A Demonstration of ST-Hadoop: A MapReduce Framework for Big Spatio-temporal Data *

Louai Alarabi
Department of Computer Science and Engineering
University of Minnesota, MN, USA
louai@cs.umn.edu

Mohamed F. Mokbel
Department of Computer Science and Engineering
University of Minnesota, MN, USA
mokbel@cs.umn.edu

ABSTRACT

This demo presents ST-Hadoop; the first full-fledged open-source MapReduce framework with a native support for spatio-temporal data. ST-Hadoop injects spatio-temporal awareness in the Hadoop base code, which results in achieving order(s) of magnitude better performance than Hadoop and SpatialHadoop when dealing with spatio-temporal data and queries. The key idea behind ST-Hadoop is its ability in indexing spatio-temporal data within Hadoop Distributed File System (HDFS). A real system prototype of ST-Hadoop, running on a local cluster of 24 machines, is demonstrated with two big-spatio-temporal datasets of Twitter and NYC Taxi data, each of around one billion records.

1. INTRODUCTION

Processing big spatio-temporal data has gained much interest in the last few years, mainly due to the emerging and popularity of devices and applications that create them in large-scale. Examples of such data include Taxi trajectory data that archive over 1.1 Billion trajectories [8], social network data (e.g., Twitter has over 500 Million new tweets everyday) [11], and NASA Satellite data that produces 4TB of data every day [6, 7]. The current efforts to process big spatio-temporal data on MapReduce environment are either: (a) run on-top of Hadoop framework [4, 5, 10]. However, using Hadoop as-is results in a sub-performance for spatio-temporal applications that need indexing, which is in contrast to Hadoop that organizes its data as non-indexed heap files, or (b) run as an ad-hoc operation using big spatial data systems e.g., SHAHED system [3] is built on top of SpatialHadoop [2]. However, big spatial data systems only support spatial indexing, which is not efficient in supporting spatio-temporal queries.

This demo presents ST-Hadoop; the first full-fledged open-source MapReduce framework with a native support for spatio-temporal data, available to download from [9]. ST-Hadoop is a comprehensive extension to Hadoop and SpatialHadoop [2] that injects spatio-temporal data awareness inside each of their layers, mainly, indexing, operatiosn, and language layers. ST-Hadoop is compatible with to 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 order(s) of magnitude better performance than Hadoop and SpatialHadoop. Figures 1(a) and 1(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 10TB and a cluster of 24 nodes takes 200 seconds on SpatialHadoop as opposed to only one second on ST-Hadoop. The main reason is that SpatialHadoop will exploit its built-in spatial index and then scan all the entries that overlap with the spatial predicate to find the data entries that overlap with the temporal predicate. 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.

```
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

Figure 1: Range query in SpatialHadoop vs. ST-Hadoop

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, and we plan to show the demo audience how to build applications on top of what we have for now. 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 inside. We will demonstrate a real system prototype of ST-Hadoop, running on a local cluster of 24 machines, with two big-spatio-temporal datasets of Twitter and NYC Taxi data, each of around one billion records.

*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.

Proceedings of the VLDB Endowment, Vol. 10, No. 12
Copyright 2017 VLDB Endowment 2150-8097/17/08.

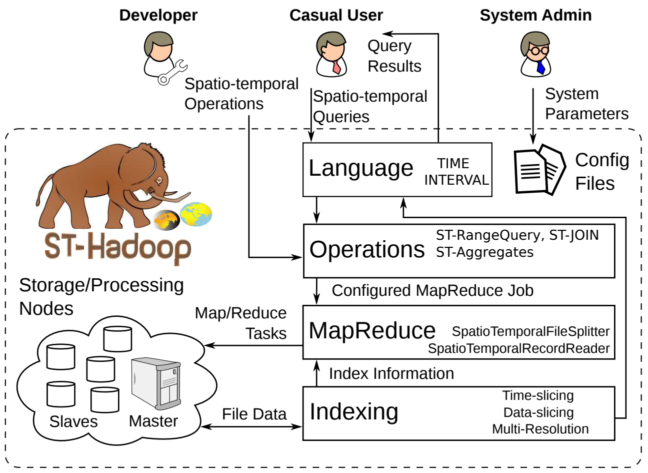


Figure 2: ST-Hadoop system architecture

2. ST-Hadoop ARCHITECTURE

Figure 2 gives the high level architecture of our ST-Hadoop system; as 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 language to process their datasets. (2) *Developers*, who have a deeper understanding of the system internals and can implement new operations, and (3) *Administrators*, who can tune up the system through adjusting system parameters in the configuration files provided with 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 [1] to supports spatio-temporal data types that are associated with (i.e., TIME and INTERVAL) and operations (e.g., OVERLAP, and JOIN).

