SIGSPATIAL: G: Summit: A Scalable System for Massive Trajectory Data Management

Louai Alarabi
University of Minnesota, Minneapolis, MN, USA. Advisor: Mohamed F. Mokbel
Email: louai@cs.umn.edu

1 PROBLEM AND MOTIVATION

Recent advances in mobile computing, sensor networks, GPS, and satellite technology have made it possible to produce massive amounts of trajectory data. This results in an increasing interest from scientists and domain experts in performing analysis tasks over such huge data [14]. For example, NASA publicly archives over 4TB of stars and asteroids movement activity [32]. Sloan Digital Sky Survey project collects over 156TB of motion data from millions of outer-space objects [40]. MoveBank project gathers more than 20 years of animal movements [31]. New York Taxi & Limousine archives over a Billion of taxi trajectories [34]. National Hurricane Center stores comprehensive details of all storms' trajectories every year [33]. Managing massive trajectory data is the foundation of many applications [58], including urban, data mining, and machine learning computing.

Domain experts who analyze trajectory data either (a) use *Heterogeneous* multiple platforms [28, 39, 46], in which trajectory operations built on-top of generic platforms, such as Hadoop or Spark. Using these platforms as-is results in sub-performance for trajectory operations that require indexing, e.g., Marmaray project from Uber uses Hadoop as a backbone for storing data as non-indexed heap files, or (b) use *Big Spatio-temporal Frameworks* [3, 50, 51] that are efficient for processing spatio-temporal data on MapReduce platform, yet, with a limited support for trajectory operations. This is mainly due to the inability of the index structure to accommodate storing the entire topology of trajectory objects. This in turn, affects the performance of basic trajectory operations, e.g., finding similarity between trajectories.

This paper presents Summit; a full-fledged open-source trajectory library for MapReduce framework, shipped with the source code of ST-Hadoop [3]. Summit injects the trajectory data awareness inside each of ST-Hadoop layers, mainly, indexing, operation, and language layers. Programs that deal with trajectory data using Summit will have up to order(s) of magnitude better performance than ST-Hadoop. The main reason is that ST-Hadoop treats the spatio-temporal information of trajectories as a set of spatio-temporal points or lines that are not connected together. This means that performing a basic trajectory operation such as similarity queries might end up scanning the whole dataset to check for trajectory connectivity before computing the similarity.

Figures 1a and 1b show code snippets that load and query similar trajectories from ST-Hadoop and Summit, respectively. The query finds similar trajectories within a specific rectangular area represented by two corner points and within a certain time interval. ST-Hadoop loads trajectories as spatio-temporal points, then it finds all records that overlaps with the time interval of interest. The retrieved records need to be grouped by their trajectory ID, and retrieved their entire trajectory sequence from secondary inverted index before starting to compute the similarity. This will incur significant I/O overhead, especially for a large spatio-temporal range. Meanwhile, Summit loads the entire sequence of trajectory and exploits its index to retrieve and compute the similarity between data records, and hence, achieves orders of magnitude better performance over ST-Hadoop.

```
Objects      = LOAD 'point' AS (id:int, STPoint);
Intermediate = FILTER Objects BY
               Overlaps ( Rectangle(x1, y1, x2, y2)
               ,Interval (t1, t2);
               GROUP Object BY (id)
               FOREACH $Object(id) Search Trajectory(id)
Result       = SIMILAR Object threshold:T From Intermediate;
```

(a) Similarity query in ST-Hadoop

```
Objects = LOAD 'trajectory' AS (id:int, STTrajectory:
          <STPoint1,STPointm >);
Result  = FILTER Objects BY
          Overlaps (Location,time,
           Rectangle(x1, y1, x2, y2),Interval (t1, t2))
          SIMILAR Object threshold:T;
```

