DISTIL: A Distributed In-Memory Data Processing System for Location-Based Services

Maria Patrou, Md Mahbub Alam, Puya Memarzia, Suprio Ray, Virendra C. Bhavsar, Kenneth B. Kent, and Gerhard W. Dueck
University of New Brunswick, Fredericton, Canada
{mpatrou,malam5,pmemarzi,sray,bhavsar,ken,gdueck}@unb.ca

ABSTRACT

Location-based services (LBS) have become an ubiquitous technology and spatio-temporal data generated by LBS is characterized by high volume and velocity. In recent times several projects, such as GeoSpark, SpatialSpark and LocationSpark, have focused on developing spatial data systems that take advantage of the distributed in-memory data processing capability of Spark. However, most of these systems assume immutable spatial data, and they do not support high throughput location data updates that are common in LBS. On the other hand, a few HBase-based systems, such as MD-HBase, have been proposed that support data updates. However, these systems do not take advantage of any distributed in-memory query processing frameworks.

To address the challenges of high velocity location data, we propose DISTIL, a distributed in-memory spatio-temporal data processing system. Our system includes a distributed in-memory index and storage infrastructure that are built on a distributed in-memory programming paradigm called APGAS (Asynchronous Partitioned Global Address Space). In our system, the location records are distributed across a cluster of nodes, using the producer-consumer model. Our experimental evaluation demonstrates that DISTIL can support high throughput location updates, and low latency concurrent processing of spatio-temporal range queries.

CCS CONCEPTS

• **Information systems → Database query processing**; **Location based services**; *Data streaming*;

KEYWORDS

spatio-temporal; LBS; distributed in-memory; index

ACM Reference format:
Maria Patrou, Md Mahbub Alam, Puya Memarzia, Suprio Ray, Virendra C. Bhavsar, Kenneth B. Kent, and Gerhard W. Dueck . 2018. DISTIL: A Distributed In-Memory Data Processing System for Location-Based Services. In *Proceedings of 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 6–9, 2018 (SIGSPATIAL '18),* 4 pages.
https://doi.org/10.1145/3274895.3274961

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11.
https://doi.org/10.1145/3274895.3274961

1 INTRODUCTION

The growth of spatial data is accelerating and a significant portion of spatial data is spatio-temporal in nature. A wide range of Location-Based Services (LBS) applications, including location-aware search, location-based games, advertising, and weather services, have emerged. The key characteristics of LBS are: high rate of location data updates, and many concurrent location-oriented queries, such as range queries and *k* Nearest Neighbor (kNN) queries.

In response to the need to support scalable processing of spatial data, a number of systems have been developed. Among these, SpatialHadoop [4] and Hadoop-GIS [8] are based on the open-source MapReduce framework, Hadoop [7]. Hadoop focuses on fault-tolerance and I/O operations. Due to technological advances in computing hardware, large main-memory machines with multiple cores, and low-latency networking are becoming quite common in modern data centers. These trends have made distributed in-memory processing of big data attractive and the Spark [24] framework very popular. Recently, several Spark-based spatial data processing systems have been proposed. These include GeoSpark [23], SIMBA [21], SpatialSpark [22], LocationSpark [19] and STARK [9]. However, according to a study [18], only STARK supports spatio-temporal data. Moreover, none of these systems support a high rate of updates, as most of the systems assume immutable spatial data. As a result, these distributed in-memory spatial big data frameworks are not suitable for LBS applications.

There have been a few distributed spatio-temporal data management systems that are designed for LBS. They are based on the distributed NoSQL data store HBase [10]. MD-HBase [12] is among the first in this category. Local and Clustering Index (LC-Index) is a more recent entry in this series [5]. However, these systems do not take the advantage of low latency distributed in-memory processing, as Spark-based systems do.

Our goal is to support LBS applications, by offering high throughput location updates and low latency distributed in-memory processing of many concurrent queries. To this end, we considered a few distributed in-memory frameworks and abstractions offered by them. While Spark is very popular, researchers have reported that [1] analytics applications developed with Spark can be an order of magnitude slower compared to handwritten implementations written with high performance language and tools. Spatial data systems that are based on Spark also inherit its performance overhead associated with scheduling, distributed coordination, and data movement. We considered alternatives to Spark, including APGAS (Asynchronous Partitioned Global Address Space) [16], which could potentially offer better performance over Spark-based systems.