Indexing Layer: ST-Hadoop temporally loads and divides data across computation nodes in the HDFS. In this layer ST-Hadoop reads a random sample obtained from the whole data, bulk loads this sample temporary in the master node memory to build its spatio-temporal index, and then uses the spatio-temporal boundaries of leaves-nodes to assign data records with its overlap partition(s). ST-Hadoop sacrifices storage to achieve more efficient performance in supporting spatio-temporal operations, by replicating its index into *temporal hierarchy index structure* that consist of two-level indexing of temporal and then spatial.

MapReduce Layer: ST-Hadoop changes the implementation of SpatialHadoop MapReduce layer, which enables ST-Hadoop to exploits its spatio-temporal indexes and realizes spatio-temporal predicates. We are not going to discuss this layer any further.

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

3. LANGUAGE LAYER

ST-Hadoop does not provide a completely new language. Instead, it extends Pigeon [1] by adding spatio-temporal data types, functions, and operations. Spatio-temporal data types (STPoint and Interval) are used to define the schema of input files upon their loading process. The following code snippet loads NYC taxi trajectories from 'NYC' file with a column of type `STPoints`.

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

`trajectory` and `NYC` 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.

ST-Hadoop encapsulate the implementation of two common spatio-temporal operations, (e.g., range query and Join query), that take the advantages of the spatio-temporal index. The following example retrieves all cars in State Fair area represented by it's minimum boundary rectangle during the time interval of August 25th and September 6th from trajectory indexed file.

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

4. INDEXING LAYER

The key idea behind the performance gain of ST-Hadoop is the idea of its indexing, where data are temporally loaded and divided across computation nodes. Figure 3 Illustrates the abstract idea of indexing in ST-Hadoop, that consist of two-level indexing of temporal and then spatial. ST-Hadoop replicates this two-level indexing into multiple layers with a different granularity. In each layer, the whole dataset is replicated and spatiotemporally partitioned. ST-Hadoop sacrifices storage to achieve a more efficient performance. ST-Hadoop index input files through the following three consecutive phases, as described in details below:

Phase 1: Sampling. The objective of this phase is to estimate the spatial distribution of objects and how that distribution evolves over time. ST-Hadoop employs a Map-Reduce job to efficiently read a sample through scanning all data records. We fit the sample to 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., 128 MB). The size of the random sample is set to a default ratio of 1% of the input file, with a maximum size of 100MB to ensure it fits in the memory of the Master node. Once the sample is read, we sort the sample in a chronological order to their time, and within individual time instance in the sample, we estimate the size and the number of data records associated with each time instance.

Phase 2: Bulk Loading. The input of this phase is the sample loaded into memory, and the output of this phase is the spatio-temporal boundaries of ST-Hadoop index. The bulk loading index can be constructed as following:

• *Temporal slicing*. In this step we determine the temporal boundaries by slicing the in-memory sample that represents the data in HDFS into multiple intervals, to efficiently support a fast random access to a sequence of objects bounded by the same time interval. ST-Hadoop employs two slicing techniques, namely, Time-Based and Data-Based. The *Time-Based* technique slices the sample into multiple splits, that are uniformly on their time intervals. Figure 4 shows the general idea of this type of slicing, where ST-Hadoop splits a year of data into the interval of one month. While the time interval of the slices is fixed, the size of slices varies according to the time distribution of the data. Meanwhile, in *Data-Based* slicing the sample is sliced to the degree that all sub-splits are uniformly in their data size. All slices hold the same number of data blocks, while their time intervals are disjointed. Figure 5 depicts the key concept such that a $\texttt{slice}_1$ and $\texttt{slice}_n$ are equally in size, but they differs in their interval coverage.

