

ST-Hadoop: A MapReduce Framework for Spatio-Temporal Data

Louai Alarabi^(✉), Mohamed F. Mokbel^(✉), and Mashaal Musleh

Department of Computer Science and Engineering,
University of Minnesota, Minneapolis, MN, USA
{louai,mokbel,musle005}@cs.umn.edu

Abstract. This paper presents ST-Hadoop; the first full-fledged open-source MapReduce framework with a native support for spatio-temporal data. ST-Hadoop is a comprehensive extension to Hadoop and Spatial-Hadoop that injects spatio-temporal data awareness inside each of their layers, mainly, language, indexing, and operations layers. In the language layer, ST-Hadoop provides built in spatio-temporal data types and operations. In the indexing layer, ST-Hadoop spatiotemporally loads and divides data across computation nodes in Hadoop Distributed File System in a way that mimics spatio-temporal index structures, which result in achieving orders of magnitude better performance than Hadoop and SpatialHadoop when dealing with spatio-temporal data and queries. In the operations layer, ST-Hadoop shipped with support for two fundamental spatio-temporal queries, namely, spatio-temporal range and join queries. Extensibility of ST-Hadoop allows others to expand features and operations easily using similar approach described in the paper. Extensive experiments conducted on large-scale dataset of size 10 TB that contains over 1 Billion spatio-temporal records, to show that ST-Hadoop achieves orders of magnitude better performance than Hadoop and SpaitalHadoop when dealing with spatio-temporal data and operations. The key idea behind the performance gained in ST-Hadoop is its ability in indexing spatio-temporal data within Hadoop Distributed File System.

1 Introduction

The importance of processing spatio-temporal data has gained much interest in the last few years, especially with the emergence and popularity of applications that create them in large-scale. For example, Taxi trajectory of New York city archive over 1.1 Billion trajectories [1], social network data (e.g., Twitter has over 500 Million new tweets every day) [2], NASA Satellite daily produces 4 TB of data [3,4], and European X-Ray Free-Electron Laser Facility produce large collection of spatio-temporal series at a rate of 40 GB per second, that collectively

This work is partially supported by the National Science Foundation, USA, under Grants IIS-1525953, CNS-1512877, IIS-1218168, and by a scholarship from the College of Computers & Information Systems, Umm Al-Qura University, Makkah, Saudi Arabia.

```

Objects = LOAD 'points' AS (id:int, Location:POINT, Time:t);
Result  = FILTER Objects BY
          Overlaps (Location, Rectangle(x1, y1, x2, y2))
          AND t < t2 AND t > t1;

```

(a) Range query in SpatialHadoop

```

Objects = LOAD 'points' AS (id:int, STPoint:(Location,Time));
Result  = FILTER Objects BY
          Overlaps (STPoint, Rectangle(x1, y1, x2, y2), Interval (t1, t2) );

```

(b) Range query in ST-Hadoop

Fig. 1. Range query in SpatialHadoop vs. ST-Hadoop

form 50 PB of data yearly [5]. Beside the huge achieved volume of the data, space and time are two fundamental characteristics that raise the demand for processing spatio-temporal data.

The current efforts to process big spatio-temporal data on MapReduce environment either use: (a) *General purpose* distributed frameworks such as Hadoop [6] or Spark [7], or (b) *Big spatial data systems* such as ESRI tools on Hadoop [8], Parallel-Secondo [9], *MD*-HBase [10], Hadoop-GIS [11], GeoTrellis [12], GeoSpark [13], or SpatialHadoop [14]. The former has been acceptable for typical analysis tasks as they organize data as non-indexed heap files. However, using these systems as-is will result in sub-performance for spatio-temporal applications that need indexing [15–17]. The latter reveal their inefficiency for supporting time-varying of spatial objects because their indexes are mainly geared toward processing spatial queries, e.g., SHAHED system [18] is built on top of SpatialHadoop [14].

Even though existing big spatial systems are efficient for spatial operations, nonetheless, they suffer when they are processing spatio-temporal queries, e.g., *find geo-tagged news in California area during the last three months*. Adopting any big spatial systems to execute common types of spatio-temporal queries, e.g., *range query*, will suffer from the following: (1) The spatial index is still ill-suited to efficiently support time-varying of spatial objects, mainly because the index are geared toward supporting spatial queries, in which result in scanning through irrelevant data to the query answer. (2) The system internal is unaware of the spatio-temporal properties of the objects, especially when they are routinely achieved in large-scale. Such aspect enforces the spatial index to be reconstructed from scratch with every batch update to accommodate new data, and thus the space division of regions in the spatial-index will be jammed, in which require more processing time for spatio-temporal queries. One possible way to recognize spatio-temporal data is to add one more dimension to the spatial index. Yet, such choice is incapable of accommodating new batch update without reconstruction.

This paper introduces ST-Hadoop; the first full-fledged open-source MapReduce framework with a native support for spatio-temporal data, available to download from [19]. ST-Hadoop is a comprehensive extension to Hadoop and

SpatialHadoop that injects spatio-temporal data awareness inside each of their layers, mainly, indexing, operations, and language layers. ST-Hadoop is compatible with SpatialHadoop and Hadoop, where programs are coded as *map* and *reduce* functions. However, running a program that deals with spatio-temporal data using ST-Hadoop will have orders of magnitude better performance than Hadoop and SpatialHadoop. Figures 1(a) and (b) show how to express a spatio-temporal range query in SpatialHadoop and ST-Hadoop, respectively. The query finds all points within a certain rectangular area represented by two corner points $\langle x1, y1 \rangle, \langle x2, y2 \rangle$, and a within a time interval $\langle t1, t2 \rangle$. Running this query on a dataset of 10 TB and a cluster of 24 nodes takes 200s on SpatialHadoop as opposed to only one second on ST-Hadoop. The main reason of the sub-performance of SpatialHadoop is that it needs to scan all the entries in its spatial index that overlap with the spatial predicate, and then check the temporal predicate of each entry individually. Meanwhile, ST-Hadoop exploits its built-in spatio-temporal index to only retrieve the data entries that overlap with *both* the spatial and temporal predicates, and hence achieves two orders of magnitude improvement over SpatialHadoop.

ST-Hadoop is a comprehensive extension of Hadoop that injects spatio-temporal awareness inside each layers of SpatialHadoop, mainly, *language*, *indexing*, *MapReduce*, and *operations* layers. In the *language* layer, ST-Hadoop extends Pigeon language [20] to supports spatio-temporal data types and operations. The *indexing* layer, ST-Hadoop spatiotemporally loads and divides data across computation nodes in the Hadoop distributed file system. In this layer ST-Hadoop scans a random sample obtained from the whole dataset, bulk loads its spatio-temporal index in-memory, and then uses the spatio-temporal boundaries of its index structure to assign data records with its overlap partitions. ST-Hadoop sacrifices storage to achieve more efficient performance in supporting spatio-temporal operations, by replicating its index into *temporal hierarchy index structure* that consists of two-layer indexing of temporal and then spatial. The *MapReduce* layer introduces two new components of *SpatioTemporalFileSplitter*, and *SpatioTemporalRecordReader*, that exploit the spatio-temporal index structures to speed up spatio-temporal operations. Finally, the *operations* layer encapsulates the spatio-temporal operations that take advantage of the ST-Hadoop *temporal hierarchy index structure* in the indexing layer, such as spatio-temporal range and join queries.