We propose a system called *DISTIL* (Distributed In-memory Spatio-Temporal data system for Location-based services). Our

system is developed with X10 [2], a language built upon the APGAS model, that also supports failure-aware programming. It utilizes a distributed in-memory array abstraction to store data in the main memory of the available nodes in the cluster. We propose a distributed in-memory spatio-temporal index for efficient multi-dimensional range query processing. To support high location update throughput, each node utilizes a persistent *Local store* based on LSM-tree [13]. To be able to tolerate node failure(s), the data in the local store at each node is periodically uploaded into and synchronized with a distributed persistent *Global store*, which is based on HDFS [17]. In our approach, the spatial domain is partitioned into smaller equally-sized tiles. The insertion of location data into the index and the local store happens at each node that is responsible for a particular spatial region. The dataset and index is split across the machines, but everything is under the same address space using the APGAS programming model. This enables our system to support parallel processing of location-oriented queries by splitting a task among the workers, so that each machine can independently operate on the data it hosts. The only need for inter-node synchronization appears when gathering and aggregating the query results.

Experimental evaluation of DISTIL demonstrates that our system can achieve high update and query throughput. We also compared the performance of our system against GeoSpark, as it is one of the most active Spark-based projects. The average query latency with DISTIL was an order of magnitude lower than GeoSpark.

To summarize, our contributions are as follows:

• We propose a scalable spatio-temporal big data system, called DISTIL, that is based on the APGAS model.

• We propose a novel distributed in-memory spatio-temporal index that supports efficient concurrent location-oriented queries.

• We conduct experiments to demonstrate the scalability and efficiency of our system.

2 BACKGROUND AND RELATED WORK

Asynchronous Partitioned Global Address Space (APGAS). APGAS is a distributed shared memory concurrency model that allows multiple threads of execution to share a common address space, distributed across a cluster of nodes. The global address space is physically partitioned and shared among different entities.

X10 Programming Language. X10 [2] is an object-oriented, statically typed and garbage-collected programming language based on the APGAS model. X10's managed runtime offers a distributed heap that is appropriate for parallel and distributed computing. Lightweight threads are spawned asynchronously or synchronously and access objects on their local or remote heap. X10's distributed arrays can split data across multiple nodes, and a global reference facilitates inter-node access. These features allow our system to distribute data across multiple nodes, leverage inter-node and intra-node concurrency, and manage synchronization.

Big Spatial Data Processing Systems. The need for high performance spatial data processing has motivated many projects to add support for spatial queries in existing distributed frameworks. Among these, SpatialHadoop [4] and Hadoop-GIS [8] are based on the MapReduce [3] framework. Another family of systems are based on Spark, an in-memory cluster computing framework [24]. These

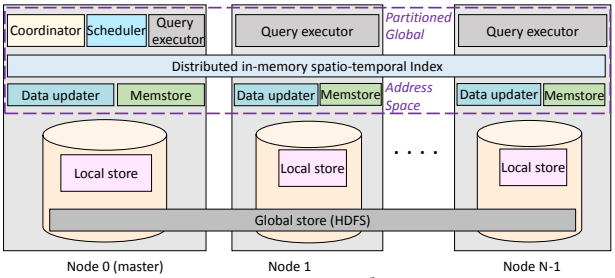

Figure 1: System Architecture

systems include GeoSpark [23], SIMBA [21], SpatialSpark [22], and LocationSpark [19]. Both families of systems are designed for spatial queries, and do not normally support spatio-temporal queries.

Scalable Spatio-Temporal Indexes. Spatio-temporal data is common in many LBS applications. Many systems have been developed to facilitate efficient queries and storage. MD-HBase [12] is a multi-level index for multi-dimensional data that uses HBase. It combines geographical and timestamp data into a single dimension index using the Z-ordering linearization. The PASTIS [15] is an in-memory index in which the spatial domain is split into grid cells. It maintains information about moving objects using compressed bitmaps. Inserts and updates are performed concurrently using multiple threads. Fox et al. [6] developed a spatial-temporal index on Apache Accumulo based on Niemeyer's geohashing technique. The location of a point is represented by a 35-bit hash that contains the location and date-time information of that record. Local and Clustering Index (LC-Index) [5] is an index built on HBase. The records are stored in Index Files (IFile) at different region servers that contain partitions of the table.

3 OUR SYSTEM

In this section we describe our system architecture, index structure, as well as, update and query processing schemes.

