will help in online scenarios as the GDELT dataset is generated on a daily basis. This will require the use of incremental spatial indexing to accommodate the new generated data without the need to redo the indexing from scratch.

Apache Spark [34] is an open-source streaming distributed framework that utilizes Hadoop for its distributed file system. Unlike Hadoop, Spark utilizes in-memory cluster computing [35] which makes it up to 100 times faster than Hadoop MapReduce. Not only does implementing GISQAF over Spark help in online spatial indexing and query processing, but also with the data analytics part as Spark incorporates data mining algorithms as well. Spark has been shown to be very efficient for real-time data analytics [36, 37] where the authors use statistical and clustering techniques for online anomaly detection. Furthermore, Spark SQL has been implemented in [38] for easier querying. Incorporating GISQAF into Spark will increase the efficiency of the framework tremendously.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Award No. CNS 1229652 and the Air Force Office of Scientific Research under Award No. FA-9550-09-1-0468. We thank Dr. Robert Herklotz for his support.

REFERENCES

1. *GDELT Event Database*. Available at: http://gdeltproject.org/ [last accessed: 13 December 2014].
2. Leetaru K, Schrodt P. Gdelt: Global data on events, location and tone. *Proceedings of the International Studies Association Annual Conference*, San Diego, CA, 2013.
3. Schrodt PA, Yilmaz Ö, Gerner DJ, Hermrick D, Bron A, Gregory A, Ingram A, Jekic M, Mcmullen L, Prather L, Price T. Coding sub-state actors using the cameo (conflict and mediation event observations) actor coding framework. *Annual Meeting of the International Studies Association*, San Francisco, CA, USA, 2008.

DOI: 10.1002/spe

4. Fox A, Eichelberger C, Hughes J, Lyon S. Spatio-temporal indexing in non-relational distributed databases. *Bigdata Conference*, IEEE, Santa Clara, CA, USA, 2013; 291–299.
5. Castiglione A, Gribaudo M, Iacono M, Palmieri F. Modeling performances of concurrent big data applications. *Software: Practice and Experience* 2015; **45**(8):1127–1144.
6. Ismail L, Khan L. Implementation and performance evaluation of a scheduling algorithm for divisible load parallel applications in a cloud computing environment. *Software: Practice and Experience* 2015; **45**:765–781. DOI: 10.1002/spe.2258.
7. Apache Hadoop. Available at: http://hadoop.apache.org/ [Last accessed: 13 December 2014].
8. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE. Bigtable: A distributed storage system for structured data. *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, Seattle, WA, 2006; 15–15.
9. Lakshman A, Malik P. Cassandra: A decentralized structured storage system. *SIGOPS - Operating Systems Review* 2010; **44**(2):35–40.
10. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Antony S, Liu H, Murthy R. Hive - a petabyte scale data warehouse using hadoop. *IEEE 26th International Conference on Data Engineering (ICDE)*, Long Beach, California, USA, 2010; 996–1005.
11. Dean J, Ghemawat S. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM* 2008; **51**(1):107–113.
12. Liao CS, Chuang CP, Chang RS. A novel monitoring mechanism by event trigger for hadoop system performance analysis. *Software - Practice and Experience* 2014; **44**(7):823–834. DOI: 10.1002/spe.2230.
13. Tan YS, Tan J, Chng ES, Lee B-S, Li J, Date S, Chak HP, Xiao X, Narishige A. Hadoop framework: impact of data organization on performance. *Software: Practice and Experience* 2013; **43**(11):1241–1260. DOI: 10.1002/spe.1082.
14. Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proceedings of the VLDB Endowment* 2009; **2**(1): 922–933.
15. Babu S. Towards automatic optimization of mapreduce programs. *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, ACM, New York, NY, USA, 2010; 137–142.
16. Lee K-H, Lee Y-J, Choi H, Chung YD, Moon B. Parallel data processing with mapreduce: A survey. *SIGMOD Record* 2012; **40**(4):11–20.
17. Nievergelt J, Hinterberger H, Sevcik KC. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems* 1984; **9**(1):38–71.
18. Guttman A. R-trees: A dynamic index structure for spatial searching. *SIGMOD Record* 1984; **14**(2):47–57.
19. Eldawy A, Mokbel MF. A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. *Proceedings of the VLDB Endowment* 2013; **6**(12):1230–1233.
20. Al-Naami K, Seker S, Khan L. Gisqf: An efficient spatial query processing system. *2014 IEEE 7th International Conference on Cloud Computing*, Alaska, USA, 2014; 681–688.
21. Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, Washington DC (USA, 1993; 207–216.
22. Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994; 487–499.
23. Mennis J, Liu JW. Mining association rules in spatio-temporal data: An analysis of urban socioeconomic and land cover change. *Transactions in GIS* 2005; **9**:5–17. DOI: 10.1111/j.1467-9671.2005.00202.x.
24. SpatialHadoop. Available at: http://spatialhadoop.cs.umn.edu/ [Last accessed: 13 December 2014].
25. Sellis T, Roussopoulos N, Faloutsos C. The r+-tree: A dynamic index for multi-dimensional objects. *Proceedings of International Conference on Very Large Databases*, Brighton, England, 1987; 507–518.
26. Dittrich J-P, Seeger B. Data redundancy and duplicate detection in spatial join processing. *16th International Conference on Data Engineering, 2000. Proceedings*, San Diego, CA, USA, 2000; 535–546.
27. Eldawy A, Li Y, Mokbel MF, Janardan R. Cg-hadoop: Computational geometry in mapreduce. *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, ACM, New York, NY, USA, 2013; 294–303.
28. Montavont J, Noel T. IEEE 802.11 handovers assisted by gps information. *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2006. (WIMOB'2006)*, 2006; 166–172.
29. Patel J, Yu J, Kabra N, Tufte K, Nag B, Burger J, Hall N, Ramasamy K, Lueder R, Ellmann C, Kupsch J, Guo S, Larson J, De Witt D, Naughton J. Building a scaleable geo-spatial dbms: Technology, implementation, and evaluation. *SIGMOD Record* 1997; **26**(2):336–347.
30. Apache Accumulo. Available at: https://accumulo.apache.org/ [Last accessed: 13 December 2014].
31. Geocouch. Available at: https://github.com/couchbase/geocouch/ [Last accessed: 13 December 2014].
32. Neo4j Spatial. Available at: https://github.com/neo4j-contrib/spatial [Last accessed: 13 December 2014].
33. Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J. Hadoop-gis: A high performance spatial data warehousing system over mapreduce. *Proceedings VLDB Endowment*, Trento, Italy, 2013. VLDB Endowment 2150-8097/13/09.
34. Apache Spark. Available at: http://spark.apache.org// [Last accessed: 28 September 2015].
35. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauly M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *Presented as Part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, San Jose, CA, 2012; 15–28.

36. Solaimani M, Iftekhar M, Khan L, Thuraisingham BM. Statistical technique for online anomaly detection using spark over heterogeneous data from multi-source vmware performance data. *2014 IEEE International Conference on Big Data, Big Data 2014*, Washington, DC, USA, October 27-30, 2014; 1086–1094.
37. Solaimani M, Iftekhar M, Khan L, Thuraisingham BM, Ingram Joey Burton. Spark-based anomaly detection over multi-source vmware performance data in real-time. *2014 IEEE Symposium on Computational Intelligence in Cyber Security*, CICS 2014, Orlando, FL, USA, December 9-12, 2014; 66–73.
38. Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A, Zaharia M. Spark sql: Relational data processing in spark. *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, 2015; 1383–1394.

DOI: 10.1002/spe