The key idea behind the performance gain of ST-Hadoop is its ability to load the data in Hadoop Distributed File System (HDFS) in a way that mimics spatio-temporal index structures. Hence, incoming spatio-temporal queries can have minimal data access to retrieve the query answer. ST-Hadoop is shipped with support for two fundamental spatio-temporal queries, namely, spatio-temporal range and join queries. However, ST-Hadoop is extensible to support a myriad of other spatio-temporal operations. We envision that ST-Hadoop will act as a research vehicle where developers, practitioners, and researchers worldwide, can either use it directly or enrich the system by contributing their operations and analysis techniques.

The rest of this paper is organized as follows: Sect. 2 highlights related work. Section 3 gives the architecture of ST-Hadoop. Details of the *language*, *spatio-temporal indexing*, and *operations* are given in Sects. 4, 5 and 6, followed by extensive experiments conducted in Sect. 7. Section 8 concludes the paper.

2 Related Work

Triggered by the needs to process large-scale spatio-temporal data, there is an increasing recent interest in using Hadoop to support spatio-temporal operations. The existing work in this area can be classified and described briefly as following:

On-Top of MapReduce Framework. Existing work in this category has mainly focused on addressing a specific spatio-temporal operation. The idea is to develop map and reduce functions for the required operation, which will be executed on-top of existing Hadoop cluster. Examples of these operations includes spatio-temporal range query [15–17], spatio-temporal join [21–23]. However, using Hadoop as-is results in a poor performance for spatio-temporal applications that need indexing.

Ad-hoc on Big Spatial System. Several big spatial systems in this category are still ill-suited to perform spatio-temporal operations, mainly because their indexes are only geared toward processing spatial operations, and their internals are unaware of the spatio-temporal data properties [8–11, 13, 14, 24–27]. For example, SHAHED runs spatio-temporal operations as an ad-hoc using Spatial-Hadoop [14].

Spatio-Temporal System. Existing works in this category has mainly focused on combining the three spatio-temporal dimensions (i.e., x, y, and time) into a single-dimensional lexicographic key. For example, GeoMesa [28] and GeoWave [29] both are built upon Accumulo platform [30] and implemented a space filling curve to combine the three dimensions of geometry and time. Yet, these systems do not attempt to enhance the spatial locality of data; instead they rely on time load balancing inherited by Accumulo. Hence, they will have a sup-performance for spatio-temporal operations on highly skewed data.

ST-Hadoop is designed as a generic MapReduce system to support spatio-temporal queries, and assist developers in implementing a wide selection of spatio-temporal operations. In particular, ST-Hadoop leverages the design of Hadoop and SpatialHadoop to loads and partitions data records according to their time and spatial dimension across computations nodes, which allow the parallelism of processing spatio-temporal queries when accessing its index. In this paper, we present two case study of operations that utilize the ST-Hadoop indexing, namely, spatio-temporal range and join queries. ST-Hadoop operations achieve two or more orders of magnitude better performance, mainly because ST-Hadoop is sufficiently aware of both temporal and spatial locality of data records.

3 ST-Hadoop Architecture

Figure 2 gives the high level architecture of our ST-Hadoop system; as the first full-fledged open-source MapReduce framework with a built-in support for spatio-temporal data. ST-Hadoop cluster contains one master node that breaks a map-reduce job into smaller tasks, carried out by slave nodes. Three types of users interact with ST-Hadoop: (1) *Casual users* who access ST-Hadoop through its spatio-temporal language to process their datasets. (2) *Developers*, who have a deeper understanding of the system internals and can implement new spatio-temporal operations, and (3) *Administrators*, who can tune up the system through adjusting system parameters in the configuration files provided with the ST-Hadoop installation. ST-Hadoop adopts a layered design of four main layers, namely, *language*, *Indexing*, *MapReduce*, and *operations* layers, described briefly below:

Language Layer: This layer extends Pigeon language [20] to supports spatio-temporal data types (i.e., STPOINT, TIME and INTERVAL) and spatio-temporal operations (e.g., OVERLAP, and JOIN). Details are given in Sect. 4.

Indexing Layer: ST-Hadoop spatiotemporally loads and partitions data across computation nodes. In this layer ST-Hadoop scans a random sample obtained from the input dataset, bulk-loads its spatio-temporal index that consists of two-layer indexing of temporal and then spatial. Finally ST-Hadoop replicates its index into *temporal hierarchy index structure* to achieve more efficient

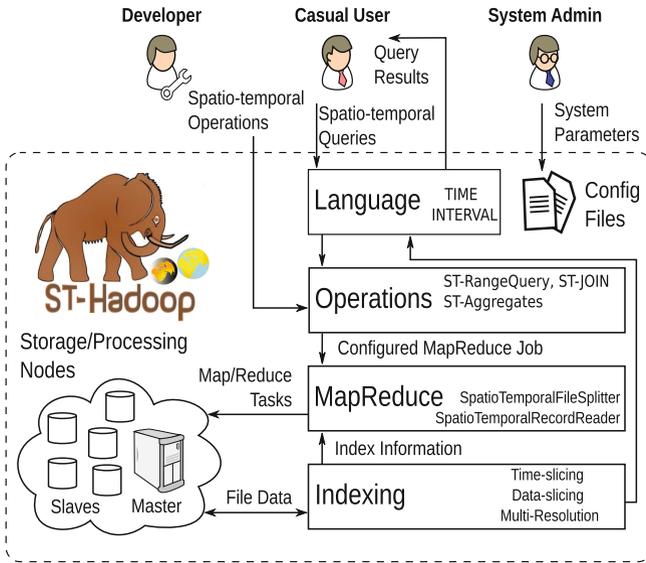


Fig. 2. ST-Hadoop system architecture

performance for processing spatio-temporal queries. Details of the index layer are given in Sect. 5.

MapReduce Layer: In this layer, new implementations added inside Spatial-Hadoop MapReduce layer to enables ST-Hadoop to exploits its spatio-temporal indexes and realizes spatio-temporal predicates. We are not going to discuss this layer any further, mainly because few changes were made to inject time awareness in this layer. The implementation of MapReduce layer was already discussed in great details [14].

