http://dx.doi.org/10.5392/IJoC.2015.11.1.069

Distributed Indexing Methods for Moving Objects based on Spark Stream

Yunsou Lee
DataStreams R&D Center
Sampyeong-dong Bundang-gu, Songnam-si, Gyeonggi-do, 463-400 Republic of Korea

Seokil Song
Department of Computer Engineering
Korea National University of Transportation, 50 Daehak-ro, Chungju-si, 380-702, Republic of Korea

ABSTRACT

Generally, existing parallel main-memory spatial index structures to avoid the trade-off between query freshness and CPU cost uses light-weight locking techniques. However, still, the lock based methods have some limits such as thrashing which is a well-known problem in lock based methods. In this paper, we propose a distributed index structure for moving objects exploiting the parallelism in multiple machines. The proposed index is a lock free multi-version concurrency technique based on the D-Stream model of Spark Stream. The proposed method exploits the multiversion nature of D-Stream of Spark Streaming.

Key words*: Moving Objects, Spark, Steaming, Index.*

1. INTRODUCTION

For the several years, as the mobile devices such as smartphones and tablet PCs have been prevalent, GPS technologies and the communication infrastructure enable the geo-positioning of the mobile devices. As a result of this development, mobile location-based services stand out as a particularly successful class of Internet applications. Subsequently, server-side technologies to process massive, location-related query and update workloads caused by large populations of users (moving objects) have been crucial.

Usually, moving objects periodically send their positions to servers. If an application needs more accurate positions of moving objects, the transmission period should be shorter. If the number of moving objects or the required accuracy is increased, the server's workload, also, is increased. For example, when the number of moving objects is 6M and the transmission period is 1 minutes, the server's workload is 0.1M updates per second. At the server, this results in one update every 10 microseconds.

Such server workloads is not able to be processed by disk based data structures and algorithms. Several in-memory indexing techniques for moving objects have been proposed as the alternatives of disk based techniques [1]-[4]. The versatile and query efficient R-tree was made cache-conscious [1] to optimize the use of the fast CPU caches. MOVIES [3] is based on frequently building short-lived throwaway indexes where the query result staleness is traded for both update and query efficiency.

Also, [4] proposed a main memory index that is capable of exploiting the inherent parallelism available in modern multicore processors. The main challenge of [4] is to avoid the contention between queries and updates that conventional indexing and locking techniques use in order to maintain a consistent database state and return correct query results. It propose a new parallel main-memory spatial index structure that avoids the trade-off between query freshness and CPU cost. In order to achieve this, it uses techniques that enable light-weight locking, thus locking as little data as possible for as short time as possible, and avoiding the overhead of heavy-weight locks

However, we believe that the lock based methods have some limits. Thrashing is a well-known problem in lock based methods. Also, we believe throughput can be increased by exploiting the parallelism with multiple machines, not only multicore processors. In this paper, we propose a distributed index structure for moving objects exploiting the parallelism in multiple machines. Also, the proposed index structure is not based on locking techniques to increase throughputs.

The proposed indexing method is based on Spark Stream [5]. Spark Stream runs each streaming computation as a series of deterministic batch computations on small time intervals. The time intervals are as low as half a second, and end-to-end latencies below a second. This model is called as discretized streams (D-Streams). D-Streams execute computations as a series of short, stateless, deterministic tasks. They then

* *Corresponding author, Email: sisong@ut.ac.kr*
Manuscript received Mar. 09, 2015; revised Mar. 18, 2015; accepted Mar. 25, 2015

represent state across tasks as fault-tolerant data structures (RDDs) that can be recomputed deterministically. This enables efficient recovery techniques like parallel recovery and speculation. Beyond fault tolerance, this model yields other important benefits, such as clear consistency semantics, a simple API, and unification with batch processing.

This paper is organized as follows. In Chapter 2, descriptions of Spark, Spark Streaming and existing distributed/parallel index structures for moving objects. In Chapter 3, we propose a new distributed index structure for moving objects based on Spark Streaming. Then we conclude this paper in Chapter 4.

2. RELATED WORK

Our proposed distributed index structure for moving objects is based on Spark Streaming. The proposed method uses D-Streams approach to process location data stream of moving objects. Therefore, we describe the Spark Streaming in more detail in this section. Also, existing parallel index structures with multi-cores and distributed index structures with multi-nodes will be described.

2.1 Spark and Spark Stream

Apache Spark is an open source and general-purpose engine for large-scale data processing system developed by the UC Berkeley AMP laboratory. It was designed to provide fast and interactive data analysis as an alternative to Hadoop MapReduce. Apache Spark provides primitives for in-memory cluster computing so as to avoid the I/O bottleneck occurred when Hadoop MapReduce repeatedly performs computations for jobs.

