https://doi.org/10.5392/IJoC.2018.14.1.028

Distributed Moving Objects Management System for a Smart Black Box

Hyunbyung Lee
Dept. of Computer Engineering
Korea National University of Transportation, Daehakro 50, Chungju, Chungbuk, Republic of Korea 27469

Seokil Song
Dept. of Computer Engineering
Korea National University of Transportation, Daehakro 50, Chungju, Chungbuk, Republic of Korea 27469

ABSTRACT

In this paper, we design and implement a distributed, moving objects management system for processing locations and sensor data from smart black boxes. The proposed system is designed and implemented based on Apache Kafka, Apache Spark & Spark Streaming, Hbase, HDFS. Apache Kafka is used to collect the data from smart black boxes and queries from users. Received location data from smart black boxes and queries from users becomes input of Apache Spark Streaming. Apache Spark Streaming preprocesses the input data for indexing. Recent location data and indexes are stored in-memory managed by Apache Spark. Old data and indexes are flushed into HBase later. We perform experiments to show the throughput of the index manager. Finally, we describe the implementation detail in Scala function level.

Key words*: Moving Objects, Distributed System, Index, Spatio-temporal.*

1. INTRODUCTION

The use of GPS enabled mobile devices such as smartphones, smart black boxes, and tablet PCs have been prevalent has increased the volumes of spatio-temporal data. The location and time pair of moving objects is one type of important spatio-temporal data. Usually, moving objects periodically send their locations and time to servers. When the number of moving objects becomes larger and the period becomes shorter, the overhead of processing the locations of moving objects becomes a bottleneck.

If an application of moving objects needs the more accurate positions of moving objects, the transmission period should be shorter. According to [1], when the number of moving objects is 6,000,000 and the transmission period is 1 minutes, the server's workload is 100,000 updates per second. In that case, a server should process one update every 10 microseconds.

To reduce the bottleneck caused by disk-based data structures and algorithms several in-memory indexing methods for moving objects have been proposed [2]-[4]. [2] proposed a cache-conscious R-tree which optimizes the use of the fast CPU caches. MOVIES proposed in [3] is based on frequently building short-lived indexes where the query result staleness is traded for both update and query efficiency. [4] proposed a main memory index that can exploit the inherent parallelism available in modern multicore processors. It avoids 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.

On the other hand, in some studies, parallel and distributed indexing methods to process the location data of moving objects have been proposed [5]-[9]. MDHbase [5], Hadoop GIS [6], Spatial Hadoop [7], Parallel Secondo [8], and Tornado [9]. These methods store and query the large amount of location data based on shared-nothing server cluster. However, still there are bottlenecks caused by disk accesses.

Recently, the significant drop in main-memory cost has initiated a wave of main-memory distributed processing systems. Apache Spark [9] is an open source and general-purpose engine for large-scale data processing system. It 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). 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.

* *Corresponding author, Email: sisong@ut.ac.kr*
Manuscript received Mar. 07, 2017; revised May. 23, 2017; accepted May. 23, 2017

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). DStream can be recovered by the same recovery mechanisms of RDDs at a much smaller timescale.

[1] proposed an indexing method for moving objects based on Spark to manage index and store location data on distributed in-memory. However, it does not consider the case that the memory is full of index structures and location data. It uses grid based indexing techniques. When the memory is full, index structures and location data is processed according to the configuration of Spark.

In this paper, we design and implement distributed in-memory moving objects management system based on Spark. It consists of data and query collector, index manager and data manager. Data and query collector which is designed based on Apache Kafka receives location data and time from vehicles and queries from users. Index manager creates grid based spatio-temporal index structures, and it is enhanced version of [1] which is based on Spark Streaming to consider the case of the full of main memory. Also, the indexing method of this paper provides snapshot isolation level of transactional processing with multi-version concurrency control techniques based on RDDs of Apache Spark. Data manager is to store old index structures to HBase and to load index structures. This paper is an extended version of [13] for implementation detail.