Operations Layer: This layer encapsulates the implementation of two common spatio-temporal operations, namely, spatio-temporal range, and spatio-temporal join queries. More operations can be added to this layer by ST-Hadoop *developers*. Details of the operations layer are discussed in Sect. 6.

4 Language Layer

ST-Hadoop does not provide a completely new language. Instead, it extends Pigeon language [20] by adding spatio-temporal data types, functions, and operations. Spatio-temporal data types (STPoint, Time and Interval) are used to define the schema of input files upon their loading process. In particular, ST-Hadoop adds the following:

Data types. ST-Hadoop extends STPoint, TIME, and INTERVAL. The TIME instance is used to identify the temporal dimension of the data, while the time INTERVAL mainly provided to equip the query predicates. The following code snippet loads NYC taxi trajectories from ‘NYC’ file with a column of type STPoint.

```
trajectory = LOAD ‘NYC’ as
(id:int, STPoint(loc:point, time:timestamp));
```

NYC and trajectory are the paths to the non-indexed heap file and the destination indexed file, respectively. loc and time are the columns that specify both spatial and temporal attributes.

Functions and Operations. Pigeon already equipped with several basic spatial predicates. ST-Hadoop changes the `overlap` function to support spatio-temporal operations. The other predicates and their possible variation for supporting spatio-temporal data are discussed in great details in [31]. ST-Hadoop encapsulates the implementation of two commonly used spatio-temporal operations, i.e., range and Join queries, that take the advantages of the spatio-temporal index. The following example “*retrieves all cars in State Fair area represented by its minimum boundary rectangle during the time interval of August 25th and September 6th*” from trajectory indexed file.

```
cars = FILTER trajectory
BY overlap( STPoint,
RECTANGLE(x1,y1,x2,y2),
INTERVAL(08-25-2016, 09-6-2016));
```

ST-Hadoop extended the JOIN to take two spatio-temporal indexes as an input. The processing of the `join` invokes the corresponding spatio-temporal procedure. For example, one might need to understand the relationship between the birds death and the existence of humans around them, which can be described as “*find every pairs from birds and human trajectories that are close to each other within a distance of 1 mile during the last year*”.

```
human_bird_pairs = JOIN human_trajectory, bird_trajectory
  PREDICATE = overlap( RECTANGLE(x1,y1,x2,y2),
    INTERVAL(01-01-2016, 12-31-2016),
    WITHIN_DISTANCE(1) );
```

5 Indexing Layer

Input files in Hadoop Distributed File System (HDFS) are organized as a heap structure, where the input is partitioned into chunks, each of size 64 MB. Given a file, the first 64 MB is loaded to one partition, then the second 64 MB is loaded in a second partition, and so on. While that was acceptable for typical Hadoop applications (e.g., analysis tasks), it will not support spatio-temporal applications where there is always a need to filter input data with spatial and temporal predicates. Meanwhile, spatially indexed HDFSs, as in SpatialHadoop [14] and ScalaGiST [27], are geared towards queries with spatial predicates only. This means that a temporal query to these systems will need to scan the whole dataset. Also, a spatio-temporal query with a small temporal predicate may end up scanning large amounts of data. For example, consider an input file that includes all social media contents in the whole world for the last five years or so. A query that asks about contents in the USA in a certain hour may end up in scanning all the five years contents of USA to find out the answer.

ST-Hadoop HDFS organizes input files as spatio-temporal partitions that satisfy one main goal of supporting spatio-temporal queries. ST-Hadoop imposes temporal slicing, where input files are spatiotemporally loaded into intervals of a specific time granularity, e.g., days, weeks, or months. Each granularity is represented as a level in ST-Hadoop index. Data records in each level are spatiotemporally partitioned, such that the boundary of a partition is defined by a spatial region and time interval.

Figures 3(a) and (b) show the HDFS organization in SpatialHadoop and ST-Hadoop frameworks, respectively. Rectangular shapes represent boundaries of the HDFS partitions within their framework, where each partition maintains a 64 MB of nearby objects. The dotted square is an example of a spatio-temporal range query. For simplicity, let’s consider a one year of spatio-temporal records loaded to both frameworks. As shown in Fig. 3(a), SpatialHadoop is unaware of the temporal locality of the data, and thus, all records will be loaded once and partitioned according to their existence in the space. Meanwhile in Fig. 3(b), ST-Hadoop loads and partitions data records for each day of the year individually, such that each partition maintains a 64 MB of objects that are close to each other

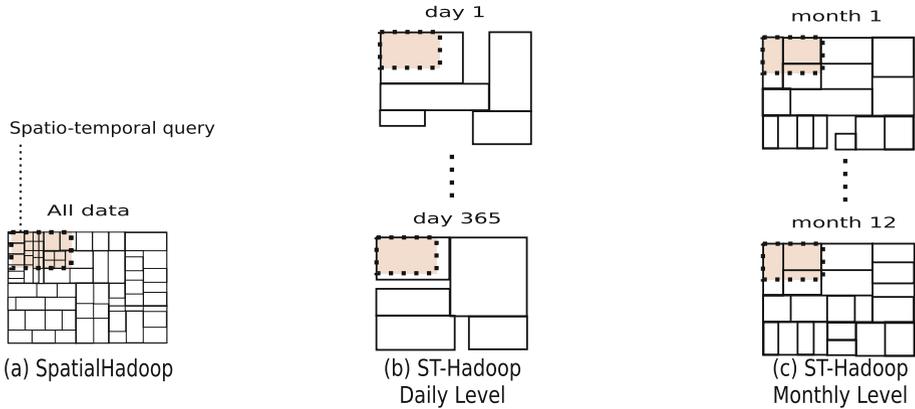


Fig. 3. HDFSS in ST-Hadoop vs. SpatialHadoop

in both space and time. Note that HDFS partitions in both frameworks vary in their boundaries, mainly because spatial and temporal locality of objects are not the same over time. Let's assume the spatio-temporal query in the dotted square “find objects in a certain spatial region during a specific month” in Figs. 3(a), and (b). SpatialHadoop needs to access all partitions overlapped with query region, and hence SpatialHadoop is required to scan one year of records to get the final answer. In the meantime, ST-Hadoop reports the query answer by accessing few partitions from its daily level without the need to scan a huge number of records.

5.1 Concept of Hierarchy

ST-Hadoop imposes a replication of data to support spatio-temporal queries with different granularities. The data replication is reasonable as the storage in ST-Hadoop cluster is inexpensive, and thus, sacrificing storage to gain more efficient performance is not a drawback. Updates are not a problem with replication, mainly because ST-Hadoop extends MapReduce framework that is essentially designed for batch processing, thereby ST-Hadoop utilizes incremental batch accommodation for new updates.