In order to provide its performance while retaining the fault-tolerance, locality, and scalability properties of MapReduce, Apache Spark proposed a memory abstraction which is called a Resilient Distributed Dataset (RDD). Existing key-value stores and databases that allow fine-grained updates to mutable state force to replicate data or log across nodes for fault tolerance. These approaches incur overhead for a data-intensive workload. On the other hand, RDDs only allow coarse-grained updates that apply the same operation to many data items (such as map, filter, or join). This approach allows Apache Spark to provide fault-tolerance through recording lineages, which is the history of operations used to build a current dataset.

Apache Spark also provides additional fault tolerance by allowing a user to specify a persistence condition on any transformation which causes it to immediately write to disk. Data locality is maintained by allowing users to control data partitioning based on a key in each record. Apache Spark Streaming is the extension of Apache Spark to process stream data. It divides the live data stream into small batches of sub-seconds. The divided small batches are treated as RDDs, and they are processed by RDD operations. This approach of Spark Streaming is called as Discretized Stream (D-Stream). D-Stream can be recovered by the same recovery mechanisms of RDDs at a much smaller timescale.

Spark Streaming provides two types of operators to let users build streaming programs. Transformation operators produce a new D-Stream from one or more parent streams. These can be either stateless (i.e., only for each interval) or stateful (across intervals). Output operators write data to external systems such as HDFS. Also, Spark Streaming supports the same stateless transformations available in typical batch frameworks.

2.2 Distributed index structures for moving objects on multiple machines

Recently, several studies to process spatial queries based on MapReduce have been proposed. ToSS-it proposed in [6] is a cloud-based index structure. It generates a new index when every location change of the moving objects is updated. Also, ToSS-it employs inter-node and intra-node multi-core parallelism paradigm to build a new index rapidly. [7] proposed kNN join methods based on MapReduce. In this method, the mappers cluster objects into groups and the reducers perform the kNN join on each group of objects separately. In study [8], exact and approximate algorithms in MapReduce to perform efficient parallel kNN joins on large data have been proposed. [8] first proposed block-nested-loop-join (BNLJ) and its improved version using the R-tree indices. Second, it proposed a MapReduce based approximate algorithm that uses space-filling curves (z-values), and transforms kNN joins into a sequence of one-dimensional range searches.

2.3 Parallel index structures for moving objects on single machine

PGrid proposed in [4] is a parallel main memory index structure to process location based query and update workloads populated by very large moving objects. It uses a grid structure to maximize the parallelism of modern processors. PGrid supports long-running queries and rapid updates on a single data structure unlike existing methods that use different data structures for updates and queries.

[4] claims that existing methods that use snapshots have some problems such as stale query results, CPU waste on full snapshots, stop-the-world problem and so on. In order to avoid these problems, the concurrency control method of PGrid is based on hardware-assisted atomic updates and object-level copy. Also, update operations are treated as atomic operations.

There are some studies that exploit the parallelism of GPU (Graphic Processing Unit) [9]-[11]. [11] proposed the repeated processing method for huge amounts of k nearest neighbours (k-NN) queries over massive sets of moving objects, where the spatial range of queries and the position of objects are continuously modified over time. This method uses a hybrid CPU/GPU pipeline that significantly enhance k-NN query processing. In [9], an indexing structure (CKDB-tree) that combines an adaptive cell and KDB-tree was proposed. This method, also, uses GPU to provide a scalable solution to filtering massive steaming location data of moving objects. [10] investigated the use of GPUs to solve a data-intensive problem that involves huge amounts of moving objects. In this study, the time is partitioned into ticks, and the parallel processing of location updates and range queries occurring in a given tick to

the next tick defer is deferred so as to delay slightly the overall computation.

3. PROPOSED DISTRIBUTED INDEXING METHOD FOR MOVING OBJECTS

As mentioned earlier, the proposed distributed indexing method is based on D-Stream model of Spark Streaming. D-Streams provide two types of operators to let users build streaming applications. One is transformation operators. They produce a new D-Stream from one or more parent streams. These can be either stateless (i.e., act independently on each interval) or stateful (share data across intervals). The other is output operators which let the program write data to external systems (e.g., save each RDD to HDFS). D-Streams support the same stateless transformations available in Spark such as map, reduce, groupBy, and join.

The input stream is the postions of moving objects that are transmitted periodically. Spark Stream transforms the input stream into D-Streams. As shown in Figure 1, the input stream is transformed into DS_t, DS_{t+1}, DS_{t+2} and DS_{t+3} continuously by Spark Stream. The proposed indexing method perform bulkLoad and bulkInsert operators on each D-Stream. bulkLoad builds a gird index GI_t with position data in DS_t. In our proposed indexing method, we use grid index to reduce the time for building and updating an index.