(b) Similarity query in Summit

Figure 1: Similarity query in ST-Hadoop vs. Summit

2 BACKGROUND AND RELATED WORK

Distributed Generic Systems, Generic platforms have been used extensively in different analytic applications that include terabyte sorting [42], machine learning [23], The current effort for processing trajectory data are either: (A) Use *Heterogeneous* multiple platforms [10, 28, 39, 46], in which trajectory operations built on-top of generic platforms, such as Hadoop [6], Spark [8], Cassandra [5], Kafka [7], or Storm [9], e.g., Marmaray project from Uber uses Hadoop as a backbone for storing data as non-indexed heap files, while carrying the execution of trajectory operations on another platform. (B) Implement on-top of a single *Generic* framework [14, 18, 30, 41, 43, 51, 52]. For example, a most recent research study investigated the kNN join query on Hadoop MapReduce employed five isolated map-reduce jobs to execute a single kNN join operation without indexing trajectory [18]. However, using general purpose distributed systems as-is will result in sub-performance for trajectory applications that require indexing, mainly because they store data as non-indexed heap files.

Distributed Spatial Systems, Extension of MapReduce platform has been developed and dedicated for spatial analytic operations in the last few years, this including SpatialHadoop [15], ScalaGiST [29], Hadoop-GIS [2], and ESRI-GIS tool on Hadoop [16]. There are also few systems extended Resilient Distributed Dataset(RDD) on Spark to support spatial operations, such as

GeoSpark [57] and Simba [53]. Although, these big distributed spatial systems are efficient in spatial operations, yet they are not well designed to efficiently process spatio-temporal data such as trajectories, mainly because the infrastructure of their indexes and operations only support spatial queries.

Distributed Spatio-temporal Systems, Big spatio-temporal systems like ST-Hadoop [3] GeoWave [50], and GeoMesa [19] that focus on supporting spatio-temporal applications. ST-Hadoop extends Hadoop and maintains a Hierarchical indexing structure that consists of two-layer indexing of a temporal and spatial. GeoMesa and GeoWave both are built upon Accumulo platform [1] and implemented a space-filling curve to combine the three dimensions of space-geometry and time. This class of systems did not attempt to enhance the contiguity and locality of trajectory data. Although, distributed spatio-temporal systems are efficient for processing basic spatio-temporal data (e.g., POINT, LINE, POLYGON), yet, they are limited in supporting trajectory (i.e., a connected sequence of basic geometry). This is mainly due to the inability of their index structure to accommodate storing the entire topology of trajectory. This, in turn, will affect the performance of basic trajectory operations, e.g., finding similarity between trajectories.

Distributed Time Series Systems Frameworks in this category are optimized for time series or time-stamped data. There are a large number of distributed time series systems, such as Informix [25], TSAR [55], OpenTSDB [35], and LittleTable [38]. Systems in this family of distributed frameworks significantly have different approaches for handling data, such that they mainly focus on building indexes and operations to efficiently performs analytical tasks on time series data. There is no sufficient support for spatial or spatio-temporal indexes or operations, and hence they do not support analysis tasks on trajectory data.

Distributed Graph Processing Systems, MapReduce has been extensively investigated for graph processing in both academia and industry [22, 26, 27, 36, 44]. The focus of these systems is to support basic graph models, where a graph consists of a set of vertices and edges. Systems belong to this category are efficient for graph operations, such as complex traversal queries. However, they do not support spatio-temporal or trajectory operations mainly because their indexing structures are not well-suited for realizing both the spatial and the temporal property of trajectory.

Trajectory Operations, Existing research studies on trajectory implement operations on-top of distributed platforms, such as range [10, 14, 21, 30, 43, 54], k-nearest neighbor [14], similarity search [41, 52], and join [18, 51]. Most recent efforts focus on supporting similarity search on-top of Spark [41, 52] and *k*NN join on Hadoop [18], yet they do not have any indexes in Hadoop Distributed File Systems (HDFS). Notably, all these research studies are limited to support specific operation. In the meantime, the Summit system is extendable in a way that it enables users to build various applications on trajectories and extends its operations library.

