Track & Know

Big Data for Mobility Tracking Knowledge Extraction in Urban Areas

SoA – Big Data Processing and Architectures

As the world's population living in metropolitan areas increases, so increases the need for effective and sustainable interventions and services to inject mobility intelligence and improve the quality of life in large urban environments. Technological developments, in particular the extended and expanding use of ICT, have resulted the collection of unprecedented volumes of data across systems operating in the transport, mobility and the urban applications domains. Moreover, the influence of digital evolution is changing the experience of consumers of services in these domains and is driving the expectations that will shape the demand in the coming years. Although certain markets have been radically changed by the influence of technology, the transport, mobility and urban services sectors are changing at a slower pace and both commercial providers and public operators are very slowly adapting to the current technology offerings.

The existing accumulated large volumes of data, known also as "big data", are generating a strong interest in the research communities, the relevant industries and among policy makers. The adoption of digital services is expected to enable service providers to deliver secure and efficient services across intelligent infrastructures with higher automation capacity. As a result, the demand for efficient and scalable smart services facilitating personalized, adaptable, environmental and sustainable capabilities impose new requirements for the improved exploitation of the immense and continuously rising amounts of data generated by industrial operations, sensors and devices (Internet of Things - IoT), social media and often aggregated Open Data sources.

Increasingly, in smart cities cloud-based infrastructures combined with behavioural, institutional and policy-related data sources create a critical pathway in achieving the four-partite goals of Health, Liveability, Adaptability and Sustainability (HLAS). Elaborating on the cross-domain Big Data Value generation, so far, the most common approaches e.g. for smart mobility focus on tracking vehicles' spatial and temporal information and have not been related with health, adaptable insurance services and life quality indices (i.e. driving and cognitive capabilities, driving skills deterioration, prevention, symptoms early-detection, prediction and control, etc.).

Unfortunately, in today's information society, the ability of retrieving knowledge from mobility and contextual data is becoming more and more critical for the competitiveness of all the economic, political and cultural entities. Companies produce within their individual activities and along with synergies (i.e. traffic monitoring, fleet and healthcare management, emergency response and/or adaptable insurance services, etc.) a huge flow of data coming from diverse domains, different devices,

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 780754.

processes, markets, user generated content/feedback or external sources. The rise of ubiquitous connectivity combined with IoT (Internet of Things) is the key enabler for the rise of “Industry 4.0” and already allows acquiring data from an evolving mobile and stationary ecosystem. The more data are available, the more the tools to manage them have advanced. Databases, software, computing power and in general technical infrastructure have grown and improved in the past years, thanks to investments and research. However, there is a gap between the availability of data and the potential information or knowledge that can be extracted from them: not every enterprise has now the means to analyse such an amount of data. In addition to this, Data Science imposes the tight collaboration among different stakeholders coming from diverse domains (i.e. ICT, Transport, Insurance, Health, etc.) to add value to this kind of data in an intelligent, efficient and scalable manner. But still, this amount of data most of the time remains decoupled and isolated within private infrastructures. Many companies can benefit from data management and analysis in order to infer knowledge by effectively blending data. Knowledge from Big Data is the key for success.

In this direction, Track&Know, by injecting computational thinking capabilities in the context of smart mobility services, aims to address the challenges of the emerging Big Data Value ecosystem, including the autonomous, connected and shared vehicles technologies, the self-enablement configuration features (such as e.g. in Intelligent Transportation Systems - ITS, self-parking, speed and/or lane control), addressing the questions of what type of information is needed, and with whom, when and how it is used. More specifically, Track&Know integrates multidisciplinary research teams from Mobility Data management, Complex Event Recognition, Geospatial Modelling, Complex Network Analysis, Transportation Engineering and Visual Analytics to develop new models and applications. An insight of Track&Know concept is represented in Figure 1 (source: www.bdva.eu), providing an overview on how the aforementioned challenges are going to be addressed conjunctively and towards providing a coherent Big Data Framework and related solutions.

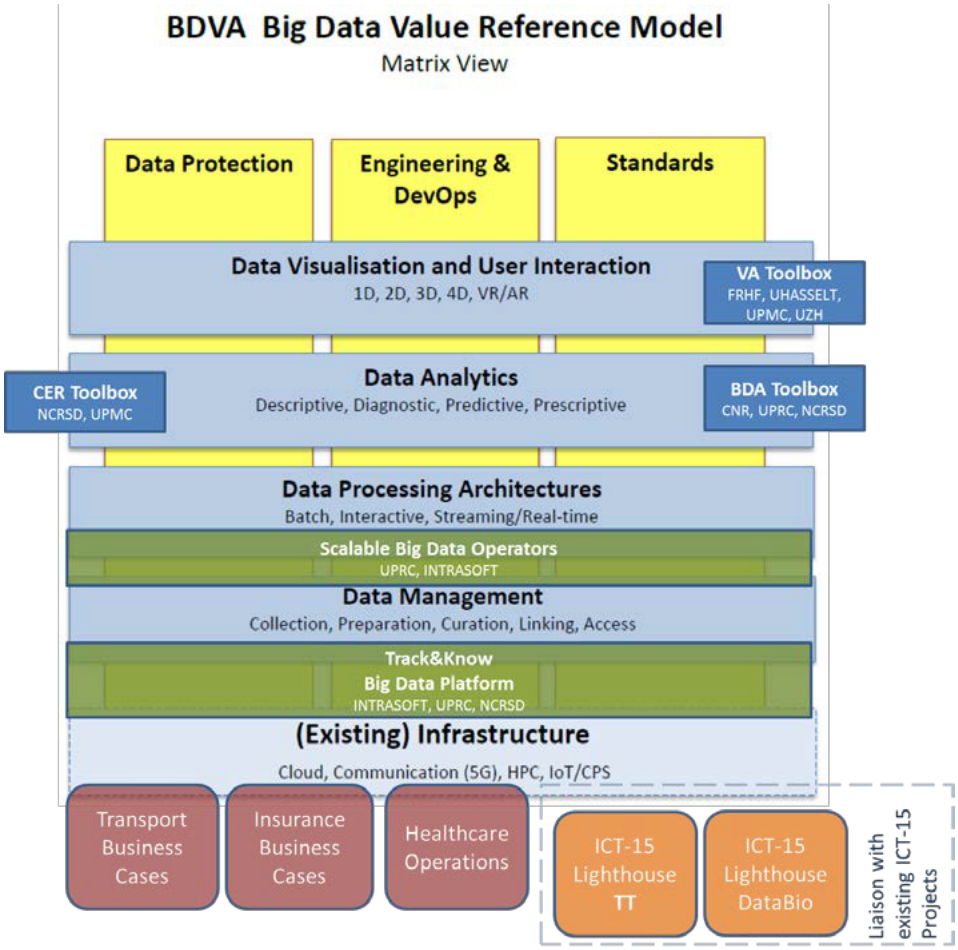

Figure 1: The Track & Know components mapped to the BDVA reference model.

1 Big Data Platforms and Infrastructure

The scope of this section is to provide an insight to available Big Data Platforms and Big Data File / Storage Systems, as well as existing Big Data Batch / Stream Processing approaches and connectors in order to present building blocks of potential systems facilitating storage, exchange and analysis of data.

1.1 Big Data Platforms and Infrastructure

Cloudera Enterprise - Cloudera Enterprise (Cloudera) in its current version 6 includes CDH, an open source Hadoop-based platform. CDH is Cloudera's 100% open source platform distribution, including Apache Hadoop and built to meet enterprise demands. By integrating Hadoop with more than a dozen other critical open source projects such as Flume Hbase, Hive, Apache Impala, Apache Spark, Sqoop, Cloudera has created a functionally advanced system that helps in performing end to end Big Data workflows. Cloudera Enterprise is available in three editions, each offering varying levels of services management capabilities. The basic edition provides management capabilities to support cluster running core CDH services that include HDFS, Hive, MapReduce, Oozie, YARN and ZooKeeper. It should be noted that although the Cloudera distribution and installer software enables a distributed and highly available deployment of the components mentioned, security features are made available in licenced only versions.