3.1 System Overview

The overall system architecture of DISTIL is shown in Figure 1. To enable low latency data processing, it aggregates the main memory capacities of the nodes in a cluster using the APGAS model. It follows a master-slave architecture, where the master node is responsible for coordinating the various activities around the slaves. It queues the records received from GPS-enabled mobile devices and sensors. The *Coordinator* redirects the records to the *Data updater* component in the slaves, based on the data placement. Managing a location update involves first inserting it into the in-memory and persistent stores, and then updating the in-memory index. The index organization is described in Section 3.3.

When a location-oriented query is issued by a client, it is handled by the *Scheduler* component. It generates the query plan and distributes the plan fragments to the *Query executor* in each slave. It also generates the final query result, by aggregating partial results from each *Query executor*.

Location updates received from moving objects are stored in a table, *LTable*. A unique record id (to populate the *RecordId* field) is generated by our system upon receiving a new location record from an object. The table is implemented as a distributed in-memory

store, (*Memstore* in Figure 1), and persisted in the stable storage with persistent stores. To ensure that no location update is lost, even in the event of node or disk failure, our system incorporates two persistent stores: a fast *Local store* hosted at each node and a *Global store* based on HDFS. Each node stores the location records corresponding to the spatial partitions (described in Section 3.2) it hosts. A new location record is serialized and is stored in the *Local store*. To support fast insert operations, our implementation is based on LSM-tree [13]. Data is transferred from the *Local store* to the *Global store* by using an offline process, which runs periodically.

3.2 Data Partitioning

In order to manage data that is larger than a single node's memory capacity, it must be partitioned. In our system, the spatial domain is organized as a grid by splitting it into equally-sized grid cells (tiles). These tiles are distributed using a tile placement policy among the nodes in a cluster. The master node stores information regarding the location of the tiles with a hashtable that maps each tile (*tileId*) to the node (*nodeId*) it belongs to.

We assume that each APGAS place corresponds to a physical node in the cluster. Each place (including the place 0) hosts a subset of tiles, as shown in Figure 2, and a hashtable maintains the mapping between each *tileId* and the tile object. These hashtables are stored in a distributed array whose index is the corresponding node id.

3.3 Distributed In-Memory Spatio-Temporal Index

The main idea behind our index (Figure 2) is to discretize the spatial, as well as the temporal dimension. Our index is composed of a spatial index (SI) and a collection of partial temporal indexes (PTI). We assume that there are N nodes such that each node is mapped to an APGAS place, where place 0 is mapped to the master node. The spatial domain is organized as a regular grid. Each grid cell (tile) is mapped to a place and hence, a place (i.e. its corresponding node) is responsible for a particular set of tiles and there is no need for synchronization among different places while processing a query. On system start-up, the tiles are assigned to the available nodes following a placement policy.

Each tile maintains a PTI for a set of discrete time intervals, and the interval length is a configurable parameter. Essentially, a PTI identifies the activities of each moving object that was inside a grid cell. Specifically, each PTI maintains an interval table. Each entry in the interval table is a tuple consisting of a bit-vector and a hashtable (RID-list map) that maps each moving object to a record id list. A bit in the bit-vector is set to 1, if an object was inside the tile during the corresponding time interval. Otherwise, the bit is set to 0. The RID-list for an object contains the record id of each location update that was inside a tile during that time interval. These record ids can be used to retrieve the actual location record that is in the *LTable*. The reason for keeping the record ids is to process tiles that are partially covered by a query spatial window.

3.4 Update Processing

The update processing involves inserting a new record into the in-memory and persistent stores, as well as updating the spatio-temporal index. When a new location update is received from a moving object, a corresponding location record object *LRec* is inserted into a concurrent queue by the *Coordinator* component in Figure 1. Further processing is carried out using the producer-consumer paradigm to enable parallel processing.