Similarity Measurements, Measuring the similarity between a pair of trajectories is essential for identifying portions that are common between two trajectories. The similarity measurement must satisfy three main criteria: (1) The flexibility to identify similar trajectories on various times, (2) Ignores outliers points in similarity computation, and (3) The ability to identify the similarity between

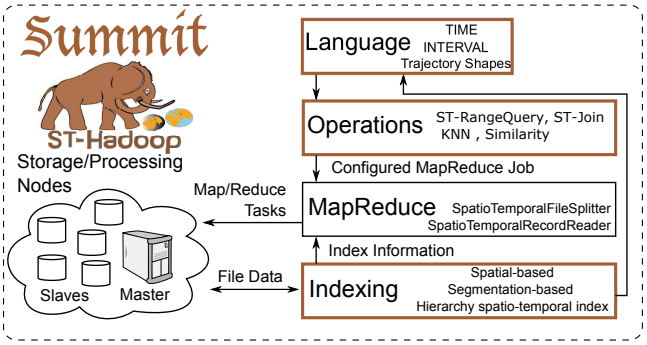


Figure 2: Summit Architecture

portions of trajectories based on some distance measurement. Formally, we can say that a similarity function takes pairs of trajectories and it generates a score, that shows the closeness between their sequences. There are over a dozen similarity measurements in literature like Dynamic Time Warping (DTW) [17, 41, 56], Edit Distance on Real sequence (EDR) [12], Edit distance with Real Penalty (ERP) [11, 20], Longest Common Subsequence distance (LCSS) [45, 47], Fréchet similarity [4]. Interested readers can refer to previous study [24], which discussed trajectory similarity in great detail. In this paper, we are going to consider the Dynamic Time Warping (DTW) measurement. Originally DTW developed for matching speech signals in speech recognition [37], ever since it is considered as the one of the most rubst and wildly adopted similarity function for trajectories and time series data [13, 48, 49].

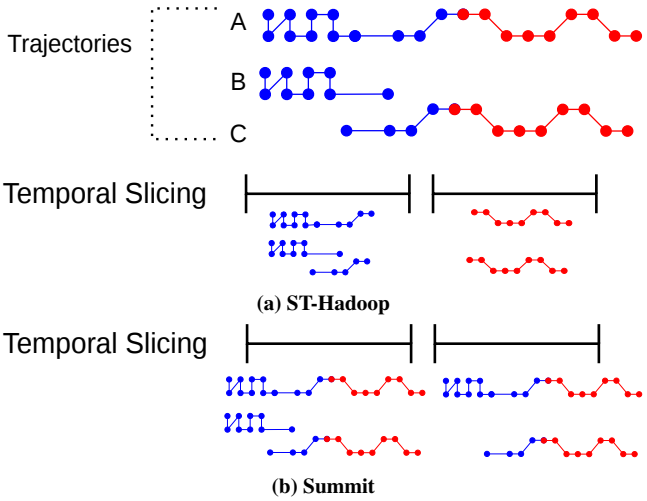
3 APPROACH AND UNIQUENESS

3.1 System Overview

Figure 2 gives an overview of Summit system architecture. Summit is a full-fledged open-source library on ST-Hadoop MapReduce framework [3] with *built-in* native support for trajectory data. Summit cluster contains one master node that breaks a map-reduce job into smaller tasks, carried out by slave nodes. Summit modifies three core layers of ST-Hadoop, namely, *Language*, *Indexing*, and *Operations*. The language layer adds new SQL-Like interface for trajectory operations and data types. The modifications and the implementation of the indexing and operation layers will be explained in the following sections.

3.2 Trajectory Indexing

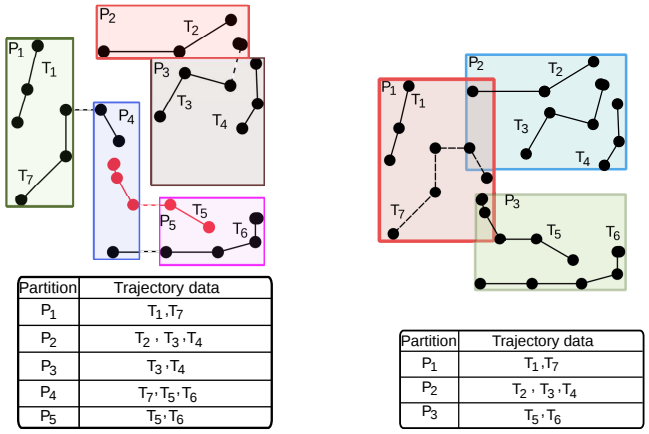
Input files in Hadoop Distributed File System (HDFS) are organized as heap files, where data is loaded into consecutive chunks, each of size 128MB. Though this was acceptable for analysis tasks that do not require indexing, it will result in sub-performance for applications, where indexing is essential. Recent efforts investigated in-memory indexing on Spark [41], yet it does not have any HDFS indices. In the meantime, spatiotemporally indexed HDFSs, as in ST-Hadoop [3, 50], are geared towards supporting queries with spatio-temporal predicates for basic geometrical shape, e.g., Point, Line, and Rectangle. On the other side, trajectories consist of a set of correlated sequence of spatio-temporal points, where ST-Hadoop is unable to realize the correlation between these sequences.



