A Survey on Big Data Processing Frameworks for Mobility Analytics

Christos Doulkeridis[1], Akrivi Vlachou[2], Nikos Pelekis[3], Yannis Theodoridis[4]
[1]Department of Digital Systems, University of Piraeus, Greece
[2]Information & Communication Systems Engineering, University of Aegean, Greece
[3]Department of Statistics and Insurance Science, University of Piraeus, Greece
[4]Department of Informatics, University of Piraeus, Greece
{[1]cdoulk, [3]npelekis, [4]ytheod}@unipi.gr, [2]avlachou@aegean.gr

ABSTRACT

In the current era of big spatial data, the vast amount of produced mobility data (by sensors, GPS-equipped devices, surveillance networks, radars, etc.) poses new challenges related to mobility analytics. A cornerstone facilitator for performing mobility analytics at scale is the availability of big data processing frameworks and techniques tailored for spatial and spatio-temporal data. Motivated by this pressing need, in this paper, we provide a survey of big data processing frameworks for mobility analytics. Particular focus is put on the underlying techniques; indexing, partitioning, query processing are essential for enabling efficient and scalable data management. In this way, this report serves as a useful guide of state-of-the-art methods and modern techniques for scalable mobility data management and analytics.

1. INTRODUCTION

Nowadays, the ever-increasing rate of mobility data generation has resulted in vast volumes of spatio-temporal data, thus leading to new challenges for scalable processing and analysis of big mobility data. Interestingly, this applies to different domains of everyday life, from urban, to marine, and even further to air-traffic management. Miscellaneous applications and systems, such as surveillance networks, sensor readings on moving objects, human-related mobile data, social activity in location-based social networks, produce and gather positional data at rapid rates and at global scale.

In tandem with this explosion of mobility data, management of big data raises numerous research challenges [34] in different phases of the big data processing and analysis pipeline, including: (a) data acquisition, (b) data pre-processing and cleaning, (c) data integration, aggregation, and representation, (d) modeling and analysis, and (e) interpretation. The modern trend for scalable storage of massive datasets is by means of a NoSQL store [13, 16]. The exact choice depends on numerous parameters, including the type of data, the data access patterns, the purpose of data processing (read/write, read-only, etc.), as well as any special requirements with respect to the Consistency, Availability, and Partition-tolerance (also known as CAP).

Also, the current landscape of big data management comprises multiple frameworks targeting different aspects of big data. One major separating line is drawn between frameworks for batch and real-time processing, although lately some systems have been designed to tackle both cases. In the batch processing domain, Spark [65] is one of the most popular solutions nowadays with a large and growing user-base. However other solutions, such as Flink [12], are also applicable with success. In particular, Spark has successfully addressed many of the limitations of Hadoop [18], and operates in main-memory by its core abstraction: RDDs (Resilient Distributed Datasets) [64]. In the real-time processing domain, the most notable systems in use today are Storm [53] and Flink [12].

This paper provides an overview of the state-of-the-art in big data storage and processing, focusing primarily on *scalable solutions for mobility data*, i.e., spatial but most importantly spatio-temporal data. Despite the rich literature on management of spatio-temporal and mobility data, only a limited number of research prototypes attempt to address this problem in the context of big data, while most evaluations and benchmarks focus mostly on big spatial data [3, 22, 29], rather than spatio-temporal data [40]. The majority of developed prototypes extend Hadoop or Spark in order to be applicable for spatial data. In this survey, we also cover big data approaches that handle the temporal dimension.

The remaining of this survey is structured as follows: Section 2 provides background concepts related to spatio-temporal and mobility data. In Section 3, we present typical partitioning techniques

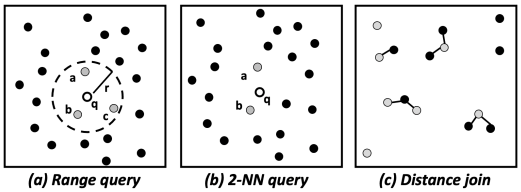

Figure 1: Basic spatial query types.

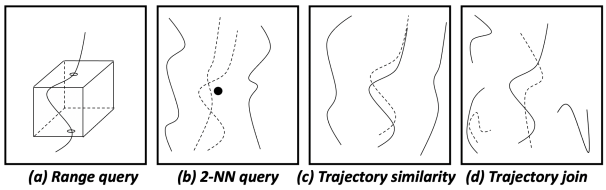

Figure 2: Basic trajectory queries.

used for mobility data. Then, in Section 4, we describe distributed indexing techniques for big spatial and spatio-temporal data. Section 5 provides an overview of query processing, focusing on range and k-NN queries as well as joins. Section 6 classifies existing storage systems and processing frameworks based on the underlying techniques that were presented in the previous sections. Finally, we conclude the paper in Section 7 and sketch future research directions.

2. BACKGROUND

In this section, we provide some basic background concepts related to query types for spatial, spatio-temporal and trajectory data.

Figure 1 depicts the most basic spatial query types for spatial point data. Obviously, these queries can be generalized for other types of spatial objects, such as polygons. In Figure 1(a), a range query is depicted which is defined by a query point q and a radius r, and retrieves all objects within distance r from q (in this example: $\{a, b, c\}$). Other ways to express the spatial constraint also exist, e.g., as a 2D box instead of a circle, but the concept remains the same. Figure 1(b) shows the case of a k-nearest neighbor (k-NN) query, defined by a point q and an integer k (in this example, the 2-NN of q are: a and b). In Figure 1(c), the case of a distance join between two data sets is depicted, where the result is pairs of objects from the two data sets that are within a user-specified distance.

Extending these queries for spatio-temporal data points is straightforward by adding time as third dimension to the query. In the case of k-nearest neighbors, different options exist, such as retrieval of the spatially k nearest objects that satisfy a temporal constraint, or the k temporally closest objects that satisfy a spatial constraint.

Figure 2 shows basic trajectory queries. On the left, a spatio-temporal range query for trajectories is depicted, which retrieves all portions of trajectories inside a spatial region during a temporal interval. Then, a k-NN query is shown, which retrieves the 2 trajectories closest to a given point. In Figure 2(c), a trajectory similarity query is depicted which, given a trajectory similarity function, retrieves the most similar trajectory to a given query trajectory. Variants of this query can use a distance threshold on similarity or retrieve the k most similar trajectories. Lastly, in Figure 2(d), the case of trajectory join is shown, where two data sets of trajectories are given, and the task is to identify pairs of trajectories that satisfy a condition (typically expressed as similarity constraint).

3. PARTITIONING TECHNIQUES

Data partitioning is the key technique for achieving efficient parallel processing of mobility data. Partitioning techniques for big mobility data have the following distinguishing features: (a) they operate on a sample of data in order to produce partitions for the complete data set, (b) they need to cope with skewed data distributions, (c) they should be adaptive both with respect to changing data distributions as well as changes in the query workload, (d) they need to balance the workload to the available nodes, which is further complicated by object duplication to nearby partitions.