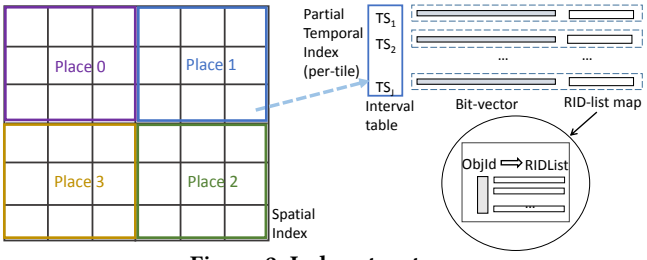

Figure 2: Index structure

One of the producer activities retrieves a particular location record from the queue and based on the coordinates of that record, it determines the tile id *tileId* and the place *pl* where the record needs to be inserted into. Each producer keeps a fixed size array, *arrayPerPlace* for each place. When the size of that *arrayPerPlace* reaches a specific threshold (*insertBatchSize*), it is sent to the node as one item (record-batch), and placed in a queue. When a consumer receives a record-batch item from the queue, it processes its records by inserting the record into their in-memory table *LTable* and persistent local store and then it updates the index.

3.5 Query Processing

We support spatio-temporal range queries (STRQ). The idea behind parallel query processing of STRQ is to isolate the calculations to the nodes containing the relevant tiles. For each relevant place a query object is instantiated with the required parameters. These query fragments are executed at each place using the related tiles. Results are obtained and aggregated into a final result by the master node, with no need for communication during the execution of the place-specific queries.

4 EVALUATION

In this section we describe the evaluation of DISTIL.

4.1 Experimental Setup

We used datasets that are based on real-world spatio-temporal data. The three datasets consist of 10, 20 and 40 million records that track the location and movement of objects across time. The details of these datasets are shown in Table 1. The dataset generation process is described next. The polyline (edges) shapefiles of Texas from the TIGER dataset [20] were fed into the mobility trace generator MOTO [11] to generate the traces. We modified MOTO to generate multiple trace files, each file for a different table fragment. Mobility traces were generated for each object for 1000 timestamps (equivalent to 1000 minutes).

The experiments were conducted on a cluster of 8 machines, each having an Intel(R) Xeon(R) CPU E5472 @ 3.00GHz with 8 cores, 16GB RAM, and 500GB HDD running on Ubuntu 14.04 64-bit OS with Oracle JDK `1.8.0_77` and the X10 2.6.0 compiler.

Table 1: Datasets

Dataset Name	10M	20M	40M
Number of objects (thousands)	10	20	40
Number of records (millions)	10	20	40
Size (MB)	454	907	1813

Table 2: Parameter settings

Parameter	Settings
Space domain	Texas, 1251 km x 1183 km
Num. of road segments	56832846
Time duration, *timestamps*	1000
Insert workers	4

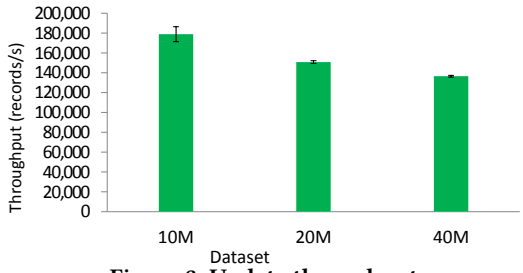

Figure 3: Update throughput

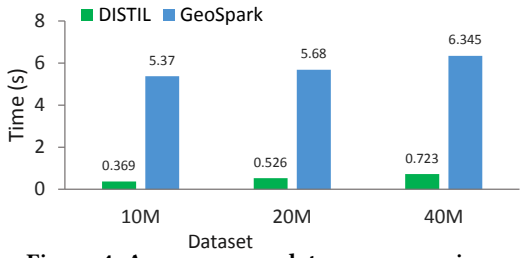

Figure 4: Average query latency comparison

4.2 Update Performance

The update operation entails inserting a new record into the in-memory and persistent stores, as well as updating the spatio-temporal index. To evaluate the update performance of DISTIL, we used 4 insertion workers and a batch size of 6000 records. A multi-dimensional range partitioning policy is used to partition the grid cells and assign them to nodes. Figure 3 shows the insert throughput achieved by DISTIL. As shown, the update throughput is over 178K records/second with the 10M dataset and 136K records/second with the 40M dataset. The throughput is an order of magnitude better than that of a popular relational database DbX reported in [14].

4.3 Query Performance

Although DISTIL supports spatio-temporal range queries, GeoSpark does not support temporal data. Therefore, to do a fair comparison of DISTIL with GeoSpark, we evaluated spatio-temporal range queries with all records (the entire time range) in the 3 datasets. In Figure 4 we present the average query latencies for DISTIL and GeoSpark, while executing a batch of 64 queries. DISTIL is an order of magnitude faster than GeoSpark with these queries.