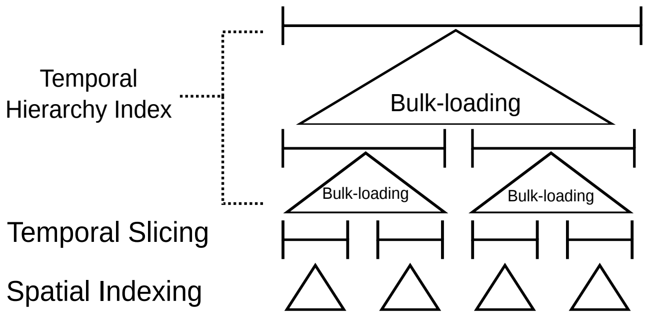


Figure 3: Indexing in ST-Hadoop

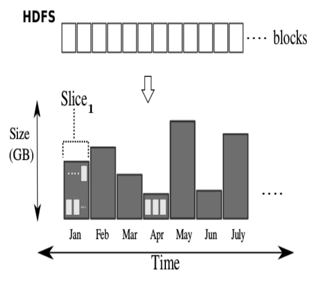


Figure 4: Time-Slice

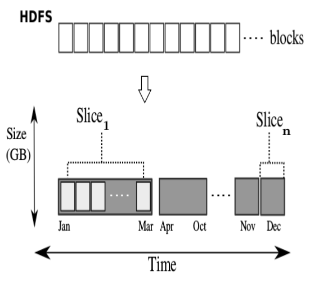


Figure 5: Data-Slice

• *Spatial Indexing*. This step ST-Hadoop determines the spatial boundaries of the data within each temporal slice. ST-Hadoop needs to find the spatial boundaries of each temporal slice independently, such decision handles a case where there are a great disparity in spatial distribution between slices. ST-Hadoop takes the advantages of applying different types of spatial bulk loading techniques that are already implemented in SpatialHadoop such as Grid, R-tree-like, Quad-tree, and Kd-tree. By the end of this step, ST-Hadoop bulk loaded the spatio-temporal boundaries for the data in its smallest granularity, as shown in the lowermost of Figure 3.

• *Temporal Hierarchy Indexing*. For an efficient retrieval of spatio-temporal objects, ST-Hadoop bulk loads its two-level indexing of temporal and then spatial, into *temporal hierarchy index*. This orthogonal structure can be described as a temporal hierarchy for spatio-temporal indices as shown in uppermost of Figure 3. ST-Hadoop combines a set of temporal slices from the lowermost layer to create a slice with a larger time interval. For simplicity let's assume the lowermost layer is sliced into days, then ST-Hadoop combines a set days to create a week-slice in the layer above. Likewise, ST-Hadoop reads a sample from the merged set to bulk load its spatio-temporal index. Note that this step is necessary as the distribution of objects vary from day to day. Once ST-Hadoop receives the sample, it determines the spatial boundaries. For each layer in the hierarchical index, ST-Hadoop iterates this bulk loading technique. Each iteration produces a set of a two-level indexing with different time resolution, such as weeks, months, and years. A system parameter can be tuned by ST-Hadoop administrator to choose the number of layers and their resolution. By default, ST-Hadoop set its *temporal hierarchy index* to four layers with a resolution of days, weeks, months and years, respectively. ST-Hadoop maintains its hierarchical index on a regular basis, using a single map-reduce job to reflect newly received data on the index.

Phase 3: Scanning. Given the spatio-temporal boundaries in all layers inside the *temporal hierarchy index*, we initiate a map-reduce job that scans through the input file, physically partitions HDFS block, and assign records to all overlapping partitions. ST-Hadoop partitions are determined by bulk loaded spatio-temporal boundaries, which they are being cloned from all layers inside the temporal hierarchical index structure. 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 partitions to the HDFS.

5. OPERATION LAYER

The combination of the temporally load balancing proposed in the spatio-temporal indexing layer with the temporal hierarchy structures gives the core novelty of ST-Hadoop, in which it enables the possibility of realizing many spatio-temporal operations. In this demonstration, we show the implementations of spatio-temporal range and join queries as case studies of how to exploit the new indexing in ST-Hadoop. Other spatio-temporal operations, e.g., *k*NN, reverse nearest neighbor, and aggregation queries, can also be realized in ST-Hadoop following a similar approach.

Spatio-temporal Range query: ST-Hadoop exploits its *temporal hierarchy index* to select partitions that overlap with the temporal and spatial query predicates. For example, consider *"finding geo-tagged news in California area during the last three months"* . The answer of this query can be obtained from any level (i.e., resolution) in the *temporal hierarchy index*, such as day-level, week-level, or month-level. The main challenge in this step is that the partitions in each level 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 resolutions. Depending on which level are used, there is a tradeoff between the number of matched partitions and the time needed to process each one of them. One extreme, if the highest-resolution that has a short interval, e.g. daily level, there will be too many partitions to process with a little work to handle each one. On the other extreme, if the lowest-resolution with long intervals, e.g., monthly level, is used, only a few partitions selected, but an extra effort required to process them. ST-Hadoop balance this tradeoff by employing an algorithm to computes the *coverage ratio*, which is defined as the ratio between the time interval of a partition and the query predicates.