The key idea behind the performance gain of ST-Hadoop is its ability to load the data in Hadoop Distributed File System (HDFS) in a way that mimics spatio-temporal index structures. To support all spatio-temporal operations including more sophisticated queries over time, ST-Hadoop replicates spatio-temporal data into a *Temporal Hierarchy Index*. Figures 3(b) and (c) depict two levels of days and months in ST-Hadoop index structure. The same data is replicated on both levels, but with different spatio-temporal granularities. For example, a spatio-temporal query asks for objects in one month could be reported from any level in ST-Hadoop index. However, rather than hitting 30 days' partitions from the daily-level, it will be much faster to access less number of partitions by obtaining the answer from one month in the monthly-level.

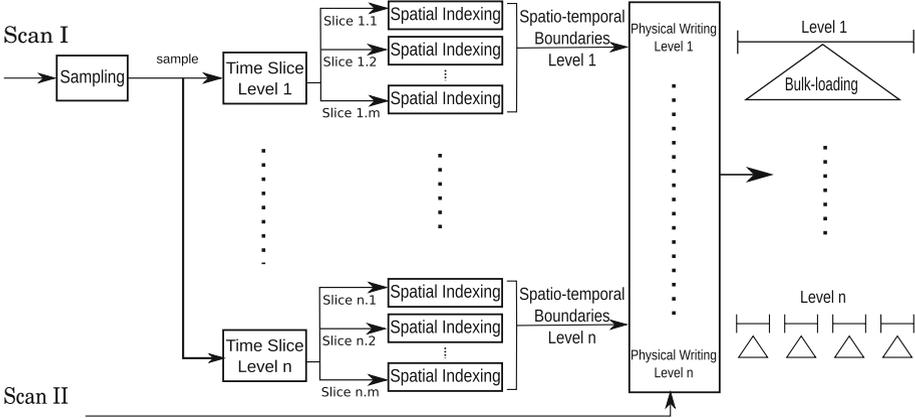


Fig. 4. Indexing in ST-Hadoop

A system parameter can be tuned by ST-Hadoop administrator to choose the number of levels in the *Temporal Hierarchy index*. By default, ST-Hadoop set its index structure to four levels of days, weeks, months and years granularities. However, ST-Hadoop users can easily change the granularity of any level. For example, the following code loads taxi trajectory dataset from “NYC” file using one-hour granularity, Where the **Level** and **Granularity** are two parameters that indicate which level and the desired granularity, respectively.

```
trajectory = LOAD 'NYC' as
    (id:int, STPoint(loc:point, time:timestamp))
    Level:1 Granularity:1-hour;
```

5.2 Index Construction

Figure 4 illustrates the indexing construction in ST-Hadoop, which involves two scanning processes. The first process starts by scanning input files to get a random sample, and this is essential because the size of input files is beyond memory capacity, and thus, ST-Hadoop obtains a set of records to a sample that can fit in memory. Next, ST-Hadoop processes the sample n times, where n is the number of levels in ST-Hadoop index structure. The temporal slicing in each level splits the sample into m number of slice (e.g., $slice_{1.m}$). ST-Hadoop finds the spatio-temporal boundaries by applying a spatial indexing on each temporal slice individually. As a result, outputs from temporal slicing and spatial indexing collectively represent the spatio-temporal boundaries of ST-Hadoop index structure. These boundaries will be stored as meta-data on the master node to guide the next process. The second scanning process physically assigns data records in the input files with its overlapping spatio-temporal boundaries. Note that each record in the dataset will be assigned n times, according to the number of levels.

ST-Hadoop index consists of two-layer indexing of a temporal and spatial. The conceptual visualization of the index is shown in the right of Fig. 4, where lines signify how the temporal index divided the sample into a set of disjoint time intervals, and triangles symbolize the spatial indexing. This two-layer indexing is replicated in all levels, where in each level the sample is partitioned using different granularity. ST-Hadoop trade-off storage to achieve more efficient performance through its index replication. In general, the index creation of a single level in the *Temporal Hierarchy* goes through four consecutive phases, namely sampling, temporal slicing, spatial indexing, and physical writing.

5.3 Phase I: Sampling

The objective of this phase is to approximate the spatial distribution of objects and how that distribution evolves over time, to ensure the quality of indexing; and thus, enhance the query performance. This phase is necessary, mainly because the input files are too large to fit in memory. ST-Hadoop employs a map-reduce job to efficiently read a sample through scanning all data records. We fit the sample into an in-memory simple data structure of a length (L), that is an equal to the number of HDFS blocks, which can be directly calculated from the equation $L = (Z/B)$, where Z is the total size of input files, and B is the HDFS block capacity (e.g., 64MB). The size of the random sample is set to a default ratio of 1% of input files, with a maximum size that fits in the memory of the master node. This simple data structure represented as a collection of elements; each element consist of a time instance and a space sampling that describe the time interval and the spatial distribution of spatio-temporal objects, respectively. Once the sample is scanned, we sort the sample elements in chronological order to their time instance, and thus the sample approximates the spatio-temporal distribution of input files.

5.4 Phase II: Temporal Slicing

In this phase ST-Hadoop determines the temporal boundaries by slicing the in-memory sample into multiple time intervals, to efficiently support a fast random access to a sequence of objects bounded by the same time interval. ST-Hadoop employs two temporal slicing techniques, where each manipulates the sample according to specific slicing characteristics: (1) *Time-partition*, slices the sample into multiple splits that are uniformly on their time intervals, and (2) *Data-partition* where the sample is sliced to the degree that all sub-splits are uniformly in their data size. The output of this phase finds the temporal boundary of each split, that collectively cover the whole time domain.

The rational reason behind ST-Hadoop two temporal slicing techniques is that for some spatio-temporal archive the data spans a long time-interval such as decades, but their size is moderated compared to other archives that are daily collect terabytes or petabytes of spatio-temporal records. ST-Hadoop proposed the two techniques to slice the time dimension of input files based on either time-partition or data-partition, to improve the indexing quality, and thus gain

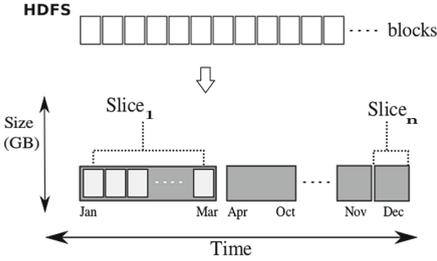


Fig. 5. Data-Slice

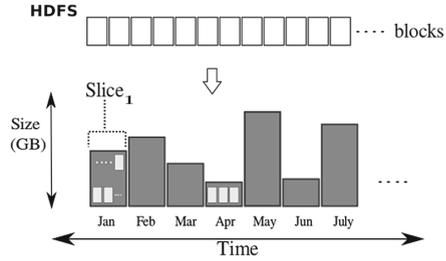


Fig. 6. Time-Slice

efficient query performance. The time-partition slicing technique serves best in a situation where data records are uniformly distributed in time. Meanwhile, data-partition slicing best suited with data that are sparse in their time dimension.