(a) ST-Hadoop

(b) Summit

Figure 3: Temporal Slicing



(a) Spatial-based

(b) Segmentational-based

Figure 4: Spatial Indexing

Summit organizes input files in HDFS in a way that preserves the geometrical topology of trajectories. In particular, data is spatiotemporally loaded and partitioned across computational nodes. Each partition holds the full sequence of trajectories that overlap with its spatio-temporal boundaries. Summit sacrifices storage to achieve higher performance by enforcing data replication across partitions. As a result, trajectory operations can have minimal data access to retrieve the query answer, reduce the computation complexity, and allow applications to run more sophisticated operations on the entire trajectories.

Summit employs a two-level indexing scheme of temporal indexing followed by a spatial one. The index is stored in the master node as auxiliary file, while actual partitions are divided across computation nodes. The process of index construction in Summit goes through the following three consecutive phases:

1. Sampling: The objective of sampling is to approximate the trajectory distribution and ensure the quality of partitioning. Due to the mass volume of data, Summit scans a representative sample that fits-in the main memory of the master node.

2. Bulkload Partitioning: Summit manipulates the sample to construct boundaries of the two-level indexing of temporal and spatial, respectively. System parameters in a configuration file guide the indexing of each level, such as the temporal granularity and the spatial partitioning mechanism. Expert users and data practitioners that have good understanding of Summit and the nature of their datasets can tune these parameters. The construction of the two-level indexing scheme goes through two main steps:

(1) *Temporal Slicing*: Figure 3 depicts the abstract idea of temporal slicing in Summit in comparison to ST-Hadoop. The temporal slicing mechanism in ST-Hadoop breaks trajectory into sub-sequences. Meanwhile, Summit slicing replicates trajectories if they overlap between temporal slices while maintaining non-overlapping disjoint. As shown in figure 3b, the lifetime of trajectory `A` overlaps both the first and second slices, and thus, the entire trajectory will be replicated between those two temporal slices. As opposed to ST-Hadoop in figure 3a where `A` chopped into two sub-sequences, each stored separately in a different temporal partition.

(2) *Spatial Indexing*: Summit is equipped with the following two spatial indexing approaches for each temporal slice from the previous step, namely *Spatial-based* or *Segmentation-based*. Figure 4 illustrates the logical design of both methods, where rectangles represent the boundaries of the HDFS partitions while dots and lines depict the trajectory information. (a) *Spatial-based*: This approach preserves the spatio-temporal locality closeness between sub-trajectories. The boundaries of the HDFS partition split trajectories as shown in figure 4a. (b) *Segmentation-based*: This is a data partitioning that guarantees that the entire trajectory is stored in a single HDFS block, as shown in figure 4b. The minimum boundaries rectangles of this index might overlap. When a trajectory intersects with more than a single rectangle, its going to be replicated between partitions. This partitioning is more in favor of operations that not only need to process the locality of trajectories but also their semantic or shapes over time, such as Similarity *k*NN and join queries.

3. Physical Assigning: The objective of this phase is to scan through the whole data and assign each record to the boundaries layout constructed from the previous phase. Summit initiates a map-reduce job that scans through the input file, physically partitions HDFS block, and assigns records to all overlapping partitions.