The organization of this paper is as follows. In Section 2, we describe the existing parallel and distributed moving object management methods. In Section 3, the proposed moving object management system is presented in detail, and the performance evaluation results of the proposed system is given in Section 4. In Section 5, we describe the implementation detail and finally, conclude our paper in Section 6.

2. RELATED WORK

Parallel Secondo [8] is a parallel and distributed version of SECONDO database system based on a cluster of computers. It integrates Hadoop with Secondo databases and provides almost all existing Secondo data types and operators. Secondo which is a base system of Parallel Secondo is a database management system to support spatial and spatio-temporal data management. Secondo provides data types and operators to represent and process the queries of moving objects such as vehicles, animals and trajectories. Parallel Secondo becomes possible to process spatio-temporal queries and analysis on the large amount of moving object data and sets of trajectory data in the cloud.

Hadoop [11] can store large amount of ⟨key, value⟩ pair data in its HDFS (Hadoop Distributed File System) [11], and allocate them to the parallel tasks to process, according to the MapReduce programming model. Like Hadoop GIS [6] which is Hadoop-based hybrid systems, Parallel Secondo uses HDFS as the communication way between data and tasks. This kind of systems cause considerable overhead to migrate and shuffle data via HDFS since the spatio-temporal data is multi-dimensional data which is much larger than the traditional data.

Parallel Secondo to reduce the overhead caused by the communication via HDFS proposes native file system, PSFS (Parallel Secondo File System). Also, it uses HDFS stores only a small size of meta data in order to schedule the MapReduce tasks by Hadoop. Consequently, Parallel Secondo avoids the useless data migration overhead.

However, since Parallel Secondo store the spatio-temporal data which is usually much larger than traditional data on hard disk, it is difficult to process the large amount of location stream in real time. To our knowledge, Parallel Secondo is proper for applications to analyze and queries the massive trajectories in batch mode.

On the other hand, Tornado [9] is a distributed system for real-time processing of spatio-textual queries over data streams. Tornado is based on Storm [12] which is a distributed and fault-tolerant general-purpose stream processing system. [9] extends Storm with an adaptive indexing layer that improves the performance of spatio-textual queries.

The main capability of Tornado is semantic search function that go beyond conventional keyword-based matching. It identifies and uses concepts over streaming data in an online fashion in order to determine how semantically related the identified concepts are to the spatio-textual queries [9]. Even though Tornado considers spatio-textual quries over massive data stream, it does not support spatio-temporal queries for moving objects.

[1] proposes an in-memory distributed indexing method for moving objects based on Apache Spark. The basic technique of [1] is simple gird index. [1] adds new transformation operators and output operators such as bulkLoad, bulkInsert, splitIndex, search to index and query moving objects in real-time. The input stream is the location data of moving objects that are transmitted periodically from vehicles. Spark Streaming transforms the input stream into D-Streams.

As shown in Fig. 1, the input stream is transformed into DSt, DSt+1, DSt+2 and DSt+3 continuously by Spark Stream. It performs bulkLoad and bulkInsert operators on each D-Stream. bulkLoad builds a gird index GIt with location data in DSt. [1] uses simple grid techniques to reduce the time for building and updating an index. The 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, the 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 GIt+2, while GIt+3 is being built.

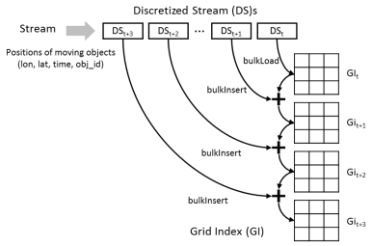

Fig. 1. In-memory Grid Index Structure based on Apache Spark Streaming

3. PROPOSED IN-MEMORY DISTRIBUTED MOVING OBJECTS MANAGEMENT SYSTEM

In this paper, we design and implement distributed in-memory moving objects management system based on Spark. The overall architecture of proposed moving object management system is shown in Fig. 2. As shown in Fig. 2, It consists of Data and Query Collector, Index Manager, Query Manager and Data Manager.

