Summit: A Scalable System for Massive Trajectory Data Management

Louai Alarabi
PhD Student, University of Minnesota, Minneapolis, MN, USA
Email: louai@cs.umn.edu, ACM Member Number: 5798098, SRC: Graduate, Advisor: Mohamed F. Mokbel

ABSTRACT

MapReduce frameworks, e.g., Hadoop, have been used extensively in different applications that include machine learning, and spatial processing. In meantime, huge volumes of spatio-temporal trajectory data are coming from different sources over sometime, raised the demand to exploit the efficiency of Hadoop, coupled with the flexibility of the MapReduce framework, in trajectory data processing. This work describes Summit; a full-fledged MapReduce framework with native support for trajectory data.

1 INTRODUCTION

The importance of processing trajectory data is growing with the emerging and popularity of applications that produce them in large-scale [3]. For example, NASA generates over 4-TB of stars and asteroids moving in the space on a daily basis. Sloan Digital Sky project collects over 156 TB from millions of outer-space objects. MoveBank project archives more than 20 years of habitat trajectory data. New York City Taxi and Limousine Commission record over 1.1 Billion trajectories. National Hurricane Center stores comprehensive details of all storms' movement every year. Brain study in neuroscience model neuron fiber as a trajectory that creates petabytes of data. The explosive increase in data volumes raises the demand for managing and analyzing these mass archives of trajectories on big distributed platforms.

Scaling trajectory data received more attention, especially in utilizing big distributed platforms. The latest efforts for processing trajectory data are built on the top of big distributed frameworks, such as on-top of Hadoop [4], or *Heterogeneous* multiple platforms [2]. Using general purpose framework as-is will result in sub-performance for trajectory applications that require indexing, mainly because they store data as non-indexed heap files. For example, a most recent research study investigated the *k*NN join query on Hadoop employed five isolated map-reduce jobs to execute a single *k*NN join operation without indexing trajectory [4]. In our proposed Summit we achieve orders of magnitude better performance when spatio-temporally indexing trajectories.

Exploit the efficiency of Hadoop, coupled with the flexibility of the MapReduce framework for processing trajectory raised many challenges. Some of the most significant challenges of processing trajectory data is the inability of Hadoop to preserve the spatio-temporal locality, load balancing efficiency, and the capability to support various trajectory operations. We propose Summit system that overcome all these challenges by spatio-temporally loads and partitions trajectory data. We equipped Summit with fundamental

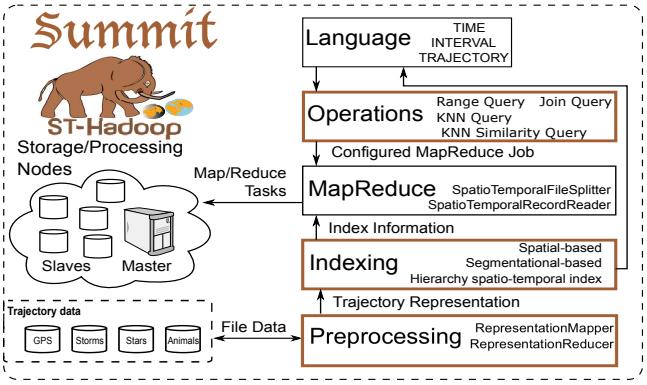


Figure 1: ST-Hadoop system architecture

operations such as Range, *k*NN, Similarity, and Join queries. Summit is an extension of Hadoop framework that built-in the spatio-temporal locality of trajectory in the base code of three layers inside ST-Hadoop [1], namely preprocessing, indexing, and operation layers. The key point behind the performance gain of Summit is the idea of indexing, where data are spatiotemporally loaded and divided as trajectory segments across computation nodes.

2 FRAMEWORK OVERVIEW

Figure 1 gives the high level architecture of our Summit; with a *built-in* support for trajectory data type and operations. Summit cluster contains one master node that breaks up a map-reduce job into smaller tasks, carried out by slave nodes. Summit adopts a layered design of ST-Hadoop briefly described below:

Language Layer: This layer provides a simple high-level SQL-like language that supports trajectory data types (i.e., TRAJECTORY, which described as a consecutive sequence of spatio-temporal points ST-POINT. Operations contains (e.g., OVERLAP and KNN).

Preprocessing Layer: This layer is responsible for reconstructing the raw representation of objects into trajectory segments, where each contains a continuous sequence of spatio-temporal points.

Indexing Layer: Summit employs a two levels index structure of *global* and *local* indexing. The *global* index partitions the data across the computation nodes, while the *local* index organizes the data inside each node. Space and time of trajectories are taken into consideration in each level.

MapReduce Layer: The primary task of this layer is to exploit the global and the local indexes, respectively, for data pruning. Summit utilized ST-Hadoop implementations to the MapReduce layer, and thus, it is not going to be discussed any further in this paper.

Operations Layer: This layer encapsulates the implementation of three common trajectory operations, namely, range query, *k*NN, and join queries. More operations can be added to this layer.