- Data-partition Slicing.* The goal of this approach is to slice the sample to the degree that all sub-splits are equally in their size. Figure 5 depicts the key concept of this slicing technique, such that a slice_1 and slice_n are equally in size, while they differ in their interval coverage. In particular, the temporal boundary of slice_1 spans more time interval than slice_n . For example, consider 128 MB as the size of HDFS block and input files of 1 TB. Typically, the data will be loaded into 8 thousand blocks. To load these blocks into ten equally balanced slices, ST-Hadoop first reads a sample, then sort the sample, and apply **Data-partition** technique that slices data into multiple splits. Each split contains around 800 blocks, which hold roughly a 100 GB of spatio-temporal records. There might be a small variance in size between slices, which is expectable. Similarly, another level in ST-Hadoop temporal hierarchy index could loads the 1 TB into 20 equally balanced slices, where each slice contains around 400 HDFS blocks. ST-Hadoop users are allowed to specify the granularity of data slicing by tuning α parameter. By default four ratios of α is set to 1%, 10%, 25%, and 50% that create the four levels in ST-Hadoop index structure.
- Time-partition Slicing.* The ultimate goal of this approach is to slices the input files into multiple HDFS chunks with a specified interval. Figure 6 shows the general idea, where ST-Hadoop splits the input files into an interval of one-month granularity. While the time interval of the slices is fixed, the size of data within slices might vary. For example, as shown in Fig. 6 Jan slice has more HDFS blocks than April.

ST-Hadoop users are allowed to specify the granularity of this slicing technique, which specified the time boundaries of all splits. By default, ST-Hadoop finer granularity level is set to one-day. Since the granularity of the slicing is known, then a straightforward solution is to find the minimum and maximum time instance of the sample, and then based on the intervals between the both times ST-Hadoop hashes elements in the sample to the desired granularity.

The number of slices generated by the time-partition technique will highly depend on the intervals between the minimum and the maximum times obtained from the sample. By default, ST-Hadoop set its index structure to four levels of days, weeks, months and years granularities.

5.5 Phase III: Spatial Indexing

This phase ST-Hadoop determines the spatial boundaries of the data records within each temporal slice. ST-Hadoop spatially index each temporal slice independently; such decision handles a case where there is a significant disparity in the spatial distribution between slices, and also to preserve the spatial locality of data records. Using the same sample from the previous phase, ST-Hadoop takes the advantages of applying different types of spatial bulk loading techniques in HDFS that are already implemented in SpatialHadoop such as Grid, R-tree, Quad-tree, and Kd-tree. The output of this phase is the spatio-temporal boundaries of each temporal slice. These boundaries stored as a meta-data in a file on the master node of ST-Hadoop cluster. Each entry in the meta-data represents a partition, such as $\langle id, MBR, interval, level \rangle$. Where id is a unique identifier number of a partition on the HDFS, MBR is the spatial minimum boundary rectangle, $interval$ is the time boundary, and the level is the number that indicates which level in ST-Hadoop temporal hierarchy index.

5.6 Phase IV: Physical Writing

Given the spatio-temporal boundaries that represent all HDFS partitions, we initiate a map-reduce job that scans through the input files and physically partitions HDFS block, by assign data records to overlapping partitions according to the spatio-temporal boundaries in the meta-data stored on the master node of ST-Hadoop cluster. For each record r assigned to a partition p , the map function writes an intermediate pair $\langle p, r \rangle$. Such pairs are then grouped by p and sent to the reduce function to write the physical partition to the HDFS. Note that for a record r will be assigned n times, depends on the number of levels in ST-Hadoop index.

6 Operations Layer

The combination of the spatiotemporally load balancing with the temporal hierarchy index structure gives the core of ST-Hadoop, that enables the possibility of efficient and practical realization of spatio-temporal operations, and hence provides orders of magnitude better performance over Hadoop and SpatialHadoop. In this section, we only focus on two fundamental spatio-temporal operations, namely, range (Sect. 6.1) and join queries (Sects. 6.2), as case studies of how to exploit the spatio-temporal indexing in ST-Hadoop. Other operations can also be realized following a similar approach.

6.1 Spatio-Temporal Range Query

A range query is specified by two predicates of a spatial area and a temporal interval, A and T , respectively. The query finds a set of records R that overlap with both a region A and a time interval T , such as “*finding geotagged news in California area during the last three months*”. ST-Hadoop employs its spatio-temporal index described in Sect. 5 to provide an efficient algorithm that runs in three steps, *temporal filtering*, *spatial search*, and *spatio-temporal refinement*, described below.

In the **temporal filtering** step, the hierarchy index is examined to select a subset of partitions that cover the temporal interval T . The main challenge in this step is that the partitions in each granularity cover the whole time and space, which means the query can be answered from any level individually or we can mix and match partitions from different level to cover the query interval T . Depending on which granularities are used to cover T , there is a tradeoff between the number of matched partitions and the amount of processing needed to process each partition. To decide whether a partition P is selected or not, the algorithm computes its *coverage ratio* r , which is defined as the ratio of the time interval of P that overlaps T . A partition is selected only if its *coverage ratio* is above a specific threshold \mathcal{M} . To balance this tradeoff, ST-Hadoop employs a top-down approach that starts with the top level and selects partitions that covers query interval T . If the query interval T is not covered at that granularity, then the algorithm continues to the next level. If the bottom level is reached, then all partitions overlap with T will be selected.

In the **spatial search** step, Once the temporal partitions are selected, the *spatial search* step applies the spatial range query against each matched partition to select records that spatially match the query range A . Keep in mind that each partition is spatiotemporally indexed which makes queries run very efficiently. Since these partitions are indexed independently, they can all be processed simultaneously across computation nodes in ST-Hadoop, and thus maximizes the computing utilization of the machines.

Finally in the **spatio-temporal refinement** step, compares individual records returned by the *spatial search* step against the query interval T , to select the exact matching records. This step is required as some of the selected temporal partitions might partially overlap the query interval T and they need to be refined to remove records that are outside T . Similarly, there is a chance that selected partitions might partially overlap with the query area A , and thus records outside the A need to be excluded from the final answer.