Data and Query Collector which is designed based on Apache Kafka and Spark Streaming receives location data and time from vehicles and queries from users. Data and Query Collector is a consumer of Kafka. Kafka is a broker to deliver location data of vehicles and queries of users to Data and Query Collector. Data and Query Collector retrieves location data and queries continuously from Kafka. When location data of moving objects is received, it partitions DStream of location data according to their cell ids in a grid.

Index Manager creates a grid based spatio-temporal index structure, and insert newly received location data into the index structure. An index structure of the proposed system is divided into multiple sub index structures according to time interval. Index Manager monitors the usage of distributed main memory in real-time and flush some of sub index structures into HBase according to LRU (Least Recently Used) policy.

Query Manager is to process spatio-temporal queries from users. It uses sub index structures which is placed on in-memory and HBase to process the queries. Data Manager is to store flushed sub index structures and load sub index structures to in-memory.

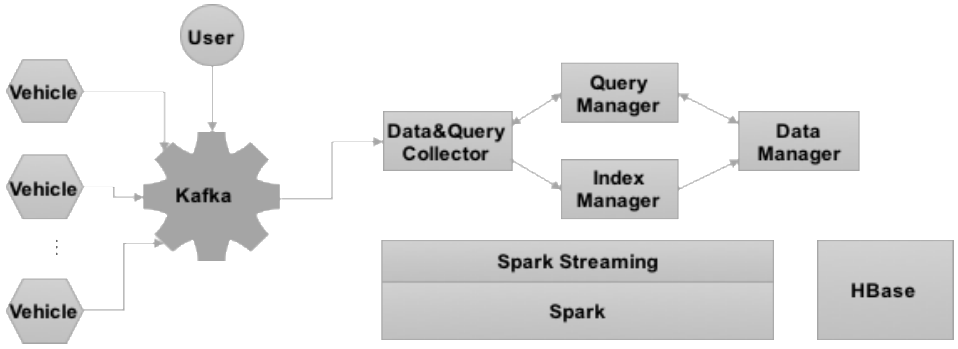

Fig. 1. Architecture of the proposed moving object management system

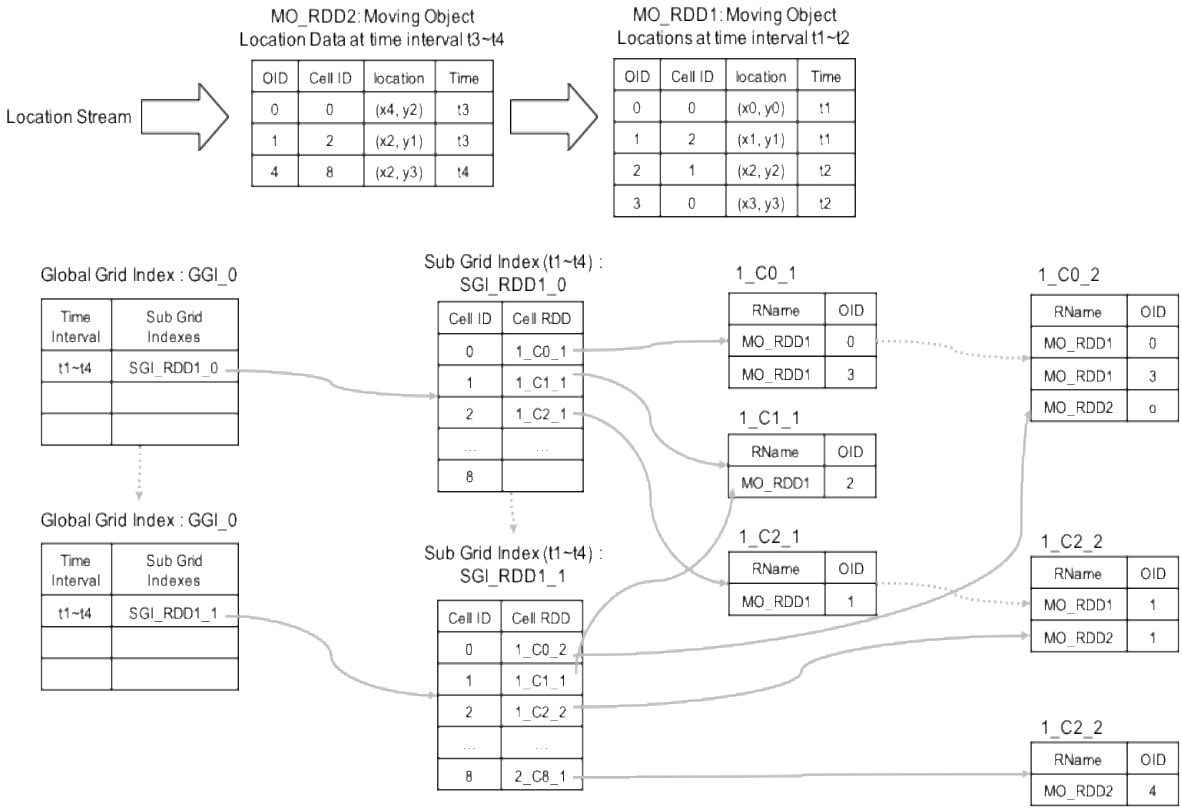

Fig. 3. Index Manager of the proposed distributed in-memory moving object management system

As mentioned earlier, the Index Manager is based on the distributed grid indexing techniques of [1]. Fig. 3 shows Index Manager of the proposed moving object management system. As shown in Fig. 3, Index Manager consists of Global Grid Index and multiple Sub Grid Indexes. The proposed Index Manager splits a Grid Index into multiple Sub Grid Index Structures according to time interval.