3.1 Spatial and Spatio-temporal Partitioning

Partitioning techniques for the 2D space include partitioning based on Grid, STR (sort-tile-recursive), Quadtree, k-d tree, as well as mapping to 1D values using space-filling curves followed by 1D partitioning. All partitioning techniques for spatial data can also be applied for spatio-temporal data, if we consider time as another dimension. However, some frameworks for big spatio-temporal data organize data based on temporal partitions, which are further partitioned in the 2D space. As an example, ST-Hadoop [4, 6] follows this approach.

Grid partitioning. This is a standard space partitioning technique that splits the underlying space in non-overlapping cells. Several frameworks use grid partitioning, including SpatialHadoop [21], SpatialSpark [60], and GeoSpark [61].

It should be noted that sometimes the partition-

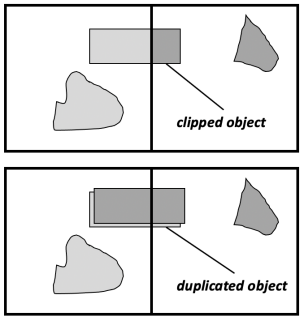

Figure 3: Object duplication vs. object clipping.

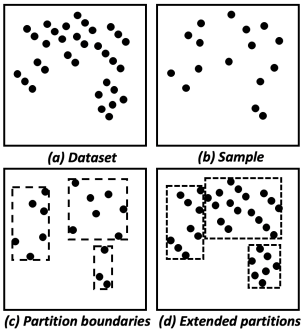

Figure 4: Sample-based data partitioning, followed by extension of partition boundaries.

ing step is followed by object duplication to neighboring cells. This is typically the case for distance joins over point data or spatial joins over data with extent. Also, in the latter case, another alternative is to perform object clipping, thus separating an object to multiple parts and assign each part to a different cell, as shown in Figure 3.

STR partitioning. A widely used partitioning scheme adopted by several prototypes (Simba [58, 59], SpatialHadoop [21], DITA [49], UlTraMan [17]) is Sort-Tile-Recursive (STR) [35], which is considered one of the best partitioning schemes for spatial data. For example, in Simba [58, 59], random sampling is performed over the input data, and then the first iteration of STR is executed to produce partition boundaries. Obviously, these partitions may not cover the entire data space, as they have been constructed based on a sample only, therefore they need to be extended to cover the entire data space, as shown in Figure 4.

R*-Grove [55] has been recently proposed for spatial partitioning, aiming to address some limitations of pre-existing partitioning schemes, such as STR. Its core idea is to use the node split algorithm of R*-tree in order to create compact, square-like partitions, in contrast to the thin and wide partitions that are often produced by STR. In addition, R*-Grove adopts a load-balancing mechanism that forces the generation of full blocks.

Quadtree partitioning. Other prototypes support Quadtree-based partitioning on a data sample as an alternative technique. Again, the aim is to produce partitions that take into account the (inferred) data distribution, and handle skewed spatial distributions gracefully. This approach is supported by frameworks such as SpatialHadoop [21], LocationSpark [52], and STJoins@ESRI [56, 57].

K-d tree partitioning. Another approach to handle skewness of input data is to use a k-d tree, in which the leaves correspond to data partitions in the distributed file system. This approach is adopted by AQWA [7] and some statistics are maintained in main memory, in order to capture the distribution of data. Furthermore, AQWA adopts an adaptive mechanism for partitioning aiming to handle changes in the query workload, where a partition may be further decomposed in case of updated data distribution or queries. Other frameworks that support partitioning based on k-d trees include SpatialHadoop [21] and SpatialSpark [60].

Partitioning based on space-filling curves. In this approach, the 2D data is mapped in 1D values using a space-filling curve (such as Z-order or Hilbert curve), and then partitions are generated by grouping the 1D values into intervals. SpatialHadoop [21] supports this type of partitioning. Also, this partitioning scheme is popular in big data storage systems, such as MD-HBase [42], Pyro [36], and QUILTS [43].

3.2 Trajectory Partitioning

Some frameworks for trajectory data management also adopt partitioning techniques such as those described above. However, other specialized partitioning techniques are also employed. For example, HadoopTrajectory [9] supports both partitioning per moving object (so as to have the complete trajectory in the same partition), as well as spatio-temporal partitions. These partitions must be small in order to avoid accessing trajectories that do not match with the query, but also large enough in order to avoid splitting trajectories into multiple partitions.

Finally, a different approach is used by DITA [49], where the STR algorithm is used, but it operates on selected points of trajectories, namely the first and last points of each trajectory. The trajectories

are grouped based on their first points, and then subgroups are created by grouping based on the last points. Intuitively, this partitioning technique aims to group together trajectories with similar starting positions and similar ending positions.

4. DISTRIBUTED INDEXING

The basic idea behind distributed indexing, which is adopted by most existing prototypes and systems, is to employ a two-level indexing scheme.

At the local level, index structures such as R-trees, Quadtrees, and Grids are typically used. An alternative approach is to map data to 1D values using space-filling curves and use traditional B-trees for local indexing. This latter approach is widely adopted by NoSQL stores. In the case of spatio-temporal data, some approaches index first the temporal dimension, and then the spatial dimensions.

At the global level, the most common approach is to assemble summary information from the local indexes of nodes, in order to build a global index for directing queries to nodes. Essentially, this summary is partition boundary information, such the Minimum Bounding Rectangles (MBRs) that describe the local data on each node. This is depicted in Figure 5, which presents the approach adopted by Simba [58, 59].

4.1 Spatial and Spatio-temporal Indexing

The combination of local and global indexing is used by several frameworks for big spatial data processing. Indicative examples of such frameworks include Hadoop-GIS [2], SpatialHadoop [21] and LocationSpark [52]. For the local indexes, classic 2D data structures are employed: R-tree, Quadtree, as well as Grid.

In the case of spatio-temporal indexing, one approach is to first organize data based on time, and then based on space. ST-Hadoop [4, 6] builds a temporal hierarchy of spatial indexes. This approach favors queries with high selectivity in the temporal dimension. Other approaches handle the three dimensions equally and build spatial indexes in the 3D space. STARK [30] uses R-trees to index spatio-temporal data, by following this idea.

4.2 Indexing Trajectory Data

In the case of trajectory data, one indexing approach is to employ the afore-mentioned solutions for 3D spatio-temporal data. As an indicative example, HadoopTrajectory [9] follows this approach and builds a grid in the 3D space or a 3DR-tree.

However, more specialized indexing techniques tailored for trajectory data are also used. DITA [49] uses global/local indexes, but proposes an approximate representation technique for trajectories based on pivot points. Two indexes are built, one for the first points of trajectories, and another one for the last points.