6.2 Spatio-Temporal Join

Given two indexed dataset R and S of spatio-temporal records, and a spatio-temporal predicate θ . The join operation retrieves all pairs of records $\langle r, s \rangle$ that are similar to each other based on θ . For example, one might need to understand the relationship between the birds death and the existence of humans around

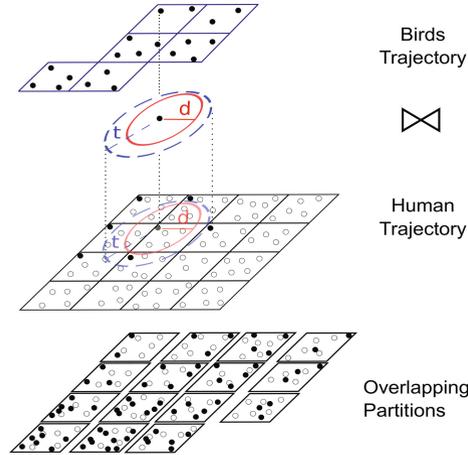


Fig. 7. Spatio-temporal join

them, which can be described as “find every pairs from bird and human trajectories that are close to each other within a distance of 1 mile during the last week”. The join algorithm runs in two steps as shown in Fig. 7, *hash* and *join*.

In the **hashing** step, the map function scans the two input files and hashes each record to candidate buckets. The buckets are defined by partitioning the spatio-temporal space using the two-layer indexing of temporal and spatial, respectively. The granularity of the partitioning controls the tradeoff between partitioning overhead and load balance, where a more granular-partitioning increases the replication overhead, but improves the load balance due to the huge number of partitions, while a less granular-partitioning minimizes the replication overhead, but can result in a huge imbalance especially with highly skewed data. The hash function assigns each point in the left dataset, $r \in R$, to all buckets within an Euclidean distance d and temporal distance t , and assigns each point in the right dataset, $s \in S$, to the one bucket which encloses the point s . This ensures that a pair of matching records $\langle r, s \rangle$ are assigned to at least one common bucket. Replication of only one dataset (R) along with the use of single assignment, ensure that the answer contains no replicas.

In the **joining** step, each bucket is assigned to one reducer that performs a traditional in-memory spatio-temporal join of the two assigned sets of records from R and S . We use the plane-sweep algorithm which can be generalized to multidimensional space. The set S is not replicated, as each pair is generated by exactly one reducer, and thus no *duplicate avoidance* step is necessary.

7 Experiments

This section provides an extensive experimental performance study of ST-Hadoop compared to SpatialHadoop and Hadoop. We decided to compare with

this two frameworks and not other spatio-temporal DBMSs for two reasons. First, as our contributions are all about spatio-temporal data support in Hadoop. Second, the different architectures of spatio-temporal DBMSs have great influence on their respective performance, which is out of the scope of this paper. Interested readers can refer to a previous study [32] which has been established to compare different large-scale data analysis architectures. In other words, ST-Hadoop is targeted for Hadoop users who would like to process large-scale spatio-temporal data but are not satisfied with its performance. The experiments are designed to show the effect of ST-Hadoop indexing and the overhead imposed by its new features compared to SpatialHadoop. However, ST-Hadoop achieves two orders of magnitude improvement over SpatialHadoop and Hadoop.

Experimental Settings. All experiments are conducted on a dedicated internal cluster of 24 nodes. Each has 64 GB memory, 2 TB storage, and Intel(R) Xeon(R) CPU 3 GHz of 8 core processor. We use Hadoop 2.7.2 running on Java 1.7 and Ubuntu 14.04.5 LTS. Figure 8(b) summarizes the configuration parameters used in our experiments. Default parameters (in parentheses) are used unless mentioned.

Datasets. To test the performance of ST-Hadoop we use the Twitter archived dataset [2]. The dataset collected using the public Twitter API for more than three years, which contains over 1 Billion spatio-temporal records with a total size of 10 TB. To scale out time in our experiments we divided the dataset into different time intervals and sizes, respectively as shown in Fig. 8(a). The default size used is 1 TB which is big enough for our extensive experiments unless mentioned.

In our experiments, we compare the performance of a ST-Hadoop spatio-temporal range and join query proposed in Sect. 6 to their spatial-temporal implementations on-top of SpatialHadoop and Hadoop. For range query, we use system throughput as the performance metric, which indicates the number of MapReduce jobs finished per minute. To calculate the throughput, a batch of 20

Twitter Data	Size	Num-Records	Time window
Large	10TB	> 1 Billion	> 3 years
Average-Large	6.7TB	692 Million	1 years
Medium-Large	3TB	152 Million	9 months
Moderate-Large	(1TB)	115 Million	3 months

(a) Datasets

Parameter	Values (default)
HDFS block capacity (B)	32, 64, (128), 256 MB
Cluster size (N)	5, 10, 15, 20, (23)
Selection ratio (ρ)	(0.01), 0.02, 0.05, 0.1, 0.2, 0.5, 1.0
Data-partition slicing ratio(α)	0.01, 0.02, 0.025, 0.05, (0.1), 1
Time-partition Slicing granularity(σ)	(days), weeks, months, years

(b) Parameters

Fig. 8. Experimental settings and Dataset

queries is submitted to the system, and the throughput is calculated by dividing 20 by the total time of all queries. The 20 queries are randomly selected with a spatial area ratio of 0.001% and a temporal window of 24 h unless stated. This experimental design ensures that all machines get busy and the cluster stays fully utilized. For spatio-temporal join, we use the processing time of one query as the performance metric as one query is usually enough to keep all machines busy. The experimental results for range and join queries are reported in Sects. 7.1, and 7.3, respectively. Meanwhile, Sect. 7.2 analyzes ST-Hadoop indexing.

7.1 Spatiotemporal Range Query

In Fig. 9(a), we increase the size of input from 1 TB to 10 TB, while measuring the job throughput. ST-Hadoop achieves more than two orders of magnitude higher throughput, due to the temporal load balancing of its spatio-temporal index. As for SpatialHadoop, it needs to scan more partitions, which explain why the throughput of SpatialHadoop decreases with the increase of data records in spatial space. Meanwhile, ST-Hadoop throughput remains stable as it processes only partition(s) that intersect with both space and time. Note that it is always the case that Hadoop needs to scan all HDFS blocks, which gives the worst throughput compared to SpatialHadoop and ST-Hadoop.

Figure 9(b) shows the effect of configuring the HDFS block size on the job throughput. ST-Hadoop manages to keep its performance within orders of magnitude higher throughput even with different block sizes. Extensive experiments are shown in Fig. 9(c), analyzed how slicing ratio (α) can affect the performance of range queries. ST-Hadoop keeps its higher throughput around the default HDFS block size, as it maintains the load balance of data records in its two-layer indexing. As expected expanding the block size from its default value will reduce the performance on SpatialHadoop and ST-Hadoop, mainly because blocks will carry more data records.