Global Index Structure manages multiple Sub Grid Indexes in order to access a specific Sub Grid Index. For example, when location data is newly received, Index Manager needs to access the most recent Sub Grid Index. Also, when a query is received, Index Manager should access one or more Sub Grid Indexes according to time predicate of the query.

Each Sub Grid Index manages location data for each cell. Location data for each cell is managed as a RDD. In Fig. 3, 1_C0_1 is a RDD to store location data for Cell 0 at time interval t1~t4. When a newly received DStream contains location data for Cell 0, Index Manager creates a new RDD 1_C0_2 for the new location data. When a new RDD 1_C0_2 is created, Sub Grid Index at time interval t1-t4, SGI_RDD1_0 is evolved to SGI_RDD1_1.

Each RDD for a Cell like 1_C0_1 contains RDD name (RName) of RDD which stores location data. In Fig. 3, MO_RDD1 and MO_RDD2 store location data of DStream includes location data between t1 and t2 and location data between t3 and t4 respectively.

Index Manager processes spatio-temporal queries concurrently while processing indexing. We guarantee Index Manager always consistent index data because Spark does not allow to access RDDs which is being changed. Also, Spark always recovers a RDD created once with its own fail over method, so when Index Manager accesses RDDs for index structure, the RDDs will be never disappeared.

4. PERFORMANCE EVALUATION

In this paper, we perform experiments to show the scalability of the proposed in-memory distributed moving object management system. Table 1 shows the parameters used in our performance evaluation. We construct a cluster consisting of 4 nodes. The specification of each node is shown in Table 1. Data set for experiments is generated with 1,000,000 synthetic moving objects. It is assumed that each moving object transmits its data to the cluster every 1 minute. We vary the number of nodes from 2 to 4, and the number of executors from 4 to 40.