Cloudera's Benefits

- One integrated system, bringing diverse users and application workloads to one pool of data on common infrastructure; no data movement required
- Perimeter security, authentication, granular authorization, and data protection (licenced versions)
- Enterprise-grade data auditing(control), data lineage, and data discovery
- Native high-availability, fault-tolerance and self-healing storage, automated backup and disaster recovery, and advanced system and data management
- Apache-licensed open source to ensure data and applications remain yours, and an open platform to connect with all existing investments in technology and skills
- One scalable platform to store any amount or type of data, in its original form, for as long as desired or required
- Integrated with your existing infrastructure and tools
- Flexible to run a variety of enterprise workloads - including batch processing, interactive SQL, enterprise search and advanced analytics

Cloudera's Disadvantages

- Not fully Open Source, some components of the distributions are privately owned, meaning with public contributions are not welcome
- More Up to date technologies
- Many attractive features that may be a comparative advantage incur licencing costs.
- Improvements to Cluster Management tool is required, which are already available in competitors

Hortonworks - During the course of the project (Oct 2018), Hortonworks has merged with Cloudera with the tools available in Hortonworks Data Platform becoming the Cloudera Data Platform. The usage of the platform in production settings now requires licencing. Initially architected, developed, and built completely in the open, Hortonworks Data Platform (HortonworksDP) provided a Hadoop distribution designed to meet the needs of enterprise data processing. The platform enabled multi-workload data processing across an array of processing methods - from batch through interactive to real-time - all supported with solutions for governance, integration, security, agile application

deployment, machine learning, deep learning workloads and operations. Furthermore, Hortonworks served as a contributor to the open-source Hadoop community focused on evolving it into a broadly capable data-management platform. Everything in the Hortonworks Data Platform (HDP) was freely available as open-source software. Hortonworks distribution featured a master-slave architecture and it supported among others MapReduce, YARN, Spark, Kafka, Flink and HBase. Furthermore, Hortonworks included no proprietary software, used Ambari and Cloudbreak for management and Stinger for handling queries, and Apache Solr for searches of data. Cloudera and Hortonworks were combined in a merge of equals with Cloudera shareholder owning more than 50 percent of the combined company. The product is no longer available open source and is only available for development, trial and paid subscription clients.

Hortonworks's Benefits

- Completely Open (currently not valid): HDP was the only completely open Hadoop data platform available. All solutions in HDP were developed as projects through the Apache Software Foundation (ASF). There were no proprietary extensions or add-ons required.
- Fundamentally Versatile: At its heart HDP offered linear scale storage and computation across a wide range of access methods from batch to interactive, to real time, search and streaming. It included a comprehensive set of capabilities across governance, integration, security, agile application deployment, machine learning, deep learning workloads and operations. These features are now in the Cloudera Data Platform
- Wholly Integrated: HDP integrated with and augmented existing applications and systems so that taking advantage of Hadoop with only minimal change to existing data architectures and skillsets was possible. It was possible to deploy HDP in-cloud, on-premise or from an appliance across both Linux and Windows.
- Hortonworks allowed the users the flexibility to run the same industry-leading, open source platform to gain data insights in the data center as well as on the public cloud of choice.

Hortonworks's Disadvantages

- Installation was complex
- Minor stability issues existed with the platform with users choosing more stable rather than latest releases.
- Upgrading from lower versions required effort.
- There was room for improvement in monitoring. The Ambari Management interface on HDP was basic.

MapR – In the course of the project (Aug 2019) following financial difficulties the technologies and intellectual property of MapR was sold to Hewlett Packard. The MapR Distribution (MapR) including Apache Hadoop provided an enterprise-grade distributed data platform capable of storing reliably and processing big and fast data. MapR Distribution represented a good foundation for running batch, interactive, and real-time applications. With an open choice approach to open source, MapR offered a broad range of technologies (multiple projects for SQL-on-Hadoop, NoSQL databases, execution engines such as Spark, etc) to choose from. Furthermore, MapR M7 Hadoop distribution addressed weaknesses in HBase by doing away with region servers, table splits and merges, and data-compaction steps. MapR has also implemented its own architecture for snapshotting, high availability, and system recovery. With M7, MapR also introduced optional LucidWorks Search software on top of Hadoop for building out recommendation engines, fraud-detection, and predictive applications.

The three platform services offered (MapR-FS, MapR-DB, and MapR Streams), were unified by common core capabilities built into the underlying platform such as high availability, real-time access, unified security, multi-tenancy, disaster recovery, a global namespace, self-healing, and management

and monitoring. The MapR Converged Data Platform allowed to quickly and easily build breakthrough, reliable, real-time applications by providing:

- Single cluster for streams, file storage, database, and analytics.
- Persistence of streaming data, providing direct data access to batch and interactive frameworks, eliminating data movement.
- Unified security framework for data-in-motion and data-at-rest, with authentication, authorization, and encryption.
- Utility-grade reliability with self-healing and no single point-of-failure architecture.

The above functionality is no longer offered and is only available via Hewlett Packard Enterprise Ezmeral Data Fabric. Data Fabric builds on innovations of MapR providing a unified data platform for ingestion, storage, management, processing and analysis of various data types originating from various data sources with a variety of ingestion mechanisms.

MapR's Benefits

- Unified Big Data Platform: Capable of creating a complete picture of all data, including high-velocity, real-time data, to find previously unidentifiable insights. Process more data types with the schema-less flexibility and the high-velocity read/write capabilities of the integrated in-Hadoop online database platform.
- Proven Production Readiness: Ability to get continuous value from your data with the technology proven in production to meet strict service level agreements. Deploy 24x7 online applications with enterprise-grade capabilities to achieve zero downtime. Run HBase-compatible applications with zero database administration.
- Consistent High Performance at Any Scale: Ability to get faster results on larger data sets to respond more quickly to more complete data. Achieve quicker application responsiveness for an enhanced user experience. Easily load and process high volumes and high velocities of incoming data.

MapR's Disadvantages

- Many fearures proprietary and not open-source
- Associated licencing costs

Apache Ambari - Ambari (Apache Ambari) is a completely open source management platform for provisioning, managing, monitoring and securing Apache Hadoop clusters. Apache Ambari can help in taking the guesswork out of operating Hadoop. Apache Ambari allows enterprises to plan, install and securely configure Hadoop making it easier to provide ongoing cluster maintenance and management, irrespective to the size of the cluster. This is achieved by step-by-step wizards for installing services across multiple hosts and by handling the configurations of these services. Ambari makes Hadoop management simpler by providing a consistent, secure platform for operational control with an intuitive Web UI as well as a robust REST API, which is particularly useful for automating cluster operations, including central management for starting, stopping and reconfiguring Hadoop services across the entire cluster. The tools allow Hadoop operators get the following core benefits:

- Simplified Installation, Configuration and Management
- Centralized Security Setup.
- Full Visibility into Cluster Health and status via dashboards
- Ambari Metrics System for metrics collection
- Ambari Alert Framework for when attention is needed such as node failure, low disk space etc.
- Highly Extensible and Customizable

Confluent – Confluent Platform currently at version 6.0 (Confluent) provides an event streaming platform for real-time and historical events, enabling the user to build event-driven applications and gain a universal event pipeline. It is a stream data platform to leverage Apache Kafka. There exist cloud-native (Confluent Cloud) and on-premise (Confluent Platform) options which include a wide set of open source, community licenced and commercially licenced features. It should be noted that even the commercial features remain free of charge in single broker deployments (for development) while the open source and community licensed components have no usage restrictions. Confluent is founded by the original creators of Apache Kafka and the main benefits of it are that it:

- Manages data streaming: Confluent platform uses the Apache Kafka as the base of all its functionalities. It creates a centralized system for data transport. It also provides all the relevant tools for connecting data sources, data sinks, and apps to the platform.
- Provides open-source platform: It provides distribution of Apache Kafka.
- Provides Community licenced features such as Proxies, KSQL DB etc.
- Streamlines admin operations: Confluent Enterprise provides Kafka with the essential monitoring, administration and management tools for the system.
- Empowers cloud technology: Confluent Cloud is a fully hosted Apache Kafka SaaS and it gives essential solutions for steaming data.

Lenses.io – The Lenses Platform (Lenses) provides a streaming platform which works with all Kafka distributions including Apache Kafka, Confluent, Cloudera, Azure and other managed services. While in the past many open source components were made available, during the course of the project, Lenses.io has evolved into a paid service for individual developers, teams and organisations. The main benefits of Lenses are:

- Monitor & Administer: Full visibility and alerting on the Kafka enterprise health and secure self-service administration.
- Discover & Investigate: Accelerate the time to develop and investigate Kafka by exploring data in the streams, viewing state and producer/consumer information.
- Secure & Comply: Reduce risk and meet compliance by discovering, securing and auditing Kafka environment and data.
- Build & Deploy: Build and deploy new flows in minutes with SQL-based stream processing via Kubernetes and data integration connectors configured through a UI.