Spatio-temporal Join: Given two indexed dataset R and S of spatio-temporal records, and a spatio-temporal predicate θ. Spatio-temporal join joins two spatio-temporal dataset $R.S$ on spatio-temporal predicate θ. 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 bird and human trajectories that are close to each other within a distance of 1 mile during the last week"*. ST-Hadoop join algorithm run a map-reduce in two steps, *hash* and *join*, respectively. In the *hash* step, a map function scans overlapped partitions from the two input files, and hashes each spatio-temporal records to intermediate buckets. The spatio-temporal boundaries of intermediate buckets created by ST-Hadoop as non-overlapping buckets. 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. In the *join* step, each bucket is assigned to one reducer which performs a traditional in-memory spatio-temporal join. We use the plane-sweep algorithm which can be generalized to multidimensional space. Since the set S is not replicated, each pair is generated by exactly one reducer and no *duplicate avoidance* step is necessary.

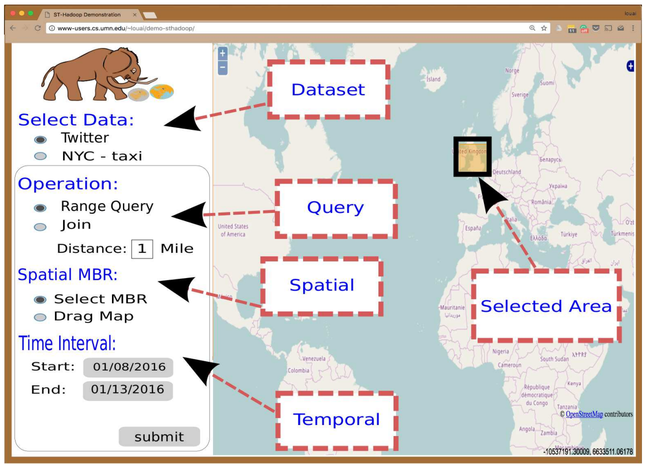
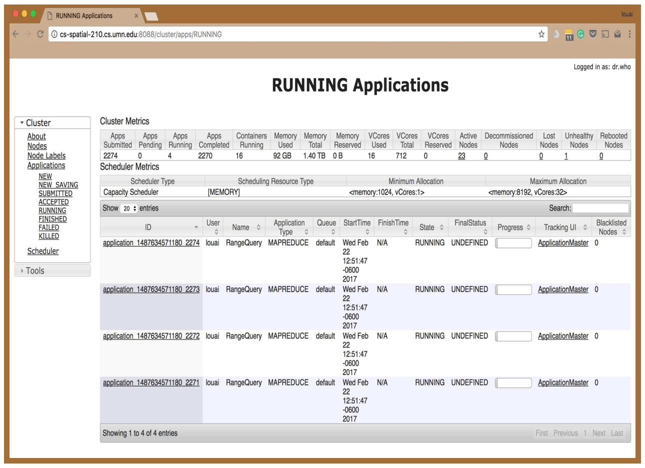


Figure 6: ST-Hadoop Front Interface



Logged in as: dr.who

RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
2274	0	4	2270	16	92 GB	1.40 TB	0 B	16	712	0	23	0	0	1	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:32>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1487634571180_2274	louai	RangeQuery	MAPREDUCE	default	Wed Feb 22 12:51:47 -0600 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster	0
application_1487634571180_2273	louai	RangeQuery	MAPREDUCE	default	Wed Feb 22 12:51:47 -0600 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster	0
application_1487634571180_2272	louai	RangeQuery	MAPREDUCE	default	Wed Feb 22 12:51:47 -0600 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster	0
application_1487634571180_2271	louai	RangeQuery	MAPREDUCE	default	Wed Feb 22 12:51:47 -0600 2017	N/A	RUNNING	UNDEFINED		ApplicationMaster	0

Showing 1 to 4 of 4 entries First Previous 1 Next Last

Figure 7: ST-Hadoop Jobs Progress

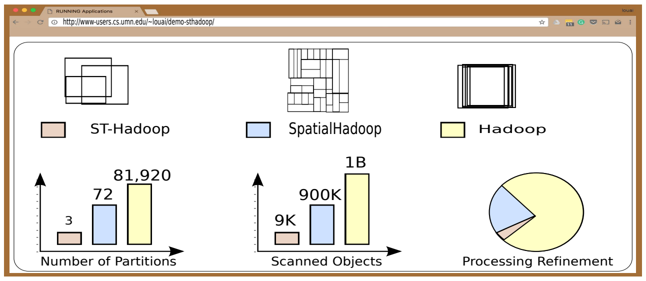


Figure 8: Partitions Visualization

6. DEMONSTRATION SCENARIO

We deploy the real system prototype of ST-Hadoop downloadable at [9] on a dedicated remote cluster of 24 nodes. The cluster is loaded with two datasets Twitter and NYC taxi where each includes one billion records, with a total size of 10TB and 300GB, respectively. The attendee can access the cluster and trigger the execution of their spatio-temporal queries through a nicely designed front-end. For a comparision the query will be submitted to ST-Hadoop, SpatialHadoop, and Hadoop frameworks, respectively. The three frameworks installed on a separate cluster with the same configuration and dataset.