Experiments in Fig. 10 examines the performance of the temporal hierarchy index in ST-Hadoop using both slicing techniques. We evaluate different granularities of time-partition slicing (e.g., daily, weekly, and monthly) with various data-partition slicing ratio. In these two figures, we fix the spatial query range

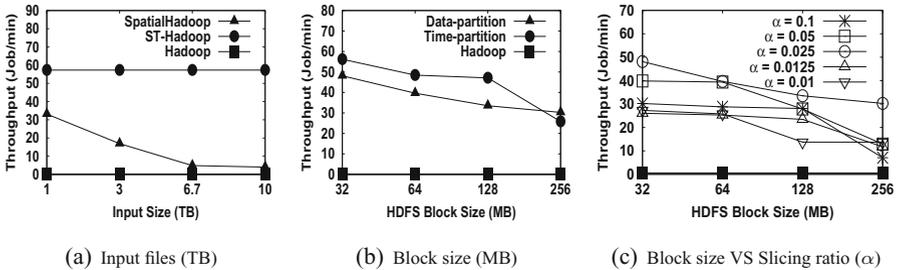


Fig. 9. Spatio-temporal range query

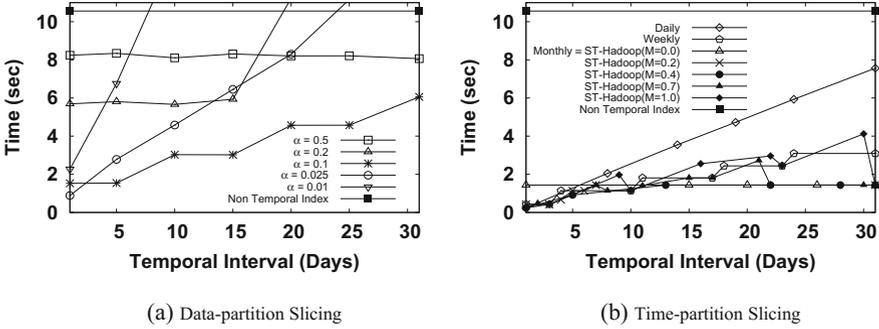


Fig. 10. Spatio-temporal range query interval window

and increase the temporal range from 1 day to 31 days, while measuring the total running time. As shown in the Figs. 10(a) and (b), ST-Hadoop utilizes its temporal hierarchy index to achieve the best performance as it mixes and matches the partitions from different levels to minimize the running time, as described in Sect. 6.1. ST-Hadoop provides good performance for both small and large query intervals as it selects partitions from any level. When the query interval is very narrow, it uses only the lowest level (e.g., daily level), but as the query interval expand it starts to process the above level. The value of the parameter \mathcal{M} controls when it starts to process the next level. At $\mathcal{M} = 0$, it always selects the up level, e.g., monthly. If \mathcal{M} increases, it starts to match with lower levels in the hierarchy index to achieve better performance. At the extreme value of $\mathcal{M} = 1$, the algorithm only matches partitions that are completely contained in the query interval, e.g., at 18 days it matches two weeks and four days while at 30 days it matches the whole month. The optimal value in this experiment is $\mathcal{M} = 0.4$ which means it only selects partitions that are at least 40% covered by the query temporal interval.

In Fig. 11 we study the effect of the spatio-temporal query range (σ) on the choice of \mathcal{M} . To measure the quality of \mathcal{M} , we define an optimal running time for a query Q as the minimum of all running times for all values of $\mathcal{M} \in [0, 1]$. Then, we determine the quality of a specific value of \mathcal{M} on a query workload as the mean squared error (MSE) between the running time at this value of \mathcal{M} and the optimal running time. This means, if a value of \mathcal{M} always provides the optimal value, it will yield a quality measure of zero. As this value increases, it indicates a poor quality as the running times deviates from the optimal. In Fig. 11(a), We repeat the experiment with three values of spatial query ranges $\sigma \in \{1E - 6, 1E - 4, 0.1\}$. As shown in the figure, $\mathcal{M} = 0.4$ provides the best performance for all the experimented spatial ranges. This is expected as \mathcal{M} is only used to select temporal partitions while the spatial range (σ) is used to perform the spatial query inside each of the selected partitions. Figure 11(b), shows the quality measures with a workload of 71 queries with time intervals

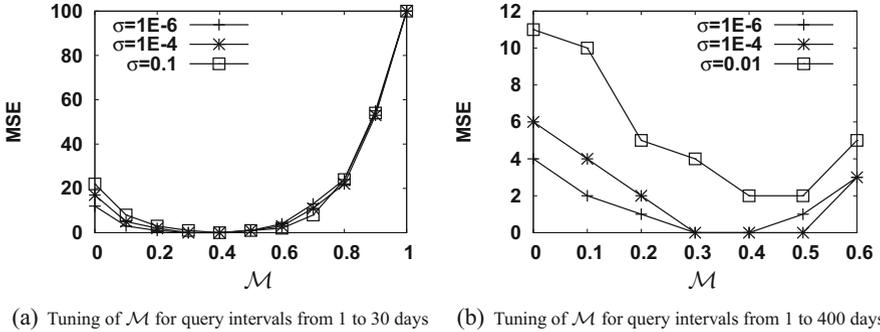


Fig. 11. The effect of the spatio-temporal query ranges on the optimal value of \mathcal{M}

that range from 1 day to 421 days. This experiment also provides a very similar result where the optimal value of \mathcal{M} is around 0.4.

7.2 Index Construction

Figure 12 gives the total time for building the spatio-temporal index in ST-Hadoop. This is a one time job done for input files. In general, the figure shows excellent scalability of the index creation algorithm, where it builds its index using data-partition slicing for a 1 TB file with more than 115 Million records in less than 15 min. The data-partition technique turns out to be the fastest as it contains fewer slices than time-partition. Meanwhile, the time-partition technique takes more time, mainly because the number of partitions are increased, and thus increases the time in physical writing phase.

In Fig. 13, we configure the temporal hierarchy indexing in ST-Hadoop to construct five levels of the two-layer indexing. The temporal indexing uses **Data-partition** slicing technique with different slicing ratio α . We evaluate the indexing time of each level individually. Because the input files are sliced into splits according to the slicing ratio, which directly effects on the number of partitions. In general with stretching the slicing ratio, the indexing time decreases, mainly because the number of partitions will be much less. However, note that in some cases the spatial distribution of the slice might produce more partitions as in shown with 0.25% ratio.