Streaml.io – Based on a cloud-native, distributed messaging and streaming platform named Apache Pulsar (Apache Pulsar) Streaml.io (Streaml.io), includes the usual commercial support and professional services, together with a closed-source management console. In Nov 2019 Splunk has acquired Streaml.io but there is rich functionality available in the core open source product, including the efficient and durable multi-tenancy support. In a similar way to Apache Kafka, Pulsar has its ecosystem for data processing while providing Spark and Storm adaptors. Some benefits of the technologies include:

- Multi-tenant or geo-replicated system from the start.
- Capable of handling large data storage needs.
- Ease of querying and processing data no matter how long ago in the past.
- Provides core features of a distributed log platform.
- Transparent usage of Tiered Storage.

1.2 Big Data File / Storage Systems

In general, Data stores are grouped according to their data model, i.e. SQL vs. NoSQL (Cattell et al. 2011):

- Key-value Stores: These systems store values and an index to find them, based on a programmer defined key.
- Document Stores: These systems store documents, as just defined. The documents are indexed, and a simple query mechanism is provided.
- Extensible Record Stores: These systems store extensible records that can be partitioned vertically and horizontally across nodes. Some papers call these “wide column stores”.
- Relational Databases: These systems store (and index and query) tuples.

Key Value Stores: The simplest data stores use a data model similar to the popular memcached distributed in-memory cache, with a single key-value index for all the data. These systems are called key-value stores. Unlike memcached, these systems generally provide a persistence mechanism and additional functionality as well: replication, versioning, locking, transactions, sorting, and/or other features. The client interface provides inserts, deletes, and index lookups. Like memcached, none of these systems offer secondary indices or keys. Some noticeable Key Value Stores include:

- Project Voldermort
- Riak
- Redis
- Tokyo Cabinet
- Aerospike CE
- LevelDB
- Scalaris
- HyperDex
- Berkeley DB

Document stores support more complex data than the key-value stores. Although termed “document store” and these systems could store “documents” in the traditional sense (articles, Microsoft Word files, etc.), a document in these systems can be any kind of “pointerless object”. Unlike the key-value stores, these systems generally support secondary indexes and multiple types of documents (objects) per database, and nested documents or lists. Like other NoSQL systems, the document stores do not provide ACID transactional properties. Here are some of the Document Stores:

- SimpleDB
- CouchDB
- Couchbase
- MongoDB
- Terrastore
- JasDB
- RaptorDB
- SisoDB
- EJDB
- djonDB
- TokuMX
- SchemafreeDB

Extensible Record Stores: The extensible record stores seem to have been motivated by Google’s success with BigTable. Their basic data model is rows and columns, and their basic scalability model is splitting both rows and columns over multiple nodes:

- Rows are split across nodes through sharding on the primary key. They typically split by range rather than a hash function. This means that queries on ranges of values do not have to go to every node.
- Columns of a table are distributed over multiple nodes by using “column groups”. These may seem like a new complexity, but column groups are simply a way for the customer to indicate which columns are best stored together.

Extensible Record Stores include:

- Hadoop Distributed File System (HDFS)
- HBase
- Cassandra
- Amazon DynamoDB
- Apache Accumulo
- Azure Tables
- Bigtable

Hadoop Distributed File System - The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

Hardware failure is the norm rather than the exception. An HDFS instance may consist of hundreds or thousands of server machines, each storing part of the file system's data. The fact that there are a huge number of components and that each component has a non-trivial probability of failure means that some component of HDFS is always non-functional. Therefore, detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.

HDFS has been designed to be easily portable from one platform to another. This facilitates widespread adoption of HDFS as a platform of choice for a large set of applications. An HDFS has a master/slave architecture and an HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are several DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

HDFS is part of the Apache Hadoop Core project. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers, each of which may be prone to failures.

HBase - Apache HBase (Jiang 2012) is the Hadoop database, is a type of NoSQL database, a distributed, scalable, big data store. When there is a need for random, real-time read/write access to your Big Data, Apache HBase is befitting. This project's goal is the hosting of very large tables (billions of rows X millions of columns) using clusters of commodity hardware. Apache HBase is an open-source, distributed, versioned, non-relational database modelled. Some key features include the following:

- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding of tables
- Automatic failover support between RegionServers.
- Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.
- Easy to use Java API for client access.
- Block cache and Bloom Filters for real-time queries.
- Query predicate push down via server-side Filters

- Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options
- Extensible jruby-based (JIRB) shell
- Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX

Cassandra - Apache Cassandra (Rabl et al. 2012) is a free and open-source distributed NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra in its current version 3.11 offers robust support for clusters spanning multiple datacentres, with asynchronous masterless replication allowing low latency operations for all clients.

Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime. There are no network bottlenecks. Every node in the cluster is identical. Every node in the cluster has the same role. There is no single point of failure. Data is distributed across the cluster (so each node contains different data), but there is no master as every node can service any request. Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.

Cassandra is not row level consistent, meaning that inserts and updates into the table that affect the same row that are processed at approximately the same time may affect the non-key columns in inconsistent ways. One update may affect one column while another affects the other, resulting in sets of values within the row that were never specified or intended.

Relational Databases

Unlike the other data stores, relational DBMSs have a complete pre-defined schema, a SQL interface, and ACID transactions. Traditionally, RDBMSs have not achieved the scalability of the some of the previously described data stores. It appears likely that some relational DBMSs will provide scalability comparable with NoSQL data stores, with two provisions:

- Use small-scope operations: As we've noted, operations that span many nodes, e.g. joins over many tables, will not scale well with sharding.
- Use small-scope transactions: Likewise, transactions that span many nodes are going to be very inefficient, with the communication and two-phase commit overhead.

It should be noted that NoSQL systems avoid these two problems by making it difficult or impossible to perform larger scope operations and transactions.

Relational Databases are:

- MySQL Cluster
- VoltDB
- Clustric
- ScaleDB
- ScaleBase
- NimbusDB
- PostgreSQL
- SQLite
- MariaDB
- Apache Hive

SQL vs NoSQL: Key Differences (SQL vs. NoSQL)

- One of the key differentiators is that NoSQL supported by column-oriented databases where RDBMS is row-oriented database.
- NoSQL seems to work better on both unstructured and unrelated data. The better solutions are the crossover databases that have elements of both NoSQL and SQL.

- RDBMSs that use SQL are schema–oriented which means the structure of the data should be known in advance to ensure that the data adheres to the schema. For example, predefined schema-based applications that use SQL include Payroll Management System, Order Processing and Flight Reservations.
- SQL Databases are vertically scalable – this means that they can only be scaled by enhancing the horsepower of the implementation hardware, thereby making it a costly deal for processing large batches of data.
- NoSQL databases give up some features of the traditional databases for speed and horizontal scalability. NoSQL databases on the other hand are perceived to be cheaper, faster and safer to extend a preexisting program to do a new job than to implement something from scratch.
- More importantly, data Integrity is a key feature of SQL based databases. This means, ensuring the data is validated across all the tables and there's no duplicate, unrelated or unauthorized data inserted in the system.
- Advantages of SQL databases are that they are typically more performant when dealing with more complex queries. Users cite the relational nature of SQL DBs encourages a well-structured database.

1.3 Big data Batch processing

Apache Flink - Apache Flink (Apache Flink) is an open source stream processing framework developed by the Apache Software Foundation. The core of Apache Flink is a distributed streaming dataflow engine written in Java and Scala. Flink executes arbitrary dataflow programs in a data-parallel and pipelined manner. With its pipelined runtime system, it enables the execution of bulk/batch and stream processing programs. Furthermore, Flink's runtime supports the execution of iterative algorithms natively.

Flink provides a high-throughput, low-latency streaming engine as well as support for event-time processing and state management. Flink applications are fault-tolerant in the event of machine failure and support exactly-once semantics. Programs can be written in Java, Scala, Python, and SQL and are automatically compiled and optimized into dataflow programs that are executed in a cluster or cloud environment. It is worth mentioning that Flink does not provide its own data storage system and provides data source and sink connectors to systems such as Amazon Kinesis, Apache Kafka, HDFS, Apache Cassandra, and ElasticSearch.

In general, Flink:

- provides results that are accurate, even in the case of out-of-order or late-arriving data;
- is stateful and fault-tolerant and can seamlessly recover from failures while maintaining exactly-once application state;
- performs at large scale, running on thousands of nodes with very good throughput and latency characteristics.