3 TRAJECTORY INDEXING

Summit organizes input files in the Hadoop Distributed File System (**HDFS**) in a way that preserves the spatio-temporal geometrical shape of trajectory. Hence, incoming trajectory operations can have minimal data access to retrieve the query answer, reduce the computation complexity, and allow applications to run more sophisticated operations on the entire trajectory or sub-trajectories; thus, more in-depth information gained. Summit indexes trajectory through the following four consecutive phases:

1. Preprocessing: Summit triggers this process to converts the trajectory into a segment representation. Each segment consists of a consecutive sequence of points in geographical space and time. This phase is necessary to assemble trajectories information in a singleton representation.

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

3. Bulkloading Partitioning: Summit manipulates the sample to constructs two-level indexing of temporal and spatial, respectively. Summit applies the temporal partitioning already equipped in ST-Hadoop to partitions the temporal dimension of trajectories into either equi-width or equi-depth [1]. As for the spatial level of indexing, Summit employs space or data partitioning algorithms for every temporal interval, namely *Spatial-based* or *Segmentational-based*. Figures 2 illustrates the logical design of both techniques. Rectangles represent the boundaries of the HDFS partitions. Dots and lines depict the trajectory information. Tables below list the contents of each partition. Described as follow:

• *Spatial-based*: This partitioning preserves the spatio-temporal locality closeness between sub-trajectories. The boundaries of the HDFS partition cut trajectory connectivity as shown in figure 2a. This organization of trajectory assists basic operations, such as range and join queries.

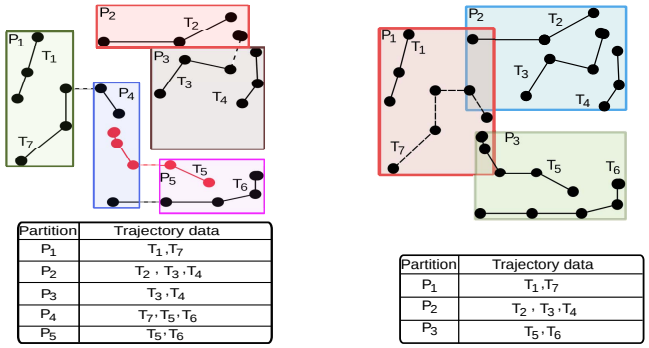
• *Segmentational-based*: This guarantee that the full information of nearby trajectories is organized in a single HDFS block, as shown in figure 2b. This technique preserves the spatio-temporal locality and shapes of trajectories. Such partitioning is more in favor for operations that not only need to process the spatio-temporal locality of trajectories but also their semantic or shapes over time, such as Similarity *k*NN query.

4. Physical Assigning: The objective of this phase is to scan through the whole data and assign each record according to the layout constructed from the previous phase.

4 CONTRIBUTION AND RESULT

The partitioning in Summit is the key feature of its superior performance over Hadoop. Preliminary experiments conducted on New York taxi dataset, to show that Summit achieves orders of magnitude higher job throughput. We present three case studies of trajectory operations that utilize Summit indexing, namely, range, *k*NN point-based, and *k*NN similarity-based queries.

• **Range query**: Given a spatio-temporal query predicates, the query retrieves all trajectories that belong to the query region in both space and time. For example, "*find taxi in downtown Seattle during time interval between January and March 2018.*



(a) Spatial-based (b) Segmentational-based

Figure 2: Trajectory Partitioning

• ***k*NN point-based**: Given a query point and time interval, the goal is to find the *k* nearest trajectories to a point during some time interval. For example, "*Find the closest four animals to a Minnehaha waterfall between August and September*".

• ***k*NN Similarity-based**: Given a query trajectory, the objective is to find the *k*NN to the whole trajectory points for every time instance according to some aggregate or similarity function, such as MinMax. For example, "*Find the similar kNN Taxi to a given trajectory Trj*".

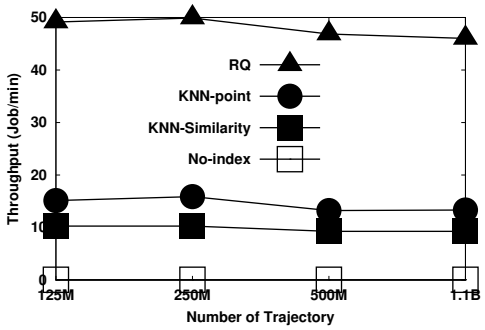


Figure 3: Preliminary Results

In figure 3, we increase the number of moving objects from 125 Million to 1.1 Billion, while measuring the job throughput per minute. We average the execution time of 20 randomly submitted queries for each operation. Summit index has more than two orders of magnitude higher throughput than on-top of Hadoop implementations, consistently, due to the early pruning employed by its index. We run the range and *k*NN point-based queries on the spatial-based trajectory partitioning. Meanwhile, we utilize the segmentational-based for *k*NN similarity query.

REFERENCES

[1] Louai Alarabi, Mohamed F. Mokbel, and Mashaal Musleh. 2018. ST-Hadoop: a MapReduce Framework for Spatio-temporal Data. *GeoInformatica* 22, 4 (2018), 785–813. https://doi.org/10.1007/s10707-018-0325-6

[2] Jie Bao, Ruiyuan Li, Xiuwen Yi, and Yu Zheng. 2016. Managing massive trajectories on the cloud. In *SIGSPATIAL*.

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

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