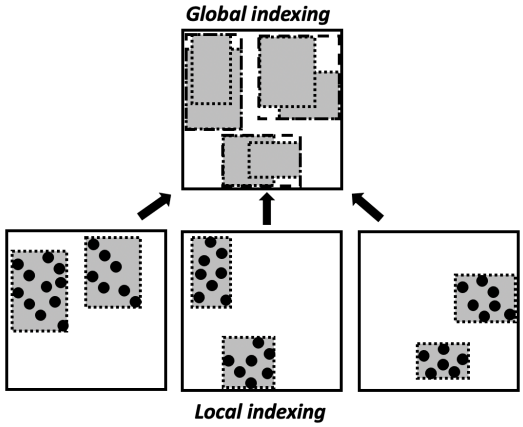

Figure 5: Example of global/local indexing.

5. QUERY PROCESSING

5.1 Big Spatial and Spatio-temporal Data

In the following, we review query processing techniques for the most standard query types for big mobility data under the global/local indexing scheme.

Range queries. Range queries comprise the most standard query type supported by all prototypes and systems. In a distributed system, processing a range query typically starts at the level of global indexing, where the partitions that overlap with the query range are identified. Then, each of these partitions is queried in parallel using its local index, and the local results are collected and returned to the user.

Nearest-neighbor queries. Typically, nearest-neighbor queries can be processed as range queries, as long as a good radius can be estimated that is guaranteed to include the k nearest neighbors. Ideally, if the distance to the k-th nearest neighbor was known in advance, we would retrieve exactly the k nearest neighbors. Consequently, the major challenge is accurate radius estimation for the query range. This issue relates to selectivity estimation for spatial queries [14, 15]. Also, in a distributed setting, the range query may intersect with multiple partitions that belong to different nodes in the system, which need to process the query, thus increasing the cost of query execution.

This approach is followed in AQWA [7], where

given a query q the surrounding cells are visited in increasing order of minimum distance (MinDist). As soon as the aggregate count of objects in the visited cells reaches k, the largest MaxDist of these cells is used as query radius.

Simba [58, 59] attempts to improve the tightness of the estimated query radius, by following a two-step approach. In the first step, the nearest partitions to q that are guaranteed to contain k points are actually queried, in order to find the k-nearest neighbors of each partition. Assuming l such partitions, then Simba uses the $l \cdot k$ candidates in order to compute the k-th minimum distance from q, and uses this value as query radius in the second step in order to ensure that the k-nearest neighbors are found. Essentially, Simba computes a tighter radius, at the cost of processing a k-nearest neighbor query locally over few partitions.

Joins. For an elaborate survey on spatial join processing, we refer to [33]. In Hadoop-based big data systems, such as SpatialHadoop [19, 21] and Hadoop-GIS [2], spatial joins are processed using the following approach: first, one data set is sampled and an in-memory index is built using the sample. Then, the leaf nodes of the index are mapped to HDFS blocks, which now contain data with spatial locality. Finally, objects are assigned to HDFS blocks based on the MBR of the block, and the join is performed between blocks, since pairs of blocks from each relation have the same MBR.

A different approach is applied in the case that data is stored without a spatial partitioning method. Quite often, a Grid-based structure is used in order to re-partition data to grid cells, followed by local join processing in each cell in parallel. To guarantee that each partition can be processed independently, one must handle the case of spatial objects that may join with objects in other cells, therefore object duplication to such cells is performed. For example, this is the approach followed in GeoSpark [61, 62]. Variations of this method include the use of data structures that are data-aware (e.g., dynamic Grid, Quadtree, R-tree, etc.) and use of leaf nodes as cells. Such data structures can typically cope better with skewed data distributions. LocationSpark [51, 52] follows a similar approach where a sample is taken from the first input, followed by the construction of a global spatial index. Then, workers re-partition their data based on the leaf nodes of the global index. When the join is processed, data from the second input are sent to the corresponding overlapping partitions, in order to produce join results locally. Notice that the overlapping partitions depend on the type of the join and the type of spatial objects. In [51], LocationSpark also reports a method for detecting skewed partitions and splitting them to smaller partitions that are repartitioned, in order to reduce the execution time.

More complex joins, as for instance k-NN joins, have also been studied in a MapReduce context, for example in [38] using Voronoi partitioning, and in [66] using Z-order and resulting in an approximate algorithm. Later, in Simba [59], a centralized processing step is used that exploits the partitions of the first dataset (built using STR), an R-tree built over a sample of the second dataset, in order to transform the join to n local k-NN joins. DISON [63] is a Spark-based approach for distributed similarity search and join for trajectory data in road networks, which computes signatures for trajectories that are used in a filter-and-refine framework. Finally, comparisons between the different systems for different join types are also of interest, e.g., for distance joins [26].

5.2 Big Trajectory Data

The most prominent and generic works in this field include HadoopTrajectory [9], DITA [49], UlTraMan [17], and MobilityDB [67], which are reviewed in Section 6.

Fang et al. [23] address the problem of k-NN join by using the MapReduce framework. More specifically, given two sets of trajectories R and S, an integer k and a time interval $[t_s, t_e]$, the objective is to return the k nearest neighbors from R for each object in S during this interval. In order to achieve this, a five step procedure (five MR jobs) is adopted, where the data are preprocessed, subtrajectories are extracted, the time-dependent upper bound is computed, candidates are found and the trajectories are joined. The intuition is to find a distance upper bound d for each trajectory of S, that includes at least k trajectories from R and then perform a plane sweep distance join based on d.

Tampakis et al. [50] study a more generic problem of sub-trajectory join, where the aim is to retrieve maximal portions of similar trajectories. Their most efficient algorithm uses a MapReduce job as preprocessing step in order to repartition data in balanced partitions based on the temporal dimension, followed by a second MapReduce job that produces the maximal sub-trajectories that join.

There is also work on joins over objects moving on road networks [47, 48]. The objective is to find all pairs of network-constrained trajectories that exceed a similarity threshold in a parallel manner. In [47], a two-phase algorithm is proposed that is parallelized and computes for each trajectory other

similar trajectories in its first phase. Then, during the second phase, it performs result merging in order to deliver the final result. However, the parallelization adopted is per trajectory, which assumes that all data need to be replicated for each trajectory, a fact that makes such a solution hard to apply in a big data setting.

6. SYSTEMS & FRAMEWORKS

In this section, we classify existing systems and frameworks for big spatial and spatio-temporal data processing. We review systems for scalable storage (Section 6.1) separately from big data processing frameworks (Section 6.2). The reason for this distinction is that although storage systems support (basic) processing, they offer only limited querying capabilities. For instance, join processing is not supported by the storage systems, whereas big data processing frameworks can compute parallel joins be re-partitioning data across nodes.