Apache Hadoop - Apache Hadoop (Hadoop) is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. In its current version 2.10.0, it provides a software framework for distributed storage and processing of big data using the MapReduce programming model. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS) and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

Apache Spark - Apache Spark (Spark), in its current preview version 3.0.0-preview, has as its architectural foundation the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way. RDD is a fundamental data structure of Spark. This processing tool is a programming abstraction that represents an immutable collection of objects that can be split across a computing cluster. Operations on the RDDs can also be split across the cluster and executed in a parallel batch process, leading to fast and scalable parallel processing. RDDs can be created from simple text files, SQL databases, NoSQL stores (such as Cassandra and MongoDB) among others. Much of the Spark Core API is built on this RDD concept, enabling traditional map and reduce functionality, but also providing built-in support for joining data sets, filtering, sampling, and aggregation.

Hadoop MapReduce - Hadoop MapReduce (Dean and Ghemawat, 2004) is a software framework for easily writing applications which process vast amounts of data (multi-terabyte datasets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. A MapReduce job usually splits the input dataset into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically, the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

At their minimal, applications specify the input / output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client. Although the Hadoop framework is implemented in Java, MapReduce applications need not be written in Java.

1.4 Big data Stream processing

Spark Streaming - Spark Streaming (Spark Streaming) makes it easy to build scalable fault-tolerant streaming applications. Spark Streaming brings Apache Spark's language-integrated API to stream processing, letting you write streaming jobs the same way you write batch jobs. It supports Java, Scala and Python. Stateful exactly-once semantics out of the box. Combine streaming with batch and interactive queries.

Spark Streaming extended the Apache Spark concept of batch processing into streaming by breaking the stream down into a continuous series of microbatches, which could then be manipulated using the Apache Spark API. In this way, code in batch and streaming operations can share (mostly) the same code, running on the same framework, thus reducing both developer and operator overhead. A criticism of the Spark Streaming approach is that microbatching, in scenarios where a low-latency response to incoming data is required, may not be able to match the performance of other streaming-capable frameworks like Apache Storm, Apache Flink, and Apache Apex, all of which use a pure streaming method rather than microbatches.

Instead of processing the streaming data one record at a time, Spark Streaming discretizes the streaming data into tiny, sub-second micro-batches. In other words, Spark Streaming's Receivers accept data in parallel and buffer it in the memory of Spark's workers nodes. Then the latency-optimized Spark engine runs short tasks (tens of milliseconds) to process the batches and output the results to other systems. Note that unlike the traditional continuous operator model, where the computation is statically allocated to a node, Spark tasks are assigned dynamically to the workers based on the locality of the data and available resources. This enables both better load balancing and faster fault recovery.

Kafka Streaming - Kafka Streams (Kafka Stream) is a client library for building applications and microservices, where the input and output data are stored in a Kafka cluster. In its current version 2.3.0, it combines the simplicity of writing and deploying standard Java and Scala applications on the client side with the benefits of Kafka's server-side cluster technology.

Some benefits of Kafka include:

- Elastic, highly scalable, fault-tolerant
- Deploy to containers, VMs, bare metal, cloud
- Equally viable for small, medium, & large use cases
- Fully integrated with Kafka security
- Write standard Java applications
- Exactly-once processing semantics
- No separate processing cluster required

Apache Storm - Apache Storm (Apache Storm, 2016), current version 2.1.0, is a free and open source distributed real-time computation system. Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing. Storm is simple, can be used with any programming language. Storm has many use cases: real-time analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Storm integrates with the queueing and database technologies you already use. A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.

Pulsar Functions - Apache Pulsar (Apache Pulsar) offers Pulsar Functions which represent lightweight compute processes. These processes can consume messages from one or multiple topics and apply a processing logic to each message according to the user needs. Finally, the results of the computation are published to another Pulsar Topic. By utilising Pulsar Functions which represent the computing infrastructure of Pulsar, complex processing logic can be achieved without introducing other technologies or systems. This can potentially increase developer productivity and ease troubleshooting while also helping in operational simplicity.

1.5 Connectors

Kafka Connect - Kafka Connect (Kafka Connect), current version 2.3.0, is a framework for scalably and reliably streaming data between Apache Kafka and other data systems. Connect makes it simple to use existing connector implementations for common data sources and sinks to move data into and out of Kafka. Kafka Connect's applications are wide ranging. A source connector can ingest entire databases and stream table updates to Kafka topics or even collect metrics from all of your application servers into Kafka topics, making the data available for stream processing with low latency. A sink connector can deliver data from Kafka topics into secondary indexes like Elasticsearch or into batch systems such as Hadoop for offline analysis.

Kafka Connect is focused on streaming data to and from Kafka. This focus makes it much simpler for developers to write high quality, reliable, and high-performance connector plugins and makes it possible for the framework to make guarantees that are difficult to achieve in other frameworks.

The main benefits of using Kafka Connect are:

- Data Centric Pipeline – use meaningful data abstractions to pull or push data to Kafka.
- Flexibility and Scalability – run with streaming and batch-oriented systems on a single node or scaled to an organization-wide service.
- Reusability and Extensibility – leverage existing connectors or extend them to tailor to your needs and lower time to production.

Spark Connectors - Both Spark and HBase are widely used, but how to use them together with high performance and simplicity is a very challenging topic. Spark HBase Connector (SHC) provides feature rich and efficient access to HBase through Spark SQL. It bridges the gap between the simple HBase key value store and complex relational SQL queries and enables users to perform complex data analytics on top of HBase using Spark. SHC implements the standard Spark data source APIs, and leverages the Spark catalyst engine for query optimization. To achieve high performance, SHC constructs the RDD from scratch instead of using the standard HadoopRDD. With the customized RDD, all critical techniques can be applied and fully implemented, such as partition pruning, column pruning, predicate pushdown and data locality. The design makes the maintenance easy, while achieving a good tradeoff between performance and simplicity.

Furthermore, Spark is collaborating with MongoDB, Cassandra, Sorl, Elasticseaerch etc. in creating connectors between them (Spark packages). There is also the ability to define custom and purpose-built connectors depending on the needs (Connector Devel.). The community offers a big selection of implementations that work with specific technologies and tools. There also exists a Connect Developer Guide so as to create connectors if necessary although in general it is preferable to use already existing connectors and it is a solution when something specific is required.

Pulsar IO - Pulsar IO (Apache Pulsar) is the equivalent of Kafka Connect and enables the creation, deployment and management of connectors that connect to other data systems as either sources or sinks. Sources feed data from external messaging systems or data pipelines into Pulsar. Sinks in turn, feed data to other messaging systems and databases (both SQL and NoSQL). Pulsar comes with many available sink and source connectors which also provide processing guarantees ranging from at-most-once, to at-least-once and effectively-once while custom implementations are offered the necessary tools for achieving delivery semantics.

1.6 Industry-related Benchmarks

In this section, some applications of big data technologies and solutions are presented by looking into specific industries which have adopted some big data approaches in order to achieve their goals. In the following paragraphs the Nissan motor company, Octo and Novartis are highlighted:

Nissan Motor Company Ltd (Nissan): Connected cars that leverage driving data are a vision that automobile manufacturers are aggressively pursuing. Nissan Motor Company Ltd (Nissan) is on this journey also. The company was experiencing a variety of business challenges, namely the need for infrastructure capable of storing huge volumes of vehicle driving data and product quality data on a long-term basis. Also, it had a need for a Hadoop platform capable of deploying a variety of data cross-functionally. Nissan turned to Hadoop as the solution for its Big Data problem, relying on Hortonworks Data Platform (HDP). The open source model appealed to the company due to the large numbers of engineering talent in the market and the flexibility to pivot if circumstances changed down the road.

OCTO (Octo): Octo Telematics is provider of telematics and data analytics solutions for the auto insurance industry. By collecting and analyzing data from connected cars, Octo Telematics gives

insurers insights to more effectively assess driver risk, deliver crash and claim services, and manage customer relationships. “We utilize every type of data—contextual data, driving data, behavioral data, and crash data—to forecast driving habits, improve crash notifications and response, evaluate crash dynamics, and detect fraud,” said Gianfranco Giannella, COO, Octo Telematics. As Octo Telematics grew, executives sought to replace a custom-made data platform with a more scalable, next generation data management platform. “We wanted to rapidly expand the footprint of our services,” said Giannella. “We needed a platform that would support a growing volume of telematics and IoT data and enable us to prototype services and products much faster.” Octo Telematics today powers its telematics Internet of Things (IoT) solution with Cloudera Enterprise. The platform stores, processes, and analyzes data on more than 170 billion miles of driving and approximately 400,000 severe crashes from five million connected cars. In all, Octo adds over 11 billion new data points from connected cars daily to the platform. Internal and external data sources, such as traffic and weather data, are also incorporated to provide additional context. Using machine learning, the company can make more accurate predictions and risk models.