3.3 Operations

In this paper, we discuss the internal execution of three basic operations in Summit, namely, range, nearest neighbor, and similarity queries. Other spatio-temporal operations on trajectories, e.g., reverse *k*NN, aggregation, and path queries, can be realized following similar approaches.

• **Trajectory Range Query (TRQ):** Given a three-dimensional query predicate, this query retrieves all trajectories that overlap with the query region in both space and time. For example, "*Find all taxis in downtown Manhattan between January and March 2019*". Regardless of the type of partitioning to answer the query, we employ an algorithm that runs in three steps namely, temporal filtering, spatial search, and spatio-temporal refinement. In the refinement phase, an extra processing is required to remove duplicates from the query answer, as trajectories might be replicated between partitions.

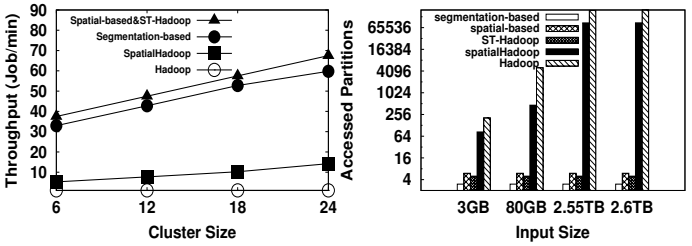
• **Trajectory *k* Nearest Neighbor Query (T*k*NN):** Summit supports the following two variants of the *k*NN operation: (1) ***k*NN point-based.** Given a query predicate that consists of query point $P_{(x,y)}$, and time interval $[t_1, t_2]$, find the k nearest trajectories to the query point during the given time interval. For example, "*Find the closest four animals to a Minnehaha waterfall between August and September*". (2) ***k*NN trajectory-based.** Given a query trajectory Trj that consists of a sequence of spatio-temporal points, find the *k*NN trajectories to the whole trajectory points for every time instance according to some aggregate function, such as Min or Max. For example, "*Find the closest two human traveled along a contaminated water stream in Jun 2018*". To answer both queries, Summit employs an algorithm that consists of three phases, namely, partitioning, local computation, and global computation. In the partitioning phase, Summit decides which partition technique will be used. Once data is partitioned, Summit triggers a local computation algorithm to find a candidate set from the overlapping partitions. After performing a local computation, each computation node in Summit cluster will have its own candidate set. The global computation phase is implemented in Summit as a reduce function, which runs on a single machine to compute the final result. Duplicate elimination is applied in this phase.

• **Trajectory Similarity Query (TSQ):** The objective of this query is to find similar trajectories to a given one based on some defined similarity function. This is a very useful query for many applications, such as transportation and advance pattern mining queries. A typical example of such queries is, "*Find the k taxis that share similar routes with a given trajectory (e.g., another taxi) during some time interval (e.g., yesterday)*". Summit goes through two phases to find similar trajectories, namely, partitioning and computation phases. The partitioning phase indexes data by segmentation-based model. The computation phase runs in two back-to-back map-reduce tasks for local and global computation. Duplicate removal takes place in the reduce phase before starting global computation. In Summit, we implemented the most robust and widely adopted similarity function, i.e., the Dynamic Time Warping [13], where we apply spatio-temporal thresholds. Other similarity measurements can be realized following the same approach.

4 RESULTS AND CONTRIBUTIONS

We compare Summit prototype against the state-of-the-art of various distributed frameworks [3, 6, 15]. In our experiments, each system runs on a cluster of 24 machines, and we use large real datasets of NYC taxi [34], which contains over one billion records. Each trip in NYC data contains GPS coordinates associated with time for both pickup and drop-off. To resemble the path of taxis' trips, we generated the full sequence between starts and ends of all trips using Dijkstra's algorithm on New York road network.