6.1 Scalable Storage

Systems that extend scalable storage solutions for multidimensional data have been proposed, most notably MD-HBase [42], but also solutions tailored specifically for spatio-temporal data (Pyro [36]) and for spatio-temporal RDF data [54], as well as spatio-textual data (ST-HBase [39]). In all these storage systems the main underlying challenge is to map spatial or spatio-temporal data (2D or 3D) to 1-dimensional (1D) values, which are used as keys for storage in key-value based NoSQL storage systems. The mapping is typically achieved using variants of space-filling curves, such as Z-order, Hilbert, or Moore encoding. An overview of the different systems is provided in Table 1.

Essentially, this mapping is necessary in order to bridge the gap between mobility data and (1D) key-based NoSQL stores. Based on this mapping to keys, data is distributed, replicated and stored based on partitioning techniques that operate at the level of 1-dimensional key. The challenge is then to translate spatial and spatio-temporal queries to multiple 1D range scans and discover efficient and scalable processing algorithms.

6.1.1 Individual Systems & Techniques

MD-HBase [42] encodes multidimensional data in 1D values using Z-order encoding. This 1D representation is then used by an index layer as a key for storing data in HBase (the storage layer). In this way, standard multidimensional index structures, such as k-d trees and Quadtrees, can be implemented on top of a distributed key-value store. By using the properties of a technique called longest common prefix naming scheme, this mapping of multidimensional indexes to 1D ranges is achieved, offering, in turn, the fundamental mechanism for answering point, range, and nearest-neighbor queries. R-HBase [32] follows a similar approach.

Pyro [36] employs the Moore encoding algorithm, inspired from the Moore space-filling curve, in order to transform (map) spatio-temporal data to 1D values. Then, range queries are translated to multiple 1D range scans, which are processed efficiently by means of different optimizations introduced at the storage layer of HDFS, resulting in PyroDFS, and at an extension of HBase, named PyroDB. In addition, a multi-scan optimizer is used to find the best reading strategy from HBase by considering multiple range scans. Also, a new block grouping algorithm is introduced at the level of the PyroDFS, which preserves data locality and improves the efficiency of dynamic load rebalancing. Pyro is shown to outperform MD-HBase by one order of magnitude for rectangular range queries.

ST-HBase [39] focuses on spatio-textual data, namely data that combines spatial location with textual description. Typical examples of spatio-textual data include geo-tagged objects, for instance tweets, images, etc. ST-HBase resembles the approach followed by MD-HBase, since it also exploits Z-order to transform spatial data to 1D values. However, it goes one step further to support combined spatial and textual retrieval, by introducing the functionality of an inverted index and representing keywords along with 1D values as key in HBase. In this way, textual filtering is supported together with spatial filtering.

GeoHashes. The concept of GeoHashes has been exploited to map spatio-temporal data to 1D values that are stored in Accumulo [25]. Practically, it relies on a hierarchical spatial data structure that partitions the data space in cells, which are then used to build string keys encoded using Base32. These keys resemble the cell identifiers produced by a space-filling curve, such as Z-order. A similar approach is taken by ST-Hash [28], where the generated 1D values are stored in MongoDB.

datAcron Encoding. In [54], an 1D encoding scheme for spatio-temporal data is proposed in the context of the H2020 datAcron project[1], which is applicable for online settings, where the temporal partitioning needs to be performed as data arrive in the system [46]. One challenge addressed by this work is dynamic temporal partitioning, since the temporal extent of the data is not known in

[1] `http://datacron-project.eu/`

System	1D Mapping	Data type	Data space	Data skew	Adaptive	Queries	NoSQL store
MD-HBase [42]	Z-order	multidimensional	static	–	–	range, k-NN	HBase
Pyro [36]	Moore	spatio-temporal	static	–	–	range	PyroDB
GeoHashes [25]	Z-order	spatio-temporal	static	–	–	range	N/A
ST-Hash [28]	Z-order	spatio-temporal	static	–	–	range	MongoDB
ST-HBase [39]	Z-order	spatio-textual	static	–	–	range	HBase
datAcron [54]	Z-order, Hilbert	spatio-temporal	dynamic	✓	–	range	N/A
QUILTS [43]	Generic	multidimensional	static	✓	✓	range	HBase

Table 1: Comparative overview of storage techniques.

advance, with the objective to keep compact 1D values. A space-filling curve (Z-order or Hilbert) is used for the spatial domain, while the temporal part is encoded in the same identifier. This encoding scheme has been applied for encoding spatio-temporal RDF data, and specifically in the storage layer of the DiStRDF [41] engine, which has been developed in Apache Spark.

QUILTS [43] investigates space-filling curves that fit a given data skew and query characteristics. A new method is proposed for partitioning multidimensional data based on a family of space-filling curves that take into account data skewness and query workload characteristics. QUILTS is implemented on top of HBase.

6.1.2 Classification

Even though the systems above share many similarities, they also have subtle differences that are important for providing a comprehensive classification of storage solutions. We identify four significant dimensions for the classification: data dimensionality, statically/dynamically defined data space, data skew, adaptivity to query workload. The result of this classification is summarized in Table 1.

First, regarding the *data dimensionality*, practically all approaches that rely on space-filling curves can be applied to multidimensional data. This also includes spatio-temporal data, which can be seen as 3D data. One exception is ST-HBase [39] which targets 2D spatial data and text.

The second observation is whether the underlying data is constrained to a *statically defined space* or whether the data space changes dynamically. Practically all systems make the assumption that data is defined within a multidimensional box of known size. Although data insertions and updates can be supported, the size of the data space must remain unchanged, otherwise the space partitions need to be redefined. The only notable exception is the mapping proposed in [54], which targets applications that collect streaming spatio-temporal data and need to accommodate this data in an online manner. In this setup, the problem is that the temporal dimension cannot be statically defined, and its range increases as new data arrive in the system.

Also, only limited works have explicitly focused on handling *data skew*. A noteworthy approach is QUILTS [43], which aims to identify the most appropriate 1D mapping for a given data set, under the presence of data skew. The approach in [54] also tackles this problem, focusing mainly on temporal skew in mobility data.

Finally, *adaptive* mappings have not been explored yet. The problem can be stated as finding the best mapping for a given query workload, and selecting when the system should adapt its storage at runtime (e.g., build a different 1D mapping). QUILTS [43] is the only system that identifies changes in the query workload that lead to decisions for adapting the storage scheme.

6.2 Processing Frameworks

Lately, several research projects have extended popular parallel data processing platforms, such as Hadoop or Spark, in order to provide customized solutions for big spatial or spatio-temporal data. The most prominent prototypes and systems in this field include Hadoop-GIS [2], SpatialHadoop [21], AQWA [7], ST-Hadoop [4, 6], SpatialSpark [60], GeoSpark [61] (recently joined Apache Incubator as Apache Sedona[2]), LocationSpark [52], Simba [58, 59], and STARK [30]. We also refer to [29] for a comparative evaluation of big spatial data processing systems.