NOVARTIS (Novartis): The MapR-based flexible workflow tool is now being used for a variety of different projects across Novartis, including video analysis, proteomics, and meta-genomics. The combined Spark and MapR-based workflow and integration layers allow the company’s life science researchers to meaningfully take advantage of the tens of thousands of experiments that public organizations have conducted, which gives them a significant competitive advantage**.** One of their areas of drug research, Next Generation Sequencing (NGS) data, requires heavy interaction with diverse data from external organizations such as 1000 Genomes, NIH’s GTEx (Genotype-Tissue Expression) and The Cancer Genome Atlas—paying particular attention to clinical, phenotypical, experimental and other associated data. Integrating these heterogeneous datasets is labor intensive, so they only want to do it once. To solve the first part of this NGS big data problem, the Novartis team built a workflow system that allows them to process NGS data while being responsive to advances in the scientific literature. Although NGS data requires high data volumes that are ideal for Hadoop, a common problem is that researchers rely on many tools that simply don’t work on native HDFS. Since these researchers previously couldn’t use systems like Hadoop, they have had to maintain complicated ‘bookkeeping’ logic to parallelize for optimum efficiency on traditional High-Performance Computing (HPC). This workflow system uses the MapR Distribution for Hadoop for its performance and robustness and to provide the POSIX file access that lets bioinformaticians use their familiar tools. Additionally, it uses the researchers’ own metadata to allow them to write complex workflows that blend the best aspects of Hadoop and traditional HPC.

2 Big Data Management and Processing

Big data management raises numerous research challenges (Jagadish et al. 2014) in different phases of the big data processing and analysis pipeline, including: (a) data acquisition, (b) information extraction 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 (Catell, 2010) (Davoudian et al., 2018). 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 (Zaharia et al. 2016) is one of the most popular solutions nowadays with a large and growing user-base, however other solutions, such as Flink (Carbone et al. 2015), are also applicable with success. In particular, Spark has successfully addressed many of the limitations (Doulkeridis & Nørvåg, 2014) of Hadoop, and operates in main-memory by its core abstraction: RDDs (Resilient Distributed Datasets) (Zaharia et al. 2012). In the real-time processing domain, the most notable systems in use today are Storm (Toshniwal et al. 2014) and Flink (Carbone et al. 2015).

This section 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, which is the core topic of Track&Know. 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, and most of them focus solely on big spatial data (Eldawy & Mokbel, 2016) (Hagedorn et al. 2017), rather than spatio-temporal data. In fact, the majority of developed prototypes extend Hadoop or Spark in order to be applicable for spatial data.

2.1 Big Data Storage and Indexing

Systems that extend scalable storage solutions for multidimensional data have been proposed, most notably MD-HBase (Nishimura et al. 2011), but also solutions tailored specifically for spatio-temporal data, such as Pyro (Li et al. 2015), as well as spatio-textual data, such as ST-HBase (Ma et al. 2013). In all these storage systems the main underlying challenge is to *map* spatial or spatio-temporal data (2D or 3D) to 1-dimensional 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.

Essentially, this mapping is necessary in order to bridge the gap between mobility data and (1-dimensional) 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 1-dimensional range scans and discover efficient and scalable processing algorithms.

As several big data storage systems also include a processing engine, they are reviewed in the following subsection on Big Data Processing. Instead, in this subsection, we mainly focus on design choices for the storage layer.

MD-HBase: MD-HBase (Nishimura et al. 2011) encodes multidimensional data in 1-dimensional values using Z-order encoding. This 1-dimensional 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 Quad trees, 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 1-dimensional ranges is achieved, offering, in turn, the fundamental mechanism for answering point, range, and nearest-neighbor queries.

Pyro: Pyro (Li et al. 2015) employs the Moore encoding algorithm, inspired from the Moore space-filling curve, in order to transform (map) spatio-temporal data to 1-dimensional values. Then, range queries are translated to multiple 1-dimensional 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 (Figure 2). In addition, a multi-scan optimizer is used to find the best reading strategy from HBase while considering multiple range scans. Also, a new block grouping algorithm is introduced at the level of the Distributed File System, 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.

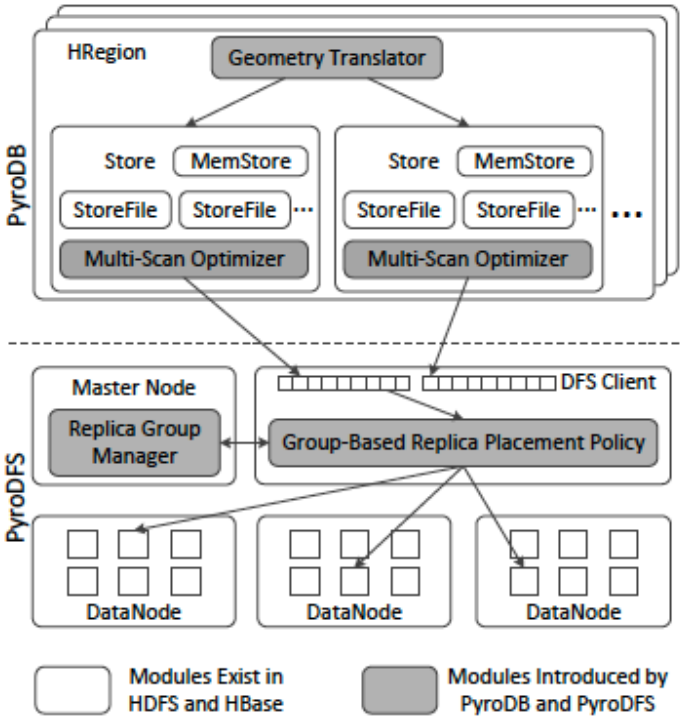

Figure 2: The architecture of Pyro (Li et al. 2015)

ST-HBase: ST-HBase (Ma et al. 2013) 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 1-dimensional 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 1-dimensional values as key in HBase. In this way, textual filtering is supported together with spatial filtering.

GeoHashes: The concept of Geohashes has been expoited to map spatio-temporal data to 1D values that are stored in Accumulo (Fox et al. 2013). Practically, it relies in 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 (Guan et al. 2017), where the generated 1D values are stored in MongoDB.

datAcron Encoding: In (Vlachou et al. 2019), an 1D encoding scheme for spatio-temporal data is proposed in the context of the H2020 datAcron project (http://datacron-project.eu/), which is applicable for online settings, where the temporal partitioning needs to be performed as data arrive in the system (Santipantakis et al. 2020). One challenge addressed by this work is dynamic temporal

partitioning, since the temporal extent of the data is not known in 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 (Nikitopoulos et al. 2018) engine, which has been developed in Apache Spark.

QUILTS (Nishimura et al. 2017) investigates space-filling curves that fit a given data skew and query characteristics. QUILTS proposes a new method to partition 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 and evaluated for different data types, including retail data and spatio-temporal data.

2.2 Big Data Processing

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 (Aji et al. 2013), Parallel SECONDO (Lu & Güting, 2013), SpatialHadoop (Eldawy & Mokbel, 2016), AQWA (Aly et al. 2015), ST-Hadoop (Alarabi et al. 2017) (Alarabi & Mokbel, 2017), SpatialSpark (You et al. 2015), GeoSpark (Yu et al. 2016), LocationSpark (Tang et al. 2016), Simba (Xie et al. 2016a) (Xie et al. 2016b), STARK (Hagedorn et al. 2017a), which are reviewed in the following subsections. We also refer to (Hagedorn et al. 2017b) for a comparative evaluation of big spatial data processing systems. In addition, a brief overview of recent systems built for parallel processing of big trajectory data is presented as a separate subsection.

Table 1: Overview of spatial and spatio-temporal parallel processing frameworks.

