Parallel Insertion and Indexing Method for Large Amount of Spatiotemporal Data Using Dynamic Multilevel Grid Technique

Sangdeok Park 1, Daesik Ko 2 and Seokil Song 3,*

1 Department of Information Convergence, Korea National University of Transportation, Daehakro 50, Chungju, Chungbuk 27469, Korea; deok31513@a.ut.ac.kr
2 Department of Electronic Engineering, Mokwon University, Doanbukro 88, Daejeon 35349, Korea; kds@mokwon.ac.kr
3 School of Computer Engineering & Information Technology, Korea National University of Transportation, Daehakro 50, Chungju, Chungbuk 27469, Korea
* Correspondence: sisong@ut.ac.kr; Tel.: +82-43-841-5349

Received: 24 July 2019; Accepted: 30 September 2019; Published: 11 October 2019

Abstract: In this paper, we propose a method to ingest big spatiotemporal data using a parallel technique in a cluster environment. The proposed method includes an indexing method for effective retrieval in addition to the parallel ingestion method of spatiotemporal data. In this paper, a dynamic multilevel grid index scheme is proposed to maximize parallelism and to adapt to the skewed spatiotemporal data. Finally, through experiments in a cluster environment, it is shown that the ingestion and query throughput increase as the number of nodes increases.

Keywords: spatiotemporal; index; parallel; ingestion

1. Introduction

Recently, a large amount of spatiotemporal data has been generated, and the applications of spatiotemporal data has been increasing. Consequently, the importance of spatiotemporal data processing also has been increasing. There are many moving objects that generate spatiotemporal data. They are everywhere, such as vehicles on the road, pedestrians on the street, trains on the railroad, ships on the sea, airplanes in the sky, objects in CCTVs, climbers in the mountains, and so on. These moving objects produce very large spatiotemporal data every day. Major areas of spatiotemporal data generation and application are as follows. New York City TLC (Taxi and Limousine Commission) archives more than 1.1 billion trajectories [1]. Twitter has more than 5 million tweets per day, and 80% of mobile users are mobile [1].

Most moving objects transmit their locations periodically to their servers. Recently, various methods have been proposed to deal with the increased importance of very large spatiotemporal data processing. In some studies, parallel and distributed indexing methods to process the location data of moving objects have been proposed [1–10]. According to reference [11], these methods can be divided into two groups depending on what big data processing frameworks, such as Apache Hadoop [12] and Apache Spark [13], are using.

Apache Hadoop is a successful big data processing framework, but it has limited performance improvements due to disk-based data storage and data sharing among MapReduce phases. The significant drop in main-memory cost has initiated a wave of main-memory distributed processing systems. Apache Spark is an open source and general-purpose engine for large-scale data processing systems. It provides primitives for in-memory cluster computing to avoid the IO (Input and Output) bottleneck that occurs when Hadoop MapReduce repeatedly performs computations for jobs.
The big spatiotemporal data processing methods proposed in References [1–4] are based on Apache Hadoop. They were designed and implemented by using HDFS [12], MapReduce [12], or HBase [14] for data storage, indexing, and query processing. References [7–9] proposed Apache Spark-based big spatiotemporal data processing methods. They proposed indexing and query processing methods based on Apache Spark to overcome the limit of performance improvements caused by the disk-based architecture of Apache Hadoop. Besides References [5,6], most of them are aimed at batch loading, querying, and analysis for spatiotemporal data rather than at real-time data ingestion and querying. Reference [5] proposes an indexing method for moving objects based on Apache Spark to manage the index and to 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 memory is full, index structures and location data are processed according to the configuration of Spark. Reference [6] also proposes a distributed in-memory moving objects management system based on Spark that solves the memory full problem of reference [5]. References [5,6] achieved a high data ingestion throughput based on a distributed in-memory framework. However, if the amount of processing is larger than the memory size, disk I/O occurs and there is a limit to performance improvement. In addition, there is a problem in that data ingestion cannot provide parallelism.

GeoMesa [15] is a commercial spatiotemporal data management system to process queries and analytics of big spatiotemporal data based on distributed computing systems. It supports spatiotemporal indexing on top of the various NoSQL (non SQL or non-relational) systems such as Accumulo [16], HBase [14], and so on. Also, GeoMesa provides near real-time stream processing of spatiotemporal data by using Apache Kafka [17].