6.2.1 Spatial and Spatio-temporal Frameworks

Hadoop-GIS [2] is a large-scale spatial data warehousing system for executing spatial queries in parallel. It is available both as a library and as an integrated package in Hive, thus facilitating ease of use. To support indexing, global indexes are built and replicated on all nodes using Hadoop's Distributed Cache. Thus, each node can efficiently determine the regions of the space that contain relevant re-

[2] `http://sedona.apache.org`

	Framework	Partitioning	Indexing	Queries
Spatial	Hadoop-GIS [2]	N/A	Global/local indexing (global region indexes, on demand local indexing)	Range queries (box), spatial joins
	SpatialHadoop [21]	Space partitioning (Grid, Quadtree), data partitioning (STR, k-d tree), space-filling curves (Z-order, Hilbert)	Global/local indexing (R-trees, Grid files)	Range queries (box), k-NN queries, spatial join
	AQWA [7]	Adaptive (based on k-d tree)	N/A	Range queries, k-NN queries
	SpatialSpark [60]	Fixed Grid partitioning, binary space partitioning, tile partitioning	Pre-built local indexes on HDFS	Range queries, spatial join
	GeoSpark [61]	Grid-based partitioning	Local indexes (R-tree and Quadtree)	Range queries, k-NN query, spatial join
	LocationSpark [52]	Data partitioning e.g. using Quadtree (based on sampling)	Global/local indexing (global: Grid and region Quadtree, local: Grid, R-tree, Quadtree, IR-tree)	Range queries, k-NN query, spatial join, k-NN join, spatio-textual queries
	Simba [58, 59]	STRPartitioner (sampling and STR)	IndexRDD	Range queries, k-NN query, distance join, k-NN join
Spatio-temporal	ST-Hadoop [4, 6]	Multi-level temporal partitioning	Temporal hierarchy of spatial indexes at multiple levels of temporal resolution	Spatio-temporal range queries and joins
	STARK [30]	Spatial-only	R-trees	Spatio-temporal range queries and joins
	STJoins@ESRI [56, 57]	Data (re-)partitioning based on Quadtree decomposition	Equi-sized splitting of complete data set and local Quadtrees	Spatio-temporal join
Trajectory	HadoopTrajectory [9]	MBR-based grouping and partitioning	Global index in the form of 3D Grid or 3DR-tree	Range queries
	Parallel Secondo [37]	N/A	Local indexing using full-featured Secondo DBMS	All those offered by Secondo
	UlTraMan [17]	Supports a repartition operator to support different partitioning strategies (including STR)	In-memory: random access RDD using on-heap arrays or using ChronicleMap, an embedded, key-value store	Range query, k-NN, aggregation, comovement pattern queries
	DITA [49]	Grouping of trajectories based on first and last point, and use of STR for partitioning	Global/local indexing: (global: two R-trees built on MBR of first and last points respectively, local: trie-like index on selected points)	Similarity search, similarity join
	MobilityDB [10, 11, 67]	Hierarchical partitioning, multidimensional partitioning	Local indexing based on PostGIS	Range, k-NN, not distributed joins

Table 2: Overview of spatial and spatio-temporal parallel processing frameworks.

sults for the spatial query at hand. Local indexes are dynamically constructed on demand, using main memory. Regarding query types, Hadoop-GIS supports range queries and spatial joins.

SpatialHadoop [21] is an extension of the basic Hadoop implementation, designed for efficient processing of spatial data, that supports spatial indexing, a feature missing from basic Hadoop. SpatialHadoop utilizes a two-layered spatial index which enables selective access to data by spatial operations. Implemented indexes include R-trees, R^+-trees and Grid files. In more detail, SpatialHadoop uses a single global index and several local indexes. The global index maintains information about the data partitions across cluster nodes. The local indexes organize data stored on single nodes. Different partitioning techniques have been studied and evaluated [21] in the context of SpatialHadoop, including Grid, Quadtree, STR, STR+ and k-d trees, as well as partitioning based on Z-order and Hilbert curve. Also, a spatial MapReduce language called Pigeon [20] is also provided as part of SpatialHadoop, thus easing the development of scalable applications that process vast-sized spatial data.

AQWA [7] is a research prototype system that focuses on adaptive partitioning for big spatial data, with a strong emphasis on query-workload-aware partitioning. AQWA is demonstrated on top of

Hadoop, but its techniques are in principle applicable to other systems as well. In contrast to SpatialHadoop that uses static partitioning, AQWA incrementally updates the partitions based on data changes and the distribution of queries.

SpatialSpark [60] is a prototype implementation that focuses mainly on efficient processing of spatial joins in parallel, although range queries are also supported. For data partitioning to machines, data partition strategies such as fixed Grid or k-d tree are employed. SpatialSpark has implemented several spatial indexing and spatial filtering techniques, and it reuses (at the local level) the popular JTS[3] API for spatial refinement, i.e., testing whether two geometric objects satisfy a certain spatial relationship (e.g., point-in-polygon) or calculating a certain metric between two geometric objects (e.g., Eucledian distance).

GeoSpark [61] (Apache Sedona) is a framework for processing large spatial data. Essentially, it offers a spatial layer built on top of Apache Spark, aiming at providing efficient support for spatial data processing. GeoSpark uses JTS to create and process geometries in order to support different query types: range queries, k-NN, and spatial join. It provides a new abstraction named Spatial Resilient Distributed Datasets (SRDDs). Spatial RDDs, such as PointRDD and RectangleRDD, are used in order to effectively partition spatial data to different machines. Partitioning is achieved using a uniform Grid partitioning mechanism, and spatial objects that intersect more than one Grid cells are duplicated to all cells. Each RDD partition can be indexed locally using Quadtree and R-tree indexes. However, global indexing is not supported.

LocationSpark [52] is a spatial data processing system developed on top of Spark that supports different spatial operators (e.g., range, k-NN, spatial join, k-NN join). It follows the global/local indexing approach, where a global index is used (based on sampling) to partition data to cluster nodes, while local indexes are built for each partition. Different options are implemented in terms of global and local indexes. Global indexing of data partitions is achieved by sampling the data and creating equisized partitions. Each partition is locally indexed on each machine using a local index of choice, including a Grid index, an R-tree, a Quadtree, or an IR-tree. In this way, data skew can be effectively addressed. Also, the authors address query skew, by means of a query scheduler that identifies data partitions that are queried by many queries and chooses to reallocate partitions when this cost is affordable. Interestingly, processing of range queries is performed by exploiting a Spatial Bloom Filter that efficiently determines whether a point is contained in a spatial range, thus avoiding the overhead of typical cases for parallel range query processing: (a) replicating points to neighboring partitions, or (b) directing a range query to all overlapping partitions. Experiments report one order of magnitude improvement in performance compared to GeoSpark.