	Framework	Partitioning	Indexing	Queries
Spatial	**Hadoop GIS**	N/A	Global/local indexing (global region indexes, on demand local indexing)	Range queries (box), spatial joins
	Parallel Secondo	N/A	Local indexing using full-featured Secondo DBMS	All those offered by Secondo
	SpatialHadoop	Space partitioning (grid, quad tree), data partitioning (STR, STR+, k-d tree), space-filling curves (Z-order, Hilbert)	Global/local indexing (R-trees, grid files)	Range queries (box), kNN queries, spatial join
	AQWA	Adaptive (based on k-d tree)	N/A	Range queries, KNN queries
	SpatialSpark	Fixed grid partitioning, binary space partitioning, tile partitioning	Pre-built local indexes on HDFS	Range queries, spatial join

	GeoSpark	Grid-based partitioning	Local indexes (R-tree and quad tree)	Range queries, KNN query, spatial join,
	LocationSpark	Data partitioning e.g. using quad-tree (based on sampling)	Global/local indexing (global: grid and region quad-tree, local: grid, R-tree, quad-tree, IR-tree)	Range queries, KNN query, spatial join, KNN join, spatio-textual queries
	Simba	STRPartitioner (sampling and STR)	IndexRDD	Range queries, KNN query, distance join, kNN join
Spatio-temporal	**ST-Hadoop**	Multi-level temporal partitioning	Temporal hierarchy of spatial indexes at multiple levels of temporal resolution	Spatio-temporal range queries and joins
	STARK	Spatial-only	R-trees	Spatio-temporal range queries and joins
	STJoins@ESRI	Data (re-)partitioning based on quad-tree decomposition	Equi-sized splitting of complete data set and local quad trees	Spatio-temporal join
Trajectory	**HadoopTrajectory**	Multi-level temporal partitioning	Temporal hierarchy of spatial indexes at multiple levels of temporal resolutionn	Spatio-temporal range queries and joins
	UlTraMan	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, KNN, aggregation, co-movement pattern queries
	DITA	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

2.2.1 Spatial and Spatio-temporal Frameworks

Hadoop GIS (Aji et al. 2013) 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 results 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.

Parallel Secondo (Lu & Güting, 2013) 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, 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 parallel queries in the same way as sequential queries, while using the features of the execution language.

SpatialHadoop (Eldawy & Mokbel, 2015) is an extension of the basic Hadoop implementation developed by the University of Minnesota. It is designed for efficient processing of spatial data, and achieves this by supporting spatial indexing, a feature missing from basic Hadoop. SpatiaHadoop utilizes a two-layered spatial index which enables selective access to data by spatial operations. Implemented indexes include 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 (Eldawy et al. 2015) in the context of SpatialHadoop, including grid and quad tree as space partitioning techniques, STR, STR+ and k-d trees as data partitioning techniques, and Z-order and Hilbert curve as partitioning based on space-filling curves. Also, a spatial MapReduce language called Pigeon (Eldawy & Mokbel, 2014) is also provided as part of SpatialHadoop, thus easing the development of scalable applications that process vast-sized spatial data.

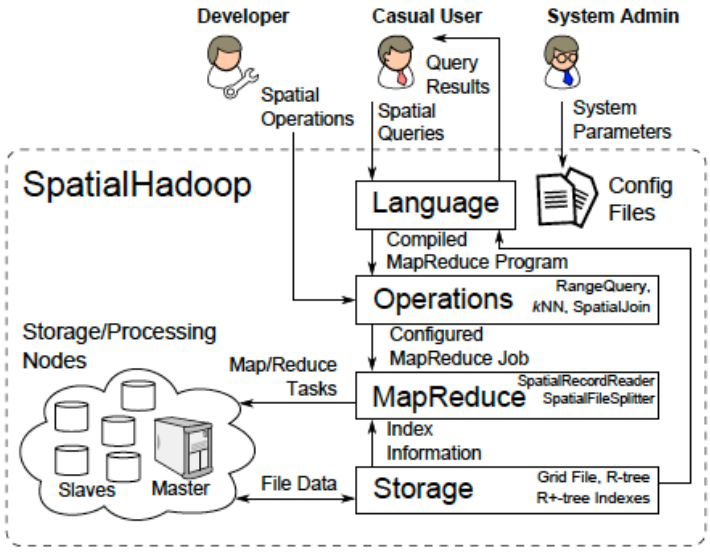

Figure 3: SpatialHadoop system architecture (Eldawy & Mokbel, 2015)

AQWA (Aly et al. 2015) is a research prototype system that focuses on adaptive partitioning for big spatial data, with a strong emphasis on query-workload-aware partitioning. In contrast to SpatialHadoop that uses static partitioning, AQWA incrementally updates the partitions based on data changes and the distribution of queries. Experiments demonstrate up to one order of magnitude performance gain compared to SpatialHadoop.

ST-Hadoop (Alarabi et al. 2017) (Alarabi & Mokbel, 2017) is an open-source MapReduce extension of Hadoop tailored for spatio-temporal data processing, also developed by the University of Minnesota. 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, in each level the entire data set is replicated and spatio-temporally partitioned based on the temporal resolution of that particular level. ST-Hadoop supports spatio-temporal range queries, aggregations and spatio-temporal joins.

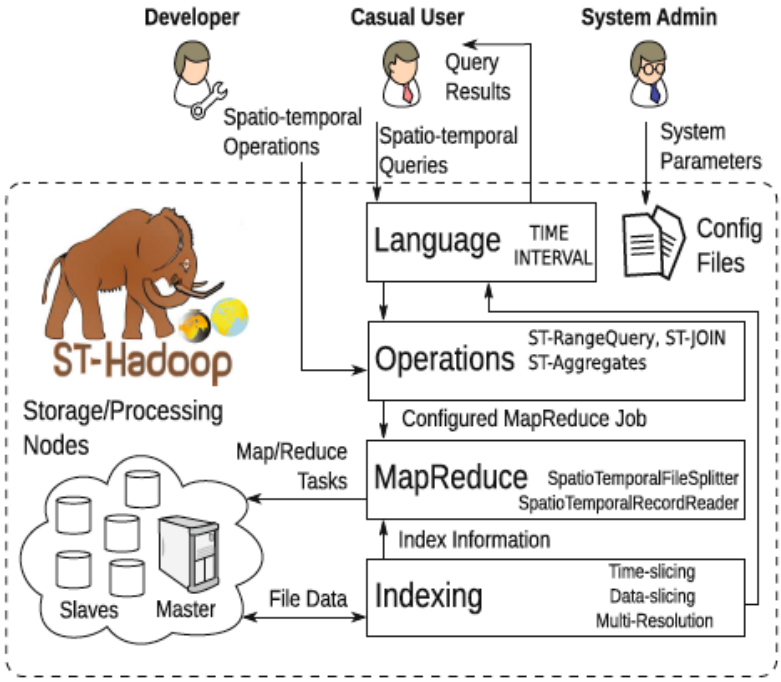

Figure 4: ST-Hadoop system architecture (Alarabi et al. 2017)

SpatialSpark (You et al. 2015) is a prototype implementation that focuses mainly of efficient processing of spatial join in parallel, although range queries are also supported. For partitioning data to machines, data partition strategies such as fixed grid or kd-tree are employed. SpatialSpark has implemented several spatial indexing and spatial filtering techniques, and it reuses (at the local level) the popular JTS[1] 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 (Yu et al. 2016) 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-nearest neighbor (kNN), 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 standard, 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 local using QuadTree and R-tree indexes. However, global indexing is not supported.

[1] URL: https://sourceforge.net/projects/jts-topo-suite/.

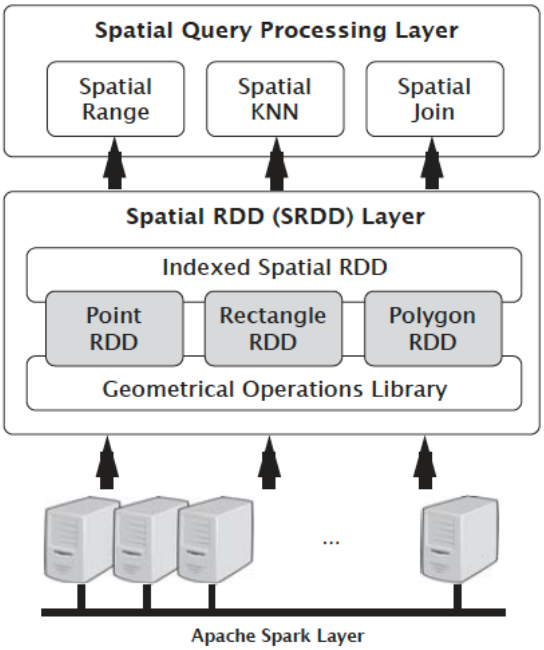

Figure 5: The architecture of GeoSpark (Yu et al. 2015)

LocationSpark (Tang et al. 2016) is a spatial data processing system developed on top of Spark that supports different spatial operators (e.g., Range, KNN, Spatial Join, KNN 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 equi-sized partitions. Each partition is locally indexed on each machine using a local index of choice, including grid index, R-tree, quad-tree, 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 overheads 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.