Table 1. Parameters of Performance Evaluation

Parameters	Specification
Cluster	4 Nnodes - Node 1 : Intel(R) Xeon(R) CPU E5506 @ 2.13GHz 16 Cores, 30 Gbyte RAM - Node 2 : Intel(R) Xeon(R) CPU E5620 @ 2.40GHz 16 Cores, 30 Gbyte RAM - Node 3 : Intel(R) Xeon(R) CPU E31220 @ 3.10GHz 4 Cores, 30 Gbyte RAM - Node 4 : Intel(R) Xeon(R) CPU E31220 @ 3.10GHz 4 Cores, 30 Gbyte RAM
Data	- Number of moving objects : 1,000,000 - Data transmit cycle of moving objects : 1 minute
Software	spark 2.1.0, kafka 0.10.2, hbase 1.3.1
Number of Nodes	2 ~ 4
Number of Executors	4 ~ 40

Fig. 4 shows the results of indexing throughput with varying the number of executors from 4 to 40. We fix the number of nodes as 4 in this experiment. As shown in this figure, as the number of executors increases, the indexing throughput, also, increases from about 60,000/sec to about 120,000/sec. Fig. 5 shows the results of indexing throughput with varying the number of nodes from 2 to 4. In this experiment, the number of executors is fixed as 4. As the number of nodes increases, the indexing throughput increases from about 52,000 to about 58,000.

Through these experiments, we can show the scalability of our Index Manager. It is rather difficult to confirm the scalability since the number of nodes used in our experiments is not sufficient. In our future work, we will perform experiments with more nodes.

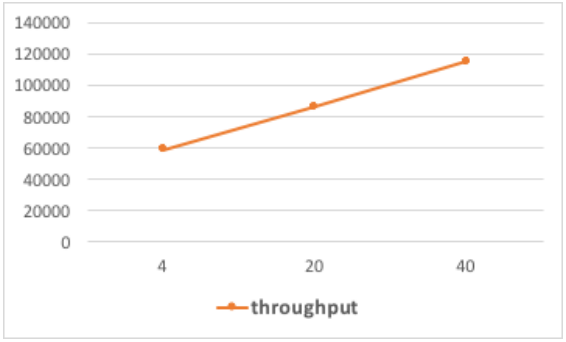

Fig. 4. Indexing throughput with varying the number of executors (the number of nodes is 4)

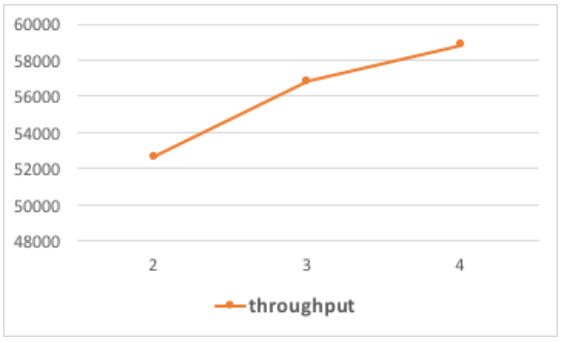

Fig. 5. Indexing throughput with varying the number of nodes (the number of executors is 4)