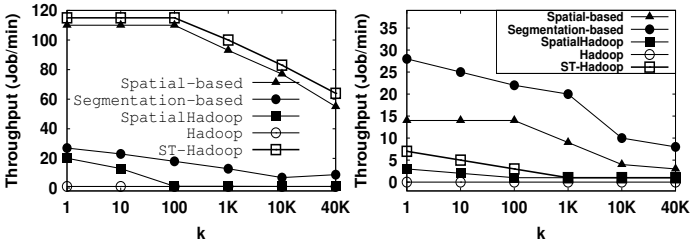
In this paper, we show a case study for each of Summit operations. To experiment with different dataset sizes in our experiments, we divided the dataset into disjoint sets, by default the whole dataset is used unless mentioned. For a range query, we measure the job throughput as the performance metric, which is the average batch of 30 queries completed per minute. For the *k*NN and similarity queries, thirty locations are set at random points (i.e., random points in both date and time) sampled from the whole input file, the default k is set to 100. Unless otherwise mentioned.



(a) Cluster Size **(b) Accessed Patitions**

Figure 5: Range Query

• **Trajectory Range Query (TRQ):** Figure 5a, shows how Summit range query scales out with cluster size changing from 6 to 24 node. Summit, ST-Hadoop, SpatialHadoop, and Hadoop are smoothly scale with cluster size, while Summit is consistently more efficient than others. Figure 5b, analyzed the maximum number of accessed partitions of the submitted 30 queries. The queries are not overlapping and are randomly selected with a spatial area ratio of 0.005% of New York City and a temporal window of one month. Hadoop scans all partitions for any input file. Due to the distribution of the data in limited geographical space, SpatialHadoop it slightly performs better but still it access more than 40% of the input file. On the other hand, the number of accessed partitions in Summit and ST-Hadoop remains stable as they only process a fixed spatiotemporal area of the input file.

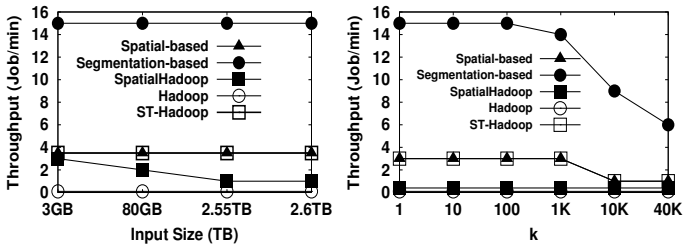


(a) *k*-Point-based **(b) *k*-trajectory-based**

Figure 6: *k*NN Query

• **Trajectory *k* Nearest Neighbor Query (T*k*NN):** In Figures 6, we study the effect of increasing k from 1 to 40K on the entire dataset for both *k*NN point-based and trajectory-based, respectively. Summit gives an order of magnitude performance with its trajectory partitioning techniques. Summit spatial-based partitioning outperforms others in case of *k*NN-point-based query. In the meantime, the segmentation-based partitioning in Summit achieves higher job throughput compared to other systems *k*NN-trajectory-based implementation. However, we notice that the job throughput decreases when k is slightly around and more than a thousand. This is expected as increasing the number of k requires more processing and partitions to be processed.

• **Trajectory Similarity Query (TSQ):** Figure 7a measures systems job throughput with increasing the input file size. Summit segmentation-based has a higher job throughput than other systems. There is no significant difference between ST-Hadoop and Summit Spatial-based segmentation, as they require extra processing steps to resemble the full sequence of trajectories. In the meantime, Summit segmentation-based keeps its speedup at two orders of magnitude, in which it maintains trajectories' sequences locality.



(a) File Size **(b) k**

Figure 7: DTW similarity Query

Figure 7b gives the effect of increasing k on the entire dataset. Summit gives an order of magnitude performance with segmentation-based indexing compared to other systems on-top implementations of DTW. In contrary, the spatial-based analyzed a subset of trajectories. Hence, to compute the full trajectory an extra processing step required to resemble trajectories' sequences. Similarly ST-Hadoop and SpatialHadoop require extra step to search for full sequence of trajectories. This will incur significant I/O overhead, especially for a sizable spatio-temporal range.

REFERENCES

[1] Accumulo 2019. https://accumulo.apache.org. accessed on April.

[2] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel H. Saltz. 2013. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *PVLDB* 6, 11 (2013), 1009–1020.

[3] Louai Alarabi, Mohamed F. Mokbel, and Mashaal Musleh. 2018. ST-Hadoop: A MapReduce Framework for Spatio-temporal Data. *GeoInformatica* 22, 4 (2018), 785–813.