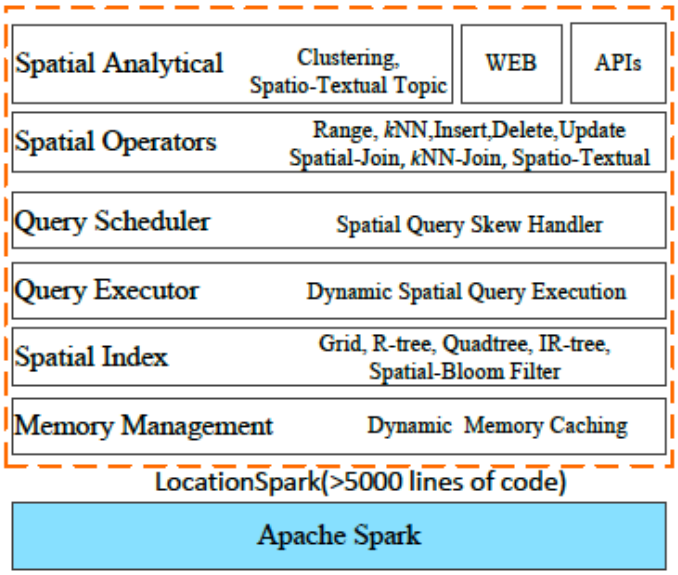

Figure 6: The architecture of LocationSpark (Tang et al. 2016)

Simba (Xie et al. 2016a; Xie et al. 2016b) 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 Sort-Tile-Recursive (STR) algorithm (Leutenegger et al. 1997) 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.

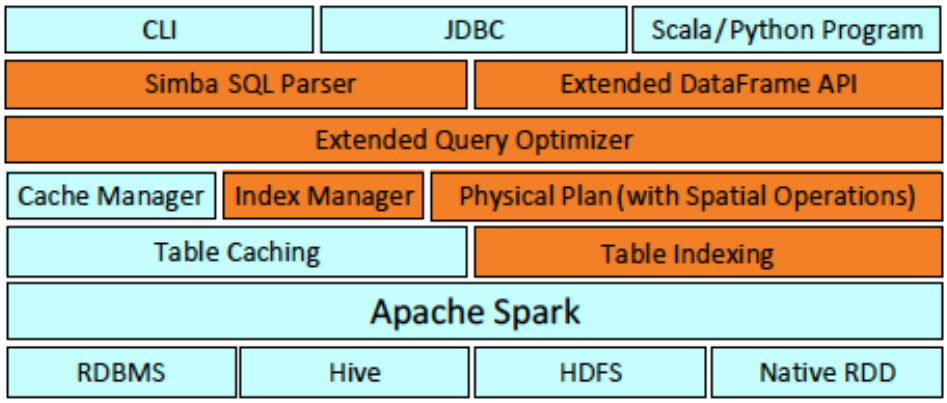

Figure 7: The architecture of Simba (Xie et al. 2016b)

In terms of query operators, Simba supports range queries, kNN, distance join, and kNN 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. Also, 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.

STARK (Hagedorn et al. 2017a) is 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 (Hagedorn et al. 2017a) is performed solely based on spatial criteria, and the temporal part of a query is used to filter out triples 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. In summary, STARK handles separately the temporal from the spatial dimension, thus not fully exploiting spatio-temporal correlations present at the data and performs data filtering separately for time and space.

STJoins@ESRI (Whitman et al. 2017) 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 more generic case where both inputs are large, a repartition join algorithm is employed, which is called bin join.

2.2.2 Trajectory Management Frameworks

Although several systems and prototypes for the management of trajectory data exist, there is limited work on the management and analysis of big trajectory data.

HadoopTrajectory (Bakli et al. 2019) 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 at the level of a moving object. The aim is to have small partitions (so as to avoid accessing trajectories that do not match the query), but also to use large enough partitions (in order to avoid splitting trajectories in many different partitions). At the processing level, various trajectory operators are implemented as MapReduce jobs, with most notable range queries for trajectories.

UlTraMan (Ding et al. 2018) 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, KNN queries, aggregation queries. In addition, co-movement pattern mining on trajectory data is also supported, demonstrating the trajectory analytics capabilities of UlTraMan.

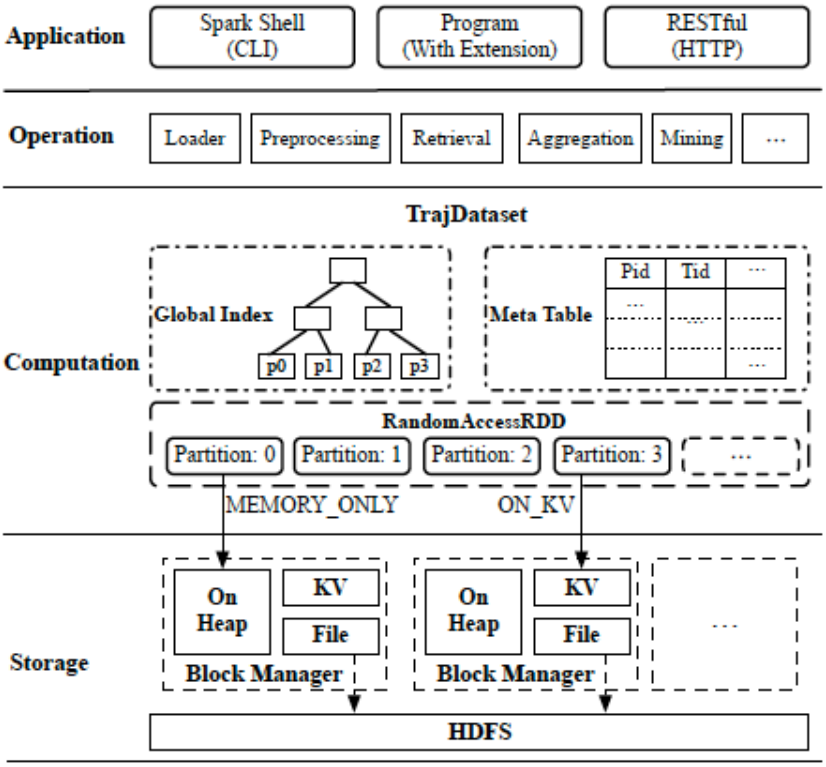

Figure 8: The architecture of UlTraMan (Ding et al. 2018)

DITA (Shang et al. 2018) is another recent research prototype that targets in-memory trajectory analytics, also extending Apache Spark. It offers an extended Spark SQL language that facilitate 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 local/global 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 each 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-verification paradigm, in order to efficiently process similarity search and similarity joins. It is shown that pruning can be achieved by specific conditions that can be checked on the global and local indexes.

Distributed trajectory similarity join is also investigated in (Shang et al. 2017), where 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.

3 References

Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., Saltz, J.H. (2013) Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. PVLDB 6(11): 1009-1020.

Alarabi, L., Mokbel, M.F. (2017) A Demonstration of ST-Hadoop: A MapReduce Framework for Big Spatio-temporal Data. PVLDB 10(12): 1961-1964.

Alarabi, L., Mokbel, M.F., Musleh, M. (2017) ST-Hadoop: A MapReduce Framework for Spatio-Temporal Data. Proceedings of SSTD.

Aly, A.M., Mahmood, A.R., Hassan, M.S., Aref, W.G., Ouzzani, M., Elmeleegy, H., Qadah, T. (2015) AQWA: Adaptive Query-Workload-Aware Partitioning of Big Spatial Data. PVLDB 8(13): 2062-2073.

Apache Ambari. URL: https://hortonworks.com/apache/ambari/

Apache Cassandra. URL: http://cassandra.apache.org/

Apache Flink features. URL: http://flink.apache.org/features.html

Apache Flink. URL: https://flink.apache.org/

Apache Hadoop HDFS. URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Apache Hadoop MapReduce. URL: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

Apache Hadoop. URL: http://hadoop.apache.org

Apache HBase. URL: https://hbase.apache.org/

Apache Pulsar. URL: https://pulsar.apache.org/

Apache Spark packages. URL: https://spark-packages.org/?q=tags%3A%22Data%20Sources%22

Apache Spark Streaming. URL: https://spark.apache.org/streaming/

Apache Spark. URL: https://spark.apache.org/releases/spark-release-2-0-0.html

Apache Storm. URL: http://storm.apache.org/

Bakli, M.S., Sakr, M.A., Soliman, T.H.A. (2019) HadoopTrajectory: A Hadoop spatiotemporal data processing extension. J. Geogr. Syst., 21(2):211-235.

Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., Tzoumas, K. (2015) Apache Flink™: Stream and Batch Processing in a Single Engine. IEEE Data Eng. Bull. 38(4): 28-38.