Simba [58, 59] is a system for in-memory spatial analytics implemented in Spark. It extends the Spark SQL engine to support spatial query processing and develops an optimizer that can exploit indexes in order to improve the performance of query processing. At a technical level, Simba introduces the concept of IndexRDDs, thus allowing efficient random access in large datasets in memory, thereby avoiding the limitation of linear (in-memory) scan of Spark when accessing RDDs. Simba supports a new partitioning type, named STRPartitioner, which performs random sampling on the input and then runs one iteration of the STR algorithm [35] in order to determine the partition boundaries. The computed partition boundaries need to be extended in order to cover the space of the complete data set.

In terms of query operators, Simba supports range queries, k-NN, distance join, and k-NN joins, and introduces new physical execution plans to Spark SQL, in order to efficiently process such spatial queries. This is a notable difference to other systems, such as GeoSpark and SpatialSpark, which are libraries implemented on top of Spark, whereas Simba introduces changes to the kernel of Spark SQL. In this way, cost-based optimization of spatial queries is also provided in Simba. Moreover, Simba supports multiple dimensions, in contrast to most other systems that are constrained to 2 dimensions. Simba is evaluated against SpatialHadoop and Hadoop GIS and is considerably faster, due to the in-memory processing. Also, Simba is shown to be more efficient than in-memory parallel processing systems, such as GeoSpark and SpatialSpark, because of its indexing and query optimizer which are built inside the query engine of Spark.

ST-Hadoop [4, 6] is an open-source MapReduce extension of Hadoop tailored for spatio-temporal data processing. Support for spatio-temporal indexing is a core feature of ST-Hadoop. It is achieved by means of a multi-level temporal hierarchy of spatial indexes. Each level corresponds to a specific time resolution (e.g., day, month, etc.). Also, at each level the entire data set is replicated and spatio-temporally partitioned based on the temporal res-

[3] `https://sourceforge.net/projects/jts-topo-suite/)`

olution of that particular level. ST-Hadoop supports spatio-temporal range queries, aggregations and spatio-temporal joins. Another recent work called Summit [5] is based on ST-Hadoop, and focuses on trajectory data.

STARK [29, 30] is one of the few existing solutions targeting big spatio-temporal data. STARK addresses query processing of spatio-temporal data in Spark, whereas other approaches only consider the spatial dimensions. STARK supports spatio-temporal partitioning and indexing using R-trees. Thus, it supports spatio-temporal filtering and join operations. However, the temporal dimension is not treated equally to the spatial dimensions. For example, partitioning in [30] is performed solely based on spatial criteria, and the temporal part of a query is used to filter out data objects that do not satisfy the temporal constraint. In essence, the temporal dimension is treated as yet another dimension that can be queried, and it cannot be used for eager pruning of data in the case of a very selective temporal constraint.

STJoins@ESRI [56, 57] presents an algorithm for spatio-temporal join over large spatio-temporal data sets. It is not a complete system of a framework that supports different functionalities, rather the focus is on a specific operation. In the case that one of the inputs is relatively small and fits in memory of cluster nodes, broadcast join is employed, where the small data set is sent to all nodes, whereas the other one is partitioned to the nodes. In the case that both inputs are large, a repartition join algorithm is employed, which is called bin join.

6.2.2 *Trajectory Management Frameworks*

HadoopTrajectory [9] is an extension of Hadoop that integrates spatio-temporal data types, indexed access and trajectory operators. At the indexing level, a global index is constructed (either as 3D Grid or in the form of 3DR-tree), and it can be used at query time to identify relevant partitions at worker nodes to the query at hand. Partitioning of trajectories can be performed either at trajectory level or per moving object. At the processing level, various trajectory operators are implemented as MapReduce jobs, most notably trajectory range queries.

Parallel Secondo [37] is a hybrid system that is built using Hadoop in order to efficiently process mobility data. It combines Hadoop with a set of single node instances of Secondo database, which has been built for mobility data management and processing. This hybrid coupling is inspired by an earlier attempt, namely HadoopDB [1], to couple Hadoop with relational DBMSs. Parallel Secondo offers the data types and execution language as a front-end, thus enabling users to express their queries to the parallel engine transparently, while using the features of the execution language.

UlTraMan [17] proposes a unified platform for the complete management cycle of big trajectory data. It provides both storage and processing layer for trajectory data. In the storage layer, ChronicleMap is used, an embedded key-value store, which is integrated in the block manager of Apache Spark. In the processing layer, UlTraMan employs an enhanced MapReduce paradigm that provides flexible APIs to applications. Interestingly, this is one of the few approaches that target the entire lifecycle of big trajectory data, from data loading and indexing, to processing and analytics. Supported query operators include range queries, k-NN queries, and aggregation queries. In addition, co-movement pattern mining on trajectory data is also supported, demonstrating the trajectory analytics capabilities of UlTraMan. Dragoon [24] is an extension that supports both offline and online analytics.

DITA [49] is another recent research prototype that targets in-memory trajectory analytics, also extending Apache Spark. It offers an extended Spark SQL language that facilitates the declarative specification of queries, but also index construction. Furthermore, DITA extends the Catalyst optimizer of Spark SQL in order to optimize trajectory similarity queries, using cost-based optimization. At the indexing level, DITA uses global/local indexes and proposes an approximate representation technique for trajectories based on pivot points. For data partitioning the STR algorithm is used, operating on selected points of trajectories, namely the first and last points of each trajectory. The trajectories are grouped based on their first points, and then subgroups are created by grouping based on the last points. Then, the global indexing mechanism consists of two R-trees, one constructed on the MBRs of first points and another one constructed on the MBRs of last points. The local indexing is a variant of trie-based indexing which is built on top of the pivot points of trajectories. At the algorithmic/processing level, DITA adopts the filter-and-refine paradigm, in order to efficiently process similarity search and similarity joins.

MobilityDB [10, 11, 67] provides a distributed system that scales PostgreSQL horizontally over multiple workers. It uses two partitioning schemes: hierarchical (first based on time, then space) and multidimensional (where the 3D space is partitioned to cells) to distribute the data to workers in a load-

balanced way. Distributed MobilityDB supports many spatio-temporal query types, but not the generic case of distributed spatio-temporal joins (e.g., self-join on positions of moving objects) that require re-distribution of data.

7. CONCLUSIONS & OUTLOOK

In this paper, we provided a succinct overview of big data processing frameworks and techniques for spatial, spatio-temporal, and trajectory data, which are key building blocks for applications that involve mobility analytics. We presented prototype systems and frameworks both at the storage layer and at the processing layer. Moreover, we couple the presentation of individual systems with an explanation of the most prominent underlying techniques for partitioning, indexing and query processing, as a guide for further research in this field.