7.3 Spatiotemporal Join

Figure 14 gives the results of the spatio-temporal join experiments, where we compare our join algorithm for ST-Hadoop with MapReduce implementation of the spatial hash join algorithm [33]. Typically, in this join algorithm we perform the following query, “*find every pairs that are close within an Euclidean distance of 1mile and a temporal distance of 2 days*”, this join query is executed on both ST-Hadoop and Hadoop and the response times are compared. The y-axis in the

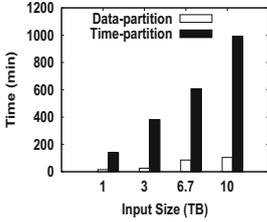


Fig. 12. Input files

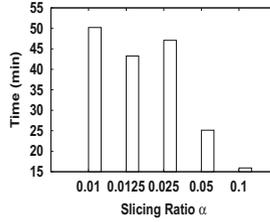


Fig. 13. Data-partition

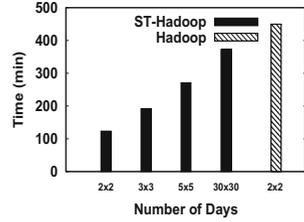


Fig. 14. Spatio-temporal join

figure represents the total processing time, while the x-axis represents the join query on numbers of days \times days in ascending order. With the increase of joining number of days, the performance of ST-Hadoops join increases, because it needs to join more indexes from the temporal hierarchy. In general, ST-Hadoop gives the best results as ST-Hadoop index replicates data in several layers, and thus ST-Hadoop significantly decreases the processing of non-overlapping partitions, as only partitions that overlap with both space and time are considered in the join algorithm. Meanwhile, the same joining algorithm without using ST-Hadoop index gives the worst performance for joining spatio-temporal data, mainly because the algorithm takes into its consideration all data records from one dataset. However, ST-Hadoop only joins the indexes that are within the temporal range, which significantly outperforms the join algorithm with double to triple performance.

8 Conclusion

In this paper, we introduced ST-Hadoop [19] as a novel system that acknowledges the fact that space and time play a crucial role in query processing. ST-Hadoop is an extension of a Hadoop framework that injects spatio-temporal awareness inside SpatialHadoop layers. The key idea behind the performance gain of ST-Hadoop is its ability to load the data in Hadoop Distributed File System (HDFS) in a way that mimics spatio-temporal index structures. Hence, incoming spatio-temporal queries can have minimal data access to retrieve the query answer. ST-Hadoop is shipped with support for two fundamental spatio-temporal queries, namely, spatio-temporal range and join queries. However, ST-Hadoop is extensible to support a myriad of other spatio-temporal operations. We envision that ST-Hadoop will act as a research vehicle where developers, practitioners, and researchers worldwide, can either use directly or enrich the system by contributing their operations and analysis techniques.

References

1. NYC Taxi and Limousine Commission (2017). http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
2. (2017). <https://about.twitter.com/company>
3. Land Process Distributed Active Archive Center, March 2017. <https://lpdaac.usgs.gov/about>
4. Data from NASA's Missions, Research, and Activities (2017). <http://www.nasa.gov/open/data.html>
5. European XFEL: The Data Challenge, September 2012. http://www.xfel.eu/news/2012/the_data_challenge
6. Apache. Hadoop. <http://hadoop.apache.org/>
7. Apache. Spark. <http://spark.apache.org/>
8. Whitman, R.T., Park, M.B., Ambrose, S.A., Hoel, E.G.: Spatial indexing and analytics on hadoop. In: SIGSPATIAL (2014)
9. Lu, J., Guting, R.H.: Parallel secondo: boosting database engines with hadoop. In: ICPADS (2012)
10. Nishimura, S., Das, S., Agrawal, D., El Abbadi, A.: *MD-HBase*: design and implementation of an elastic data infrastructure for cloud-scale location services. *DAPD* **31**, 289–319 (2013)
11. 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. In: VLDB (2013)
12. Kini, A., Emanuele, R.: Geotrellis: adding geospatial capabilities to spark (2014). <http://spark-summit.org/2014/talk/geotrellis-adding-geospatial-capabilities-to-spark>
13. Yu, J., Wu, J., Sarwat, M.: GeoSpark: a cluster computing framework for processing large-scale spatial data. In: SIGSPATIAL (2015)
14. Eldawy, A., Mokbel, M.F.: SpatialHadoop: a MapReduce framework for spatial data. In: ICDE (2015)
15. Ma, Q., Yang, B., Qian, W., Zhou, A.: Query processing of massive trajectory data based on MapReduce. In: CLOUDDB (2009)
16. Tan, H., Luo, W., Ni, L.M.: Clost: a hadoop-based storage system for big spatio-temporal data analytics. In: CIKM (2012)
17. Li, Z., Hu, F., Schnase, J.L., Duffy, D.Q., Lee, T., Bowen, M.K., Yang, C.: A spatiotemporal indexing approach for efficient processing of big array-based climate data with mapreduce. *Int. J. Geograph. Inf. Sci. IJGIS* **31**, 17–35 (2017)
18. Eldawy, A., Mokbel, M.F., Alharthi, S., Alzaidy, A., Tarek, K., Ghani, S.: SHAHED: a MapReduce-based system for querying and visualizing Spatio-temporal satellite data. In: ICDE (2015)
19. ST-Hadoop website. <http://st-hadoop.cs.umn.edu/>
20. Eldawy, A., Mokbel, M.F.: Pigeon: a spatial mapreduce language. In: ICDE (2014)
21. Han, W., Kim, J., Lee, B.S., Tao, Y., Rantzau, R., Markl, V.: Cost-based predictive spatiotemporal join. *TKDE* **21**, 220–233 (2009)
22. Al-Naami, K.M., Seker, S.E., Khan, L.: GISQF: an efficient spatial query processing system. In: CLOUDCOM (2014)
23. Fries, S., Boden, B., Stepien, G., Seidl, T.: PHIDJ: parallel similarity self-join for high-dimensional vector data with mapreduce. In: ICDE (2014)
24. Stonebraker, M., Brown, P., Zhang, D., Becla, J.: SciDB: a database management system for applications with complex analytics. *Comput. Sci. Eng.* **15**, 54–62 (2013)

25. Zhang, X., Ai, J., Wang, Z., Lu, J., Meng, X.: An efficient multi-dimensional index for cloud data management. In: CIKM (2009)
26. Wang, G., Salles, M., Sowell, B., Wang, X., Cao, T., Demers, A., Gehrke, J., White, W.: Behavioral simulations in MapReduce. PVLDB **3**, 952–963 (2010)
27. Lu, P., Chen, G., Ooi, B.C., Vo, H.T., Wu, S.: ScalaGiST: scalable generalized search trees for MapReduce systems. PVLDB **7**, 1797–1808 (2014)
28. Fox, A.D., Eichelberger, C.N., Hughes, J.N., Lyon, S.: Spatio-temporal indexing in non-relational distributed databases. In: BIGDATA (2013)
29. GeoWave. <https://ngageoint.github.io/geowave/>
30. Accumulo. <https://accumulo.apache.org/>
31. Erwig, M., Schneider, M.: Spatio-temporal predicates. In: TKDE (2002)
32. Pavlo, A., Paulson, E., Rasin, A., Abadi, D., DeWitt, D., Madden, S., Stonebraker, M.: A comparison of approaches to large-scale data analysis. In: SIGMOD (2009)
33. Lo, M.L., Ravishankar, C.V.: Spatial hash-joins. In: SIGMODR (1996)