Cattell, R. (2010) Scalable SQL and NoSQL data stores. SIGMOD Record 39(4): 12-27.

Cattell, R. (2011) Scalable SQL and NoSQL data stores. SIGMOD Rec. 39, 4, 12-27.

Chintaballi, S., Dagit, D., Evans, B., et al. (2016) Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL, pp. 1789-1792. IEEE. DOI: 10.1109/IPDPSW.2016.138.

Cloudera CDH. URL: https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html

Confluent. URL: https://docs.confluent.io/current/platform.html

Connector Devel. URL: https://docs.confluent.io/current/connect/devguide.html

Das, T., M. Zaharia, and P. Wendell (2015) Diving into Apache Spark Streaming's Execution Model. Databricks Engineering Blog. URL: https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html

Davoudian, A., Chen, L., Liu, M. (2018) A Survey on NoSQL Stores. ACM Computing Surveys, 51(2), June 2018.

Dean, J., and Ghemawat, S. (2008) MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM 51, 1, 107-113. DOI: https://doi.org/10.1145/1327452.1327492

Ding, X., Chen, L., Gao, Y., Jensen, C.S., Bao, H. (2018) UlTraMan: A Unified Platform for Big Trajectory Data Management and Analytics. PVLDB (to appear).

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

Eldawy A., Mokbel, M.F. (2015) SpatialHadoop: A MapReduce framework for spatial data. Proceedings of ICDE.

Eldawy, A., Alarabi, L., Mokbel, M.F. (2015) Spatial Partitioning Techniques in Spatial Hadoop. PVLDB 8(12): 1602-1605.

Eldawy, A., Mokbel, M.F. (2014) Pigeon: A spatial MapReduce language. Proceedings of ICDE.

Eldawy, A., Mokbel, M.F. (2016) The Era of Big Spatial Data: A Survey. Foundations and Trends in Databases 6(3-4): 163-273.

Fox, A.D., Eichelberger, C.N., Hughes, J.N., and Lyon, S. (2013) Spatio-temporal indexing in non-relational distributed databases. In Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA, pages 291-299.

Guan, X., Bo, C., Li, Z., and Yu, Y. (2017) ST-Hash: An efficient spatiotemporal index for massive trajectory data in a NoSQL database. In 25th International Conference on Geoinformatics, Geoinformatics 2017, Buffalo, NY, USA, August 2-4, 2017, pages 1-7.

Hagedorn, S., Götze, P., Sattler, K-U. (2017) Big Spatial Data Processing Frameworks: Feature and Performance Evaluation. Proceedings of EDBT.

Hagedorn, S., Götze, P., Sattler, K-U. (2017) Big Spatial Data Processing Frameworks: Feature and Performance Evaluation. Proceedings of EDBT.

Hagedorn, S., Götze, P., Sattler, K-U. (2017): The STARK Framework for Spatio-Temporal Data Analytics on Spark. Proceedings of BTW.

Hortonworks HDP. URL: https://www.cloudera.com/products/hdp.html

Jagadish, H.V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., Shahabi, C. (2014) Big data and its technical challenges. Commun. ACM 57(7), 86-94.

Jiang, Y. (2012). HBase Administration Cookbook. ISBN: 9781849517140. Packt Publishing Ltd.

Kafka Connect. URL: https://docs.confluent.io/current/connect/intro.html

Kafka Stream. URL: https://docs.confluent.io/current/streams/index.html

Key Values Store. URL: https://www.linuxlinks.com/keyvaluestores/

Lenses URL: https://lenses.io/architecture-and-integrations/

Leutenegger, S.T., Edgington, J.M., López, M.A. (1997) STR: A Simple and Efficient Algorithm for R-Tree Packing. Proceedings of ICDE.

Li, S., Hu, S., Ganti, R.K., Srivatsa, M., Abdelzaher, T.F. (2015) Pyro: A Spatial-Temporal Big-Data Storage System. Proceedings of USENIX Annual Technical Conference.

Lu, J., Güting, R.H. (2013) Parallel SECONDO: Practical and efficient mobility data processing in the cloud. Proceedings of BigData.

Ma, Y., Zhang, Y., Meng, X. (2013) ST-HBase: A Scalable Data Management System for Massive Geo-tagged Objects. Proceedings of WAIM.

MapR Distr. URL: https://mapr.com/products/mapr-distribution-including-apache-hadoop/

Nikitopoulos, P., Vlachou, A., Doulkeridis, C., and Vouros, G.A. (2018) DiStRDF: Distributed spatio-temporal RDF queries on Spark. In Proceedings of the Workshops of the EDBT/ICDT 2018 Joint Conference (EDBT/ICDT 2018), Vienna, Austria, March 26, 2018., pages 125-132.

Nishimura, S., Das, S., Agrawal, D., Abbadi, A.E. (2011) MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. Proceedings of MDM.

Nishimura, S., Yokota, H. (2017) QUILTS: Multidimensional data partitioning framework based on query-aware and skew-tolerant space-filling curves. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, pages 1525-1537.

Nissan. URL: https://hortonworks.com/customers/nissan/

Novartis. URL: https://mapr.com/resources/novartis-relies-mapr-flexible-big-data-solutions-drug-discovery/

Octo. URL: https://www.cloudera.com/more/customers/octo-telematics.html

Pointer, I. (2017) What is Apache Spark? The big data analytics platform explained. InfoWorld Analytics blog. URL : https://www.infoworld.com/article/3236869/analytics/what-is-apache-spark-the-big-data-analytics-platform-explained.html

Rabl, T., Sadoghi, M., Jacobsen, H.-A., Gomez-Villamor, S., Muntes-Mulero, V., Mankovskii, S. (2012) In Proceedings of VLDB, Istanbul, Turkey.

Santipantakis, G.M., Glenis, A., Patroumpas, K., Vlachou, A., Doulkeridis, C., Vouros, G.A., Pelekis, N., and Theodoridis, Y. (2020) Spartan: Semantic integration of big spatio-temporal data from streaming and archival sources. Future Generation Computer Systems.

Shang, S., Chen, L., Wei, Z., Jensen, C.S., Zheng, K., Kalnis, P. (2017) Trajectory Similarity Join in Spatial Networks. PVLDB 10(11): 1178-1189.

Shang, Z., Li, G., Bao, Z. (2018) DITA: Distributed In-Memory Trajectory Analytics. Proceedings of SIGMOD (to appear).

Streaml.io. URL: http://streaml.io

Tang, M., Yu, Y., Malluhi, Q.M., Ouzzani, M., Aref, W.G. (2016) LocationSpark: A Distributed In-Memory Data Management System for Big Spatial Data. PVLDB 9(13): 1565-1568.

Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D.V. (2014) Storm@twitter. Proceedings of SIGMOD.

Vlachou, A., Doulkeridis, C., Glenis, A., Santipantakis, G.M., and Vouros, G.A. (2019) Efficient spatio-temporal RDF query processing in large dynamic knowledge bases. In Proceedings of the 34th Annual ACM Symposium on Applied Computing, SAC 2019.

Whitman, R.T., Park, M.B., Marsh, B.G., Hoel, E.G. (2017) Spatio-Temporal Join on Apache Spark. Proceedings of SIGSPATIAL.

Wodehouse, C. (2016) SQL vs. NoSQL Databases: What's the Difference? Upwork Hiring Headquarters blog. URL: https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

Xie, D., Li, F., Yao, B., Li, G., Chen, Z., Zhou, L., Guo, M. (2016) Simba: spatial in-memory big data analysis. Proceedings of SIGSPATIAL.

Xie, D., Li, F., Yao, B., Li, G., Zhou, L., Guo, M. (2016) Simba: Efficient In-Memory Spatial Analytics. Proceedings of SIGMOD.

Yang, W. and M. Tang (2017) Apache Spark—Apache HBase Connector: Feature Rich and Efficient Access to HBase through Spark SQL. URL: https://databricks.com/session/apache-spark-apache-hbase-connector-feature-rich-and-efficient-access-to-hbase-through-spark-sql

You, S., Zhang, J., Gruenwald, L. (2015) Large-scale spatial join query processing in Cloud. Proceedings of ICDE Workshops.

Yu, J., Wu, J., Sarwat, M. (2016) A demonstration of GeoSpark: A cluster computing framework for processing big spatial data. Proceedings of ICDE.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S., Stoica, I. (2012) Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Proceedings of NSDI.

Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I. (2016) Apache Spark: a unified engine for big data processing. Commun. ACM 59(11): 56-65.