Regarding open problems and research directions, a challenging problem is extending big data frameworks towards handling spatio-temporal-textual retrieval, which is very common in modern applications. Incorporating text makes the setup high-dimensional and this raises challenges for effective indexing and partitioning. Existing efforts [8, 31] have so far focused on mapping/encoding the textual information in numeric values to be handled uniformly with the spatio-temporal information, however there exist limitations with respect to the used mappings and they still focus on centralized settings.

Another challenge in a big data setting relates to management of skewed spatio-temporal data, which may result in partitions of uneven size. Addressing the problem of data skew, in order to achieve load balancing and minimize the execution time of distributed query processing, is still a promising research field. In addition to this, learned indexes [44, 45] are researched lately, as an alternative way to index spatial data by learning the data distribution

Last, but not least, real-time processing and analysis of spatio-temporal data calls for stream processing frameworks and online techniques, often in conjunction with building concise data summaries and approximate processing, aiming at low latency execution. Although this direction is outside the scope of this survey, we acknowledge research initiatives in this direction [27, 46].

Acknowledgements

This work was supported by EU projects Track&Know (Grant Agreement No 780754), VesselAI (Grant Agreement No 957237), and from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Innovation (GSRI), under Grant Agreement No 1667.

8. REFERENCES

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. J. Abadi, A. Rasin, and A. Silberschatz. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *Proc. VLDB Endow.*, 2(1):922–933, 2009.

[2] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz. Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. *PVLDB*, 6(11):1009–1020, 2013.

[3] M. M. Alam, S. Ray, and V. C. Bhavsar. A performance study of big spatial data systems. In *Proc. of SIGSPATIAL*, pages 1–9, 2018.

[4] L. Alarabi and M. F. Mokbel. A demonstration of ST-Hadoop: A MapReduce framework for big spatio-temporal data. *PVLDB*, 10(12):1961–1964, 2017.

[5] L. Alarabi and M. F. Mokbel. A demonstration of Summit: A scalable data management framework for massive trajectory. In *Proc. of MDM*, pages 226–227, 2020.

[6] L. Alarabi, M. F. Mokbel, and M. Musleh. ST-Hadoop: A MapReduce framework for spatio-temporal data. In *Proc. of SSTD*, pages 84–104, 2017.

[7] A. M. Aly, A. R. Mahmood, M. S. Hassan, W. G. Aref, M. Ouzzani, H. Elmeleegy, and T. Qadah. AQWA: Adaptive query-workload-aware partitioning of big spatial data. *PVLDB*, 8(13):2062–2073, 2015.

[8] Y. Arseneau, S. Gautam, B. G. Nickerson, and S. Ray. STILT: Unifying spatial, temporal and textual search using a generalized multi-dimensional index. In *Proc. of SSDBM*, pages 11:1–11:12. ACM, 2020.

[9] M. S. Bakli, M. A. Sakr, and T. H. A. Soliman. HadoopTrajectory: A Hadoop spatiotemporal data processing extension. *J. Geogr. Syst.*, 21(2):211–235, 2019.

[10] M. S. Bakli, M. A. Sakr, and E. Zimányi. Distributed mobility data management in MobilityDB. In *Proc. of MDM*, pages 238–239, 2020.

[11] M. S. Bakli, M. A. Sakr, and E. Zimányi. Distributed spatiotemporal trajectory query processing in SQL. In *Proc. of SIGSPATIAL*, pages 87–98, 2020.

[12] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache Flink™: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[13] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2010.

[14] H. Chasparis and A. Eldawy. Experimental evaluation of selectivity estimation on big spatial data. In *Proc. of GeoRich*, pages 8:1–8:6, 2017.

[15] A. Das, J. Gehrke, and M. Riedewald. Approximation techniques for spatial data. In *Proc. of SIGMOD*, pages 695–706, 2004.

[16] A. Davoudian, L. Chen, and M. Liu. A survey on NoSQL stores. *ACM Comput. Surv.*, 51(2), 2018.

[17] X. Ding, L. Chen, Y. Gao, C. S. Jensen, and H. Bao. UlTraMan: A unified platform for big trajectory data management and analytics. *PVLDB*, 11(7):787–799, 2018.

[18] C. Doulkeridis and K. Nørvåg. A survey of large-scale analytical query processing in MapReduce. *VLDB J.*, 23(3):355–380, 2014.

[19] A. Eldawy, L. Alarabi, and M. F. Mokbel. Spatial partitioning techniques in SpatialHadoop. *PVLDB*, 8(12):1602–1605, 2015.

[20] A. Eldawy and M. F. Mokbel. Pigeon: A spatial MapReduce language. In *Proc. of ICDE*, pages 1242–1245, 2014.

[21] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce framework for spatial data. In *Proc. of ICDE*, pages 1352–1363, 2015.

[22] A. Eldawy and M. F. Mokbel. The era of big spatial data: A survey. *Foundations and Trends in Databases*, 6(3-4):163–273, 2016.

[23] Y. Fang, R. Cheng, W. Tang, S. Maniu, and X. S. Yang. Scalable algorithms for nearest-neighbor joins on big