Figure 6 depicts the system interface, which is a querying and visualization tool to show the usability and efficiency of ST-Hadoop. In this demonstration, we will demonstrate two scenarios, (Scenario 1) will show the usability and running process of ST-Hadoop, and the other one will reveal to the audience a live comparison between ST-Hadoop, SpatialHadoop, and Hadoop (Scenario 2).

6.1 Scenario 1: Spatio-temporal Queries

Conference audience interact with ST-Hadoop by issuing two spatio-temporal queries (i.e., range and join) as shown in the forntend interface in Figure 6. This interface facilitates the ST-Hadoop user to provide, (1) spatial range, (2) time range, (3) operation, and (4) switch between dataset. For example, a user requests a spatio-temporal range query that *"finds all objects in downtown California during last weeks"* . The user can freely navigates through the query answer on the front-end map. We will show how ST-Hadoop performs spatio-temporal join to *"finds similar pairs of objects in downtown Minneapolis, such that pair is close to each other by a spatial threshold d (1 Mile) and within the temporal duration t (Jun-Sep)"* . These are two scenarios can be tuned by audience to show the powerful ability of ST-Hadoop to perform spatio-temporal operations efficiently; even with a more complex query such as the spatio-temporal join.

6.2 Scenario 2: Comparison with (Spatial) Hadoop

Conference attendees can see the progress of the submitted query on all three system side-by-side as shown in Figure 7. This administration interface lists the progress of all running jobs as well as all completed job. To contrast the index of ST-Hadoop with Spatial-Hadoop and Hadoop, the audience can grasp an overview about the processed partitions. As shown in Figure 8, a comparison between the selected partitions from all three frameworks will be dynamically displayed on the screen as a bar chart. Likewise, the number of scanned objects and the percentage of refinement depicted in bar and pie chart, respectively. Since the data inside Hadoop treated as non-indexed heap file, it will always be the case that all partitions will be processed for any task. Meanwhile, SpatialHadoop equipped with its spatial indexes can choose a fewer number of partitions to process, but it fails to realize the temporal locality of the data. On the other hand, ST-Hadoop index realizes the spatial and temporal locality of its indexed data. Thus, ST-Hadoop will utilize to minimize the number of selected partitions compared to other frameworks, which plays a crucial role in query processing.

7. REFERENCES

[1] A. Eldawy and M. F. Mokbel. Pigeon: A spatial mapreduce language. In *ICDE*, pages 1242–1245, 2014.
[2] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce Framework for Spatial Data. In *ICDE*, pages 1352–1363, 2015.
[3] A. Eldawy, M. F. Mokbel, S. Alharthi, A. Alzaidy, K. Tarek, and S. Ghani. SHAHED: A MapReduce-based System for Querying and Visualizing Spatio-temporal Satellite Data. In *ICDE*, pages 1585–1596, 2015.
[4] Z. Li, F. Hu, J. L. Schnase, D. Q. Duffy, T. Lee, M. K. Bowen, and C. Yang. A spatiotemporal indexing approach for efficient processing of big array-based climate data with mapreduce. *International Journal of Geographical Information Science*, pages 17–35, 2017.
[5] Q. Ma, B. Yang, W. Qian, and A. Zhou. Query Processing of Massive Trajectory Data Based on MapReduce. In *CLOUDDB*, pages 9–16, 2009.
[6] Land Process Distributed Active Archive Center, Mar. 2015. https://lpdaac.usgs.gov/about.
[7] Data from NASA's Missions, Research, and Activities, 2017. http://www.nasa.gov/open/data.html.
[8] Data from NYC Taxi and Limosuine Commission, 2017. http://www.nyc.gov/html/tlc/.
[9] http://st-hadoop.cs.umn.edu/.
[10] H. Tan, W. Luo, and L. M. Ni. Clost: a hadoop-based storage system for big spatio-temporal data analytics. In *CIKM*, pages 2139–2143, 2012.
[11] Twitter. The About webpage., 2017. https://about.twitter.com/company.