5. IMPLEMENTATION

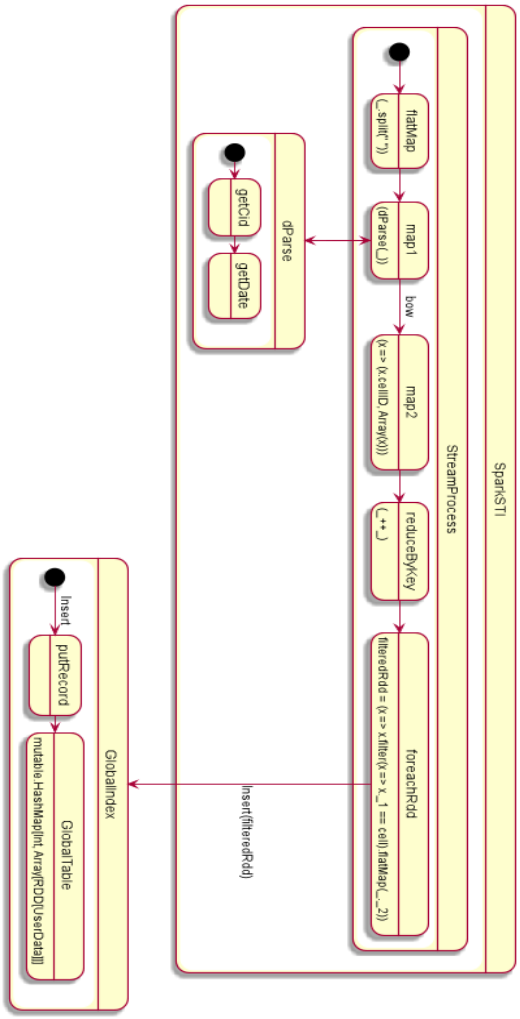

Fig. 6. Lineage of transformations for the implemented moving object management system

Fig. 6 shows the Apache Spark transformation lineage of the implemented smart black box data processing and analyzer. Input stream data via Apache Kafka or Raw Socket is transformed into the required data type via the function dParse. At this time, the input stream becomes Dstream [UserData] through Transformation. The function dParse used in this step processes the input data (string), assigns CellID, and decomposes and reads the date in year, month, day, hour, minute and second.

After converting the data type to Dstream [UserData] through dParse, map and reduceByKey operations are performed on the data. map operation to convert Dstream data into key-value data and then group the data with the same key (CellID) through reduceByKey operation.

The foreachRDD function is used for grouped data. If the Dstream is a set of consecutive RDDs, the foreachRDD function can perform the desired operation by accessing each RDD. foreachRDD first performs a collection operation on the key data. collection is a function that collects data processed by each of the exciters in the spark, and performs a filter function for each key in the RDD with the gathered key. filter extracts only the data satisfying the condition in RDD, and the filtered RDD is input through putRecord function of GlobalIndex.

Fig. 7 shows the function call diagram of the implemented moving object management system. Also, Table 2 shows the function list and input/output and function of each functions.

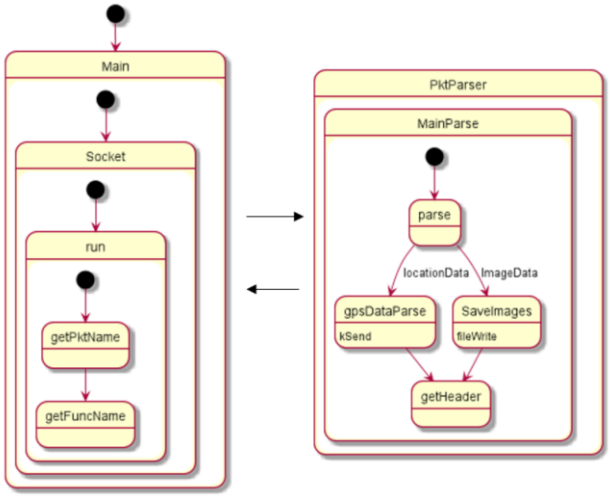

Fig. 7. Block diagram of the implemented moving object management system

Table 2. Function list for the implemented moving object management system

Functions	Descriptions
fileWrite	input - (bytes:Array[Byte], fName:String, ftype:String) receive image, file name from black box and store
SaveImages	input- (bParam:Array[Byte], Loop:Int, ImageStartLoc:Int) output − returnVal receive images and store them
parse	input - (bHeader: Array[Byte], bParam : Array[Byte]) output − ReturnClass decoding received data
getHeader	input - (bHeader: Array[Byte], FuncName:String, ErrCode:Int) output − ReturnClass create header
getPktName	input - (Bytes:Array[Byte]) output − String return received packet header name
MainParser	input - (Bytes:Array[Byte]) output − ReturnClass decoding input byte stream
gpsDataParse	input − (buffer:Array[Byte], len:Int, gpsBuffer:ArrayBuffer[GpsDataView]) decoding location data
chBodyLengh	input - (bytes:Array[Byte], length:Int) resize the size of decoded packet
kSend	input - (data:RowData, topic:String) send input data to Kafka