trajectory data. *IEEE Trans. Knowl. Data Eng.*, 28(3):785–800, 2016.
[24] Z. Fang, L. Chen, Y. Gao, L. Pan, and C. S. Jensen. Dragoon: A hybrid and efficient big trajectory management system for offline and online analytics. *The VLDB Journal*, 30:287–310, 2021.
[25] A. D. Fox, C. N. Eichelberger, J. N. Hughes, and S. Lyon. Spatio-temporal indexing in non-relational distributed databases. In *Proc. of IEEE Big Data*, pages 291–299, 2013.
[26] F. García-García, A. Corral, L. Iribarne, M. Vassilakopoulos, and Y. Manolopoulos. Efficient distance join query processing in distributed spatial data management systems. *Inf. Sci.*, 512:985–1008, 2020.
[27] N. Giatrakos, E. Alevizos, A. Artikis, A. Deligiannakis, and M. N. Garofalakis. Complex event recognition in the big data era: A survey. *VLDB J.*, 29(1):313–352, 2020.
[28] X. Guan, C. Bo, Z. Li, and Y. Yu. ST-hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database. In *Proc. of Geoinformatics*, pages 1–7, 2017.
[29] S. Hagedorn, P. Götze, and K. Sattler. Big spatial data processing frameworks: Feature and performance evaluation. In *Proc. of EDBT*, pages 490–493, 2017.
[30] S. Hagedorn, P. Götze, and K. Sattler. The STARK framework for spatio-temporal data analytics on Spark. In *Proc. of BTW*, pages 123–142, 2017.
[31] T. Hoang-Vu, H. T. Vo, and J. Freire. A unified index for spatio-temporal keyword queries. In *Proc. of CIKM*, pages 135–144, 2016.
[32] S. Huang, B. Wang, J. Zhu, G. Wang, and G. Yu. R-HBase: A multi-dimensional indexing framework for cloud computing environment. In *Proc. of ICDMW*, pages 569–574, 2014.
[33] E. H. Jacox and H. Samet. Spatial join techniques. *ACM Trans. Database Syst.*, 32(1):7, 2007.
[34] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, 2014.
[35] S. T. Leutenegger, J. M. Edgington, and M. A. López. STR: A simple and efficient algorithm for R-tree packing. In *Proc. of ICDE*, pages 497–506, 1997.
[36] S. Li, S. Hu, R. K. Ganti, M. Srivatsa, and T. F. Abdelzaher. Pyro: A spatial-temporal big-data storage system. In *Proc. of USENIX*, pages 97–109, 2015.
[37] J. Lu and R. H. Güting. Parallel SECONDO: Practical and efficient mobility data processing in the cloud. In *Proc. of IEEE Big Data*, pages 17–25, 2013.
[38] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using MapReduce. *Proc. VLDB Endow.*, 5(10):1016–1027, 2012.
[39] Y. Ma, Y. Zhang, and X. Meng. ST-HBase: A scalable data management system for massive geo-tagged objects. In *Proc. of WAIM*, pages 155–166, 2013.
[40] S. Maguerra, A. Boulmakoul, L. Karim, and B. Hassan. A survey on solutions for big spatio-temporal data processing and analytics. In *Proc. of INTIS*, pages 127–140, 2018.
[41] P. Nikitopoulos, A. Vlachou, C. Doulkeridis, and G. A. Vouros. DiStRDF: Distributed spatio-temporal RDF queries on Spark. In *Proc. of BMDA*, pages 125–132, 2018.
[42] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. MD-HBase: A scalable multi-dimensional data infrastructure for location aware services. In *Proc. of MDM*, pages 7–16, 2011.
[43] S. Nishimura and H. Yokota. QUILTS: Multidimensional data partitioning framework based on query-aware and skew-tolerant space-filling curves. In *Proc. of SIGMOD*, pages 1525–1537, 2017.
[44] V. Pandey, A. van Renen, A. Kipf, J. Ding, I. Sabek, and A. Kemper. The case for learned spatial indexes. In *Proc. of AIDB*, 2020.
[45] J. Qi, G. Liu, C. S. Jensen, and L. Kulik. Effectively learning spatial indices. *Proc. VLDB Endow.*, 13(11):2341–2354, 2020.
[46] G. M. Santipantakis, A. Glenis, K. Patroumpas, A. Vlachou, C. Doulkeridis, G. A. Vouros, N. Pelekis, and Y. Theodoridis. SPARTAN: Semantic integration of big spatio-temporal data from streaming and archival sources. *Future Gener. Comput. Syst.*, 110:540–555, 2020.
[47] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *Proc. VLDB Endow.*, 10(11):1178–1189, 2017.
[48] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Parallel trajectory similarity joins in spatial networks. *VLDB J.*, 27(3):395–420, 2018.
[49] Z. Shang, G. Li, and Z. Bao. DITA: Distributed in-memory trajectory analytics. In *Proc. of SIGMOD*, pages 725–740, 2018.
[50] P. Tampakis, C. Doulkeridis, N. Pelekis, and Y. Theodoridis. Distributed subtrajectory join on massive datasets. *ACM Trans. Spatial Algorithms Syst.*, 6(2):8:1–8:29, 2020.
[51] M. Tang, Y. Yu, W. G. Aref, A. R. Mahmood, Q. M. Malluhi, and M. Ouzzani. LocationSpark: In-memory distributed spatial query processing and optimization. *CoRR*, abs/1907.03736, 2019.
[52] M. Tang, Y. Yu, Q. M. Malluhi, M. Ouzzani, and W. G. Aref. LocationSpark: A distributed in-memory data management system for big spatial data. *PVLDB*, 9(13):1565–1568, 2016.
[53] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. V. Ryaboy. Storm@twitter. In *Proc. of SIGMOD*, pages 147–156, 2014.
[54] A. Vlachou, C. Doulkeridis, A. Glenis, G. M. Santipantakis, and G. A. Vouros. Efficient spatio-temporal RDF query processing in large dynamic knowledge bases. In *Proc. of SAC*, pages 439–447, 2019.
[55] T. Vu and A. Eldawy. R*-Grove: Balanced spatial partitioning for large-scale datasets. *Frontiers Big Data*, 3:28, 2020.
[56] R. T. Whitman, B. G. Marsh, M. B. Park, and E. G. Hoel. Distributed spatial and spatio-temporal join on Apache Spark. *ACM Trans. Spatial Algorithms Syst.*, 5(1):6:1–6:28, 2019.
[57] R. T. Whitman, M. B. Park, B. G. Marsh, and E. G. Hoel. Spatio-temporal join on Apache Spark. In *Proc. of SIGSPATIAL*, pages 20:1–20:10, 2017.
[58] D. Xie, F. Li, B. Yao, G. Li, Z. Chen, L. Zhou, and M. Guo. Simba: Spatial in-memory big data analysis. In *Proc. of SIGSPATIAL*, pages 86:1–86:4, 2016.
[59] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient in-memory spatial analytics. In *Proc. of SIGMOD*, pages 1071–1085, 2016.
[60] S. You, J. Zhang, and L. Gruenwald. Large-scale spatial join query processing in cloud. In *Proc. of ICDEW*, pages 34–41, 2015.
[61] J. Yu, J. Wu, and M. Sarwat. A demonstration of GeoSpark: A cluster computing framework for processing big spatial data. In *Proc. of ICDE*, pages 1410–1413, 2016.
[62] J. Yu, Z. Zhang, and M. Sarwat. Spatial data management in Apache Spark: The GeoSpark perspective and beyond. *GeoInformatica*, 23(1):37–78, 2019.
[63] H. Yuan and G. Li. Distributed in-memory trajectory similarity search and join on road network. In *Proc. of ICDE*, pages 1262–1273, 2019.
[64] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proc. of NSDI*, pages 15–28, 2012.
[65] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache Spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016.
[66] C. Zhang, F. Li, and J. Jestes. Efficient parallel kNN joins for large data in MapReduce. In E. A. Rundensteiner, V. Markl, I. Manolescu, S. Amer-Yahia, F. Naumann, and I. Ari, editors, *Proc. of EDBT*, pages 38–49, 2012.
[67] E. Zimányi, M. A. Sakr, and A. Lesuisse. MobilityDB: A mobility database based on PostgreSQL and PostGIS. *ACM Trans. Database Syst.*, 45(4):19:1–19:42, 2020.