5 CONCLUSION AND FUTURE WORK

Due to the massive growth of spatio-temporal data, there is crucial need to develop scalable data systems for LBS. Key requirements of LBS are support for high location update (insertion) throughput and many concurrent location-oriented queries.

We presented DISTIL, a distributed in-memory data processing system for managing spatio-temporal data. Our system is based on the APGAS model and implemented with the X10 language. DISTIL can utilize the aggregate memory capacity of a cluster of machines.

Experimental evaluation demonstrates that DISTIL can support scalable LBS with high insertion throughput. Moreover, query latency is significantly lower with DISTIL than GeoSpark.

In future, we would like to explore different tile placement policies. We will also consider load-balancing algorithms to handle data skew. Currently, our system supports spatio-temporal range queries. Our plans for future work also include support for kNN queries and inter-query parallelism.

REFERENCES

[1] Michael Anderson, Shaden Smith, Narayanan Sundaram, Mihai Capotă, Zheguang Zhao, Subramanya Dulloor, Nadathur Satish, and Theodore L. Willke. 2017. Bridging the Gap Between HPC and Big Data Frameworks. *PVLDB* (2017).

[2] Philippe Charles, Christian Grothoff, Vijay Saraswat, Christopher Donawa, Allan Kielstra, Kemal Ebcioglu, Christoph von Praun, and Vivek Sarkar. 2005. X10: An Object-oriented Approach to Non-uniform Cluster Computing. In *OOPSLA*.

[3] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. of the ACM* (2008).

[4] Ahmed Eldawy and Mohamed F Mokbel. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *ICDE*.

[5] Chen Feng, Xi Yang, Fan Liang, Xian-He Sun, and Zhiwei Xu. 2015. LCIndex: A Local and Clustering Index on Distributed Ordered Tables for Flexible Multi-dimensional Range Queries. In *ICPP*. 719–728.

[6] Anthony Fox, Chris Eichelberger, James Hughes, and Skylar Lyon. 2013. Spatio-temporal indexing in non-relational distributed databases. In *IEEE Big Data*.

[7] Hadoop [n. d.]. Apache Hadoop. http://hadoop.apache.org/. ([n. d.]).

[8] Hadoop-GIS [n. d.]. http://bmidb.cs.stonybrook.edu/hadoopgis/index. ([n. d.]).

[9] Stefan Hagedorn, Philipp Götze, and Kai-Uwe Sattler. 2017. The STARK Framework for Spatio-Temporal Data Analytics on Spark. *BTW 2017* (2017).

[10] HBase [n. d.]. https://hbase.apache.org/. ([n. d.]).

[11] MOTO [n. d.]. . http://moto.sourceforge.net/. ([n. d.]).

[12] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. 2011. MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. In *MDM*.

[13] O'Neil, Patrick and Cheng, Edward and Gawlick, Dieter and O'Neil, Elizabeth. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* (1996).

[14] Suprio Ray, Rolando Blanco, and Anil K. Goel. 2013. Enhanced Database Support for Location-based Services. In *IWGS @ ACM AIGSPATIAL*.

[15] Suprio Ray, Rolando Blanco, and Anil K. Goel. 2014. Supporting Location-Based Services in a Main-Memory Database. In *MDM*.

[16] Vijay Saraswat, George Almasi, Ganesh Bikshandi, Calin Cascaval, David Cunningham, David Grove, Sreedhar Kodali, Igor Peshansky, and Olivier Tardieu. 2010. The asynchronous partitioned global address space model. In *AMP*.

[17] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The hadoop distributed file system. In *IEEE MSST*.

[18] Stefan Hagedorn and Philipp Götze and Kai-Uwe Sattler. 2017. Big Spatial Data Processing Frameworks: Feature and Performance Evaluation. In *EDBT*.

[19] Mingjie Tang, Yongyang Yu, Qutaibah M Malluhi, Mourad Ouzzani, and Walid G Aref. 2016. Locationspark: A distributed in-memory data management system for big spatial data. *PVLDB* (2016).

[20] Tiger® [n. d.]. http://www.census.gov/geo/www/tiger. ([n. d.]).

[21] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient in-memory spatial analytics. In *SIGMOD*.

[22] Simin You, Jianting Zhang, and Le Gruenwald. 2015. Large-scale spatial join query processing in cloud. In *ICDEW*.

[23] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. Geospark: A cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*.

[24] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud*.