Our indexing method does not use lock based concurrency control method. The D-Stream model of Spark Stream is immutable, so update operations and search operations are not performed concurrently on an index. As shown in Fig. 1, an index is updated by bulkLoad and bulkInsert operators, and multiple versions of the index at time t, t+1, t+2 an t+3, which can be accessed by users, are on main memory. Therefore, users can access GI_{t+2}, while GI_{t+3} is being built.

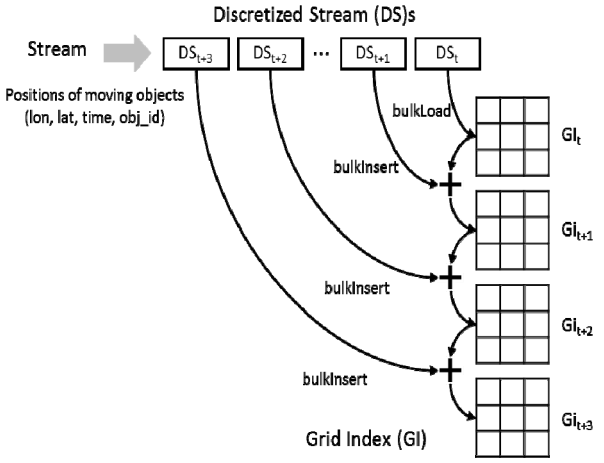

Fig. 1. Build grid index with input stream

The proposed indexing method add some transformation operators and output operators to Spark Stream for indexing and querying moving objects. Table 1 shows our new operators. Our indexing method provides three transform operators and one output operator. In the table, DS_t means that D-Stream at time t and GI_t means that a grid index at time t.

Table 1. Operators of proposed indexing method

Operators	Function
bulkLoad (transform)	Build an initial index and transform DS_t into GI_t
bulkInsert (transform)	Insert a set of position data into an index and transform GI_t with DS_{t+1} into GI_{t+1}
splitIndex (transform)	Split an index into two according to time and transform GI_t into GI_{t_upper} and GI_{t_lower}
search (output)	Operators for querying such as range queries and KNN queries

The proposed indexing method add some transformation operators and output operators to Spark Stream for indexing and querying moving objects. Table 1 shows our new operators. Our indexing method provides three transform operators and one output operator. In the table, DS_t means that D-Stream at time t and GI_t means that a grid index at time t.

We will describe our proposed method with an example. As shown in Fig. 2, the proposed index method consists of time index, gird index and RDD store. Location data of moving objects are divided into small RDDs by Spark Streaming according to a time window, and stored in RDD store. Grid index manages the locations of each objects on the two dimensional grid. Actually, as shown in Fig. 3, Grid index is a list of RDDs whose index numbers are uniquely assigned numbers to cells on the gird. An item of Grid index is RDD numbers that contains moving objects on a corresponding cell. As shown in the figure, the first item of Grid index 1 is R1 and R2 which means RDD1 and RDD2 of RDD Store respectively. It means that location data 1, 4 of RDD1 and 1 of RDD2 are on the cell 1 of Grid index 1.

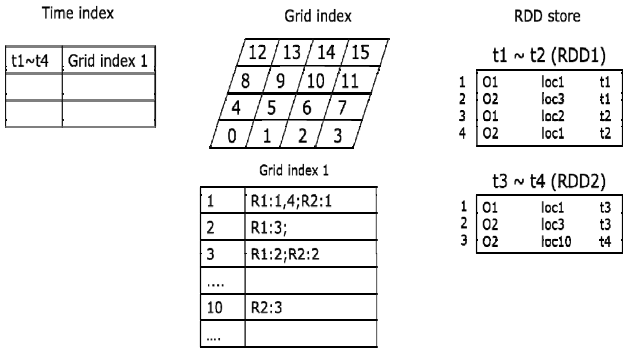

Fig. 2. Architecture of proposed indexing method and an example with 2 moving objects

Time index manages the time interval that each Grid index covers. In Fig. 2, generated RDDs from t1 to t4 are indexed in Grid index 1. After new location data of moving objects are received and new RDDs 3, 4 are generated, Grid index 2 is created to index the new RDDs. Then in Time index Grid index

2 is inserted. In the proposed indexing method, Grid index and Time index also RDDs. Thus when updates are required, new index are generated as RDDs.