[4] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.* 5 (1995), 75–91.

[5] Apache. Cassandra 2019. http://cassandra.apache.org. accessed on April.

[6] Apache. Hadoop 2019. http://hadoop.apache.org. accessed on April.

[7] Apache. KafKa 2019. https://kafka.apache.org. accessed on April.

[8] Apache. Spark 2019. http://spark.apache.org. accessed on April.

[9] Apache. STORM 2019. http://storm.apache.org. accessed on April.

[10] Jie Bao, Ruiyuan Li, Xiuwen Yi, and Yu Zheng. 2016. Managing massive trajectories on the cloud. In *SIGSPATIAL*. 41:1–41:10.

[11] Lei Chen and Raymond T. Ng. 2004. On The Marriage of Lp-norms and Edit Distance. In *VLDB*. 792–803.

[12] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*. 491–502.

[13] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB* 1, 2 (2008), 1542–1552.

[14] Xin Ding, Lu Chen, Yunjun Gao, Christian S. Jensen, and Hujun Bao. 2018. UlTraMan: A Unified Platform for Big Trajectory Data Management and Analytics. *PVLDB* 11, 7 (2018), 787–799.

[15] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE*. 1352–1363.

[16] ESRI-GIS 2014. GIS Tools for Hadoop. http://esri.github.io/gis-tools-for-hadoop/.

[17] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast Subsequence Matching in Time-Series Databases. In *SIGMOD*. 419–429.

[18] Yixiang Fang, Reynold Cheng, Wenbin Tang, Silviu Maniu, and Xuan S. Yang. 2016. Scalable Algorithms for Nearest-Neighbor Joins on Big Trajectory Data. *TKDE* 28, 3 (2016), 785–800.

[19] Anthony D. Fox, Christopher N. Eichelberger, James N. Hughes, and Skylar Lyon. 2013. Spatio-temporal indexing in non-relational distributed databases. In *BIGDATA*. 291–299.

[20] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. 2007. Index-based Most Similar Trajectory Search. In *ICDE*. 816–825.

[21] Zdravko Galic, Emir Meskovic, and Dario Osmanovic. 2017. Distributed processing of big mobility data as spatio-temporal data streams. *GeoInformatica* 21, 2 (2017), 263–291.

[22] Jun Gao, Jiashuai Zhou, Chang Zhou, and Jeffrey Xu Yu. 2014. GLog: A high level graph analysis system using MapReduce. In *ICDE*. 544–555.

[23] Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan. 2011. SystemML: Declarative Machine Learning on MapReduce. In *ICDE*. 231–242.

[24] Joachim Gudmundsson, Patrick Laube, and Thomas Wolle. 2012. Computational Movement Analysis. In *Die Dynamik sozialer und sprachlicher Netzwerke, Konzepte, Methoden und empirische Untersuchungen an Beispielen des WWW*. 423–438.

[25] Sheng Huang, Yaoliang Chen, Xiaoyan Chen, Kai Liu, Xiaomin Xu, Chen Wang, Kevin Brown, and Inge Halilovic. 2014. The next generation operational data historian for IoT based on informix. In *SIGMOD*. 169–176.

[26] janusgraph 2019. https://janusgraph.org. accessed on April.

[27] Wenqing Lin, Xiaokui Xiao, and Gabriel Ghinita. 2014. Large-scale frequent subgraph mining in MapReduce. In *ICDE*. 844–855.

[28] Jiamin Lu and Ralf Hartmut Guting. 2012. Parallel Secondo: Boosting Database Engines with Hadoop. In *ICPADS*. 738–743.

[29] Peng Lu, Gang Chen, Beng Chin Ooi, Hoang Tam Vo, and Sai Wu. 2014. ScalaGiST: Scalable Generalized Search Trees for MapReduce Systems. *PVLDB* 7, 14 (2014), 1797–1808.

[30] Qiang Ma, Bin Yang, Weining Qian, and Aoying Zhou. 2009. Query Processing of Massive Trajectory Data Based on MapReduce. In *CLOUDDB*. 9–16.