6. CONCLUSION

In this paper, we designed and implemented a distributed in-memory moving object management system to process the large amount of location stream from moving objects. The proposed system consists of Data and Query Collector, Index Manager, Query Manager and Data Manager. We performed experiments to show the scalability of indexing throughput of our proposed system. From the experimental results, we could know the proposed system is scalable to the number of nodes and executors of Spark. Finally, we describe the implementation detail.

In our future work, we will perform experiments with more nodes, and also, compare with existing moving object management systems.

REFERENCES

[1] H. Li, Y. Lee, and S. Song, "Grid based Distributed In-memory Indexing for Moving Objects," Proceedings of International Symposium on Information Technology Convergence, Jeonju, Republic of Korea, Oct. 30-31 2014.

[2] K. Kim, S. K. Cha, and K. Kwon, "Optimizing Multidimensional Index Trees for Main Memory Access," SIGMOD Record, vol. 30, no. 2, 2001, pp.139-150.

[3] L. Biveinis, S. Saltenis, and C. S. Jensen, "Main-memory Operation Buffering for Efficient R-tree Update," Proceedings of the 33rd 41st International Conference on Very Large Data Bases, Vienna, Austria, Sep. 23-28 2007, pp. 591-602.

[4] D. Šidlauskas, S. Šaltenis, and C. S. Jensen, "Parallel Mainmemory Indexing for Moving-object Query and Update Workloads," Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, Arizona, USA, May. 20-24 2012, pp. 37-48.

[5] S. Nishimura, S. Das, D. Agrawal, and A. E. Abbadi, "MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services," Proceedings of the 2011 12th IEEE International Conference on Mobile Data Management, Lulea, Sweden, Jun. 6-9 2011, pp.7-16.

[6] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. H. Saltz, "Hadoop-GIS: A High Performance Spatial Data Warehousing System over Mapreduce," Proceedings of the 39rd 41st International Conference on Very Large Data Bases, Trento, Italy, Aug. 26-30 2013, pp. 1009-1020.

[7] A. Eldawyand and M. F. Mokbel, "Spatial Hadoop: A Mapreduce Framework for Spatial Data," Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Republic of Korea, Apr. 3-17 2015, pp. 1352-1363.

[8] J. Lu and R. H. Guting, "Parallel Secondo: A Practical System for Largescale Processing of Moving Objects," Proceedings of the 2014 IEEE 30st International Conference on Data Engineering, Chicago, USA, Mar. 31-Apr. 4 2014, pp. 1190-1193.

[9] A. R. Mahmood, A. M. Aly, T. Qadah, E. K. Rezig, A. Daghistani, A. Madkour, A. S. Abdelhamid, M. S. Hassan, S. B. Walid, and G. Aref, "Tornado: A Distributed Spatio-textual Stream Processing System," Proceedings of the 41st International Conference on Very Large Data Bases, Hawaii, USA, Aug. 31- Sep. 4 2015, pp. 2020-2023.

[10] Apache Spark, http://spark.apache.org/

[11] Apache Hadoop, http://hadoop.apache.org/

[12] Apache Storm, http://storm.apache.org/

[13] H. Lee, Y. Kwak, and S. Song, "Implementation of Distributed In-Memory Moving Objects Management System," Advanced Science Letters, vol. 23, no.10, 2017, pp. 10361-10365.

Hyeonbyeong Lee

He received the BS degree in Computer Engineering Department from Korea National University of Transportation, Republic of Korea in 2014. He is an graduate student of Korea National University of Transportation, Republic of Korea. His research interests are bioinformatics, moving object databases 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.