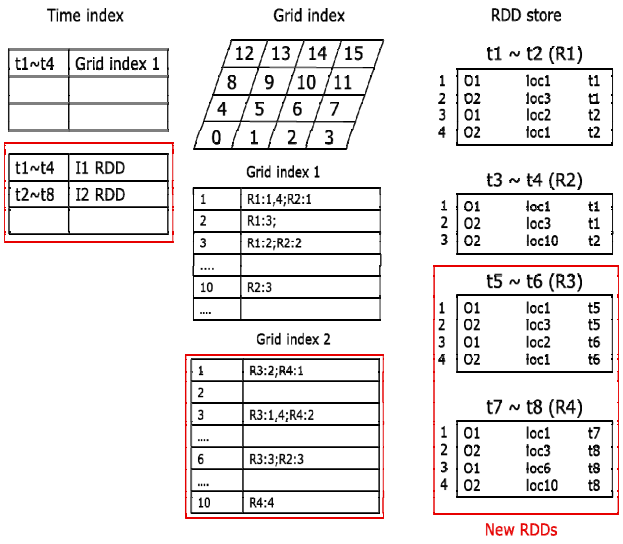

Fig. 3. Architecture of proposed indexing method and an example with 2 moving objects

4. CONCLUSIONS

The proposed distributed indexing method for moving objects based on Spark Stream. The proposed indexing method consists of time index, gird index and RDD store. Time index and grid index are also RDDs in our method. The versions of them are maintained on the memory. With this approach we eliminate lock based concurrency control. Also, we proposed new transform and output operators for Spark Stream such as bulkLoad, bulkInsert, splitIndex and search. bulkLoad, bulkInsert and splitIndex operators are transform operators that make new RDDs. On the other hand, search is output operator to return query results to users. In our future work, we will implement the proposed operators based on Spark Stream and perform various experiments.

ACKNOWLEDGEMENT

This research was jointly supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (NRF-2014R1A1A2059342)

REFERENCES

[1] K. Kim, S. K. Cha, and K. Kwon, “Optimizing Multidimensional Index Trees for Main Memory Access,” SIGMOD Rec., vol. 30, no. 2, 2001, pp. 139-150.
[2] L. Biveinis, S. Saltenis, and C. S. Jensen, “Main-memory Operation Buffering for Efficient R-tree Update,” Proc. VLDB, 2007, pp. 591-602.
[3] J. Dittrich, L. Blunschi, and M. A. V. Salles, “Indexing Moving Objects using Short-lived Throwaway Indexes,” Proc. SSTD, 2009, pp. 189-207.
[4] D. Šidlauskas, S. Šaltenis, and C. S. Jensen, “Parallel Main-memory Indexing for Moving-object Query and Update Workloads,” Proc. ACM SIGMOD, 2012, pp. 37-48.
[5] M. Zaharia, et al, “Discretized Streams: An Efficient and Fault-tolerant Model for Stream Processing on Large Clusters.” Proc. USENIX on Hot Topics in Cloud Computing, 2012, p. 10.
[6] A. Akdogan, C. Shahabi, and U. Demiryurek, “ToSS-it: A Cloud-Based Throwaway Spatial Index Structure for Dynamic Location Data”, Proc. MDM, 2014, pp. 249-258.
[7] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, “Efficient Processing of k Nearest Neighbor Joins using MapReduce,” Proc. VLDB, 2012, pp. 1016-1027.
[8] C. Zhang, F. Li, and J. Jestes, “Efficient parallel kNN Joins for Large Data in MapReduce,” Proc. EDBT, 2012, pp. 38-39.
[9] Z. Deng, X. Wu, L. Wang, X. Chen, R. R. Zomaya, and A. Dan Chen, “Parallel Processing of Dynamic Continuous Queries over Streaming Data Flows,” IEEE Transactions on Parallel and Distributed Systems, vol. 26, issue 3, 2015 , pp. 834-846.
[10] C. S. Jensen, “GPU-Based Computing of Repeated Range Queries over Moving Objects,” Proc. Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 2014, pp. 640-647.
[11] F. Lettich, S. Orlando, C. Silvestri, and C. S. Jensen, “Manycore Processing of Repeated Range Queries over Massive Moving,” CoRR, 2014.

Yunsou Lee
He received the BS and MS degrees in Computer Engineering Department from Korea National University of Korea, Republic of Korea in 2013 and 2015 respectively. He is a researcher of Datastreams, Republic of Korea. His research interests are database systems, storage systems and so on

Seokil Song
He received the BS, MS and PhD degrees in Computer and Communication Department from Chungbuk National University of South Korea in 1998, 2000 and 2003, respectively. He is an Associate Professor of the Computer Engineering Department, Korea National University of Transportation, Republic of Korea. His research interests are database systems, index structures, concurrency control, storage systems, sensor network and XML database.