[31] MoveBank 2019. Wikelski, M., and Kays, R. Movebank:s archive, analysis and sharing of animal movement data. Hosted by the Max Planck Institute for Ornithology. www.movebank.org. accessed on April.

[32] NASA Spacecraft projects 2019. Data from NASA's Missions, Research, and Activities. http://www.nasa.gov/open/data.html. accessed on April.

[33] NHC 2019. The National Hurricane Center. www.nhc.noaa.gov. accessed on April.

[34] NYC 2019. Data from NYC Taxi and Limosuine Commission. http://www.nyc.gov/html/tlc/. accessed on April.

[35] OpenTSDB 2019. OpenTSDB. http://opentsdb.net. accessed on April.

[36] Lu Qin, Jeffrey Xu Yu, Lijun Chang, Hong Cheng, Chengqi Zhang, and Xuemin Lin. 2014. Scalable big graph processing in MapReduce. In *SIGMOD*. 827–838.

[37] Lawrence R. Rabiner and Biing-Hwang Juang. 1993. *Fundamentals of speech recognition*. Prentice Hall.

[38] Sean Rhea, Eric Wang, Edmund Wong, Ethan Atkins, and Nat Storer. 2017. LittleTable: A Time-Series Database and Its Uses. In *SIGMOD*. 125–138.

[39] Sijie Ruan, Ruiyuan Li, Jie Bao, Tianfu He, and Yu Zheng. 2018. CloudTP: A Cloud-Based Flexible Trajectory Preprocessing Framework. In *ICDE*. 1601–1604.

[40] SDSS 2019. Sloan Digital Sky Survey. http://www.sdss.org/dr14/data_access/volume/. accessed on April.

[41] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-Memory Trajectory Analytics. In *SIGMOD*. 725–740.

[42] Sort Benchmark 2019. http://sortbenchmark.org. accessed on April.

[43] Haoyu Tan, Wuman Luo, and Lionel M. Ni. 2012. CloST: a hadoop-based storage system for big spatio-temporal data analytics. In *CIKM*. 2139–2143.

[44] titan 2019. http://titan.thinkaurelius.com. accessed on April.

[45] Kevin Toohey and Matt Duckham. 2015. Trajectory similarity measures. *SIGSPATIAL Special* 7, 1 (2015), 43–50.

[46] Uber. Marmaray 2019. https://github.com/uber/marmaray. accessed on April.

[47] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002. Discovering Similar Multidimensional Trajectories. In *ICDE*. 673–684.

[48] Haozhou Wang, Han Su, Kai Zheng, Shazia Wasim Sadiq, and Xiaofang Zhou. 2013. An Effectiveness Study on Trajectory Similarity Measures. In *ADC*. 13–22.

[49] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. 2013. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.* 26, 2 (2013), 275–309.

[50] Michael A. Whitby, Rich Fecher, and Chris Bennight. 2017. GeoWave: Utilizing Distributed Key-Value Stores for Multidimensional Data. In *SSTD*. 105–122.

[51] Randall T. Whitman, Michael B. Park, Bryan G. Marsh, and Erik G. Hoel. 2017. Spatio-Temporal Join on Apache Spark. In *SIGSPATIAL*. 20:1–20:10.

[52] Dong Xie, Feifei Li, and Jeff M. Phillips. 2017. Distributed Trajectory Similarity Search. *PVLDB* 10, 11 (2017), 1478–1489.

[53] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *SIGMOD*. 1071–1085.

[54] Xike Xie, Benjin Mei, Jinchuan Chen, Xiaoyong Du, and Christian S. Jensen. 2016. Elite: an elastic infrastructure for big spatiotemporal trajectories. *VLDBJ* 25, 4 (2016), 473–493.

[55] Peilin Yang, Srikanth Thiagarajan, and Jimmy Lin. 2018. Robust, Scalable, Real-Time Event Time Series Aggregation at Twitter. In *SIGMOD*. 595–599.

[56] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. 1998. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *ICDE*. 201–208.

[57] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. GeoSpark: a cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*. 70:1–70:4.

[58] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. 2014. Urban Computing: Concepts, Methodologies, and Applications. *ACM TIST* 5, 3 (2014), 38:1–38:55.