Since References [1–4] are based on Apache Hadoop, which is a disk-based and IO optimized system, they inherit the performance limitations of Apache Hadoop. Although the methods in References [5–9] are based on Apache Spark, they do not perform well due to its overhead such as scheduling, distributed coordination, and lack of memory for large amount of data. As described above, Geomesa is designed to use various NoSQL systems as its storage engine. Specifically, it uses Apache Accumulo, which uses memory table, cache, and write ahead logs to alleviate IO overhead. It exploits the data distribution feature of Apache Accumulo for indexing spatiotemporal data with space-filling curve techniques. However, to our knowledge, Geomesa may suffer performance degradation since its indexing method is not pipelined and the resolution of space-filling curve is static [18].

In this paper, we propose a real-time parallel ingestion method for big spatiotemporal data by using Apache Accumulo like Geomesa. In addition, a spatiotemporal query processing method that maximizes parallelism of query processing is proposed. The proposed parallel query processing and ingestion methods are based on the table split feature of Apache Accumulo and use in-memory storage. The spatiotemporal data is classified into cells in a dynamic grid index by clients, and the data is parallelly ingested into the partitioned tables (tablets) distributed to the nodes using the threads corresponding to each cell. Our dynamic grid index can be adapted to the data distribution. Query processing is also performed in a similar manner. We identify the cells that need to be accessed during query processing and perform query processing on the tablets corresponding to each cell in parallel.

This paper is organized as follows. Section 2 describes the existing distributed spatiotemporal data processing methods, and Section 3 presents the architecture of the proposed spatiotemporal data processing system and its application for location tracking for mountain hikers. In Section 4, the performance evaluation results of the proposed system are given, and finally, we conclude this paper in Section 5.

2. Related Work

The proposed spatiotemporal data processing method is based on Apache Accumulo’s table partitioning feature. Consequently, before presenting the proposed method in detail, we first describe the structure and features of Apache Accumulo. In addition, we describe existing distributed parallel spatiotemporal data processing methods for comparison with the proposed method.
2.1. Apache Accumulo

Apache Accumulo is a distributed key/value storage system to store and manage large data sets across a cluster. It stores data in table, and a table is divided horizontally into tablets. The master of an Apache Accumulo cluster assigns a group of tablets to a tablet server. Figure 1 shows this process. This allows row-level transactions to be processed without the need for distributed locking or complex synchronization methods. When a client inserts or queries and a node is added or removed from the cluster, the master migrates the tablets so that the ingest or query processing load is distributed across the cluster.

![Figure 1. Data distribution of Apache Accumulo [16].](image1)

As shown in Figure 2, when a write operation is passed to a proper tablet server, it is first written to the WAL (Write Ahead Log) as a log and inserted into memory called a MemTable. When the MemTable reaches a certain size, the tablet server writes the sorted key-value pairs to HDFS as an RFile. This process is called minor compaction. After that, the MemTable is created again and the compaction is written to the WAL.

![Figure 2. Tablet server of Apache Accumulo [16].](image2)

When a tablet server receives a request to read data, the tablet server performs a binary search on the index blocks associated with MemTable and RFile to perform the search. When the client performs
a scan, multiple key-value pairs are returned. If caching is enabled for the table, the index or data block
is stored in the block cache for future scans.

The proposed spatiotemporal method uses the data distribution feature of Apache Accumulo like
Geomesa to improve its parallelism for ingestion and query operations. Each spatiotemporal record is
mapped to one cell of the 3-dimensional space grid according to its GPS location and timestamp. Then,
a number for the cell is assigned by the Hilbert Curve [19] technique. The Hilbert Curve number for
the cell is used to determine which tablet server should take care of the record.

2.2. Distributed and Parallel Spatiotemporal Data Processing Methods

Reference [1] proposes ST-Hadoop. ST-Hadoop is an extension of Apache Hadoop that injects
spatiotemporal recognition in four layers of code bases such as language, indexing, MapReduce, and
the operating layer. A key point that underpins ST-Hadoop’s performance improvement is the idea of
indexing data being loaded and divided over time through compute nodes. Hadoop-GIS [2] extends
Hadoop for handling large spatial data using the MapReduce framework. It separates the data and
store it in HDFS and adds the global index to each tile that is stored in HDFS and shared among the
cluster nodes. Its query engine can index data quickly if needed and is stored in memory for faster
query processing. Its basic indexing method uses Hilbert Tree and R*-tree for global and local data
indexing. Advanced indexing methods support several partitioning and indexing strategies such as
fixed grid, binary partitioning, Hilbert curve, strip, optimized strip, and R-tree. Optimal strategies can
be selected during spatial data processing.

Spatial Hadoop [3] consists of multiple layers of Hadoop such as storage, MapReduce, operational,
and language layers. At the storage layer, it has added a two-level index structure (global and local
indexes). The global index is created for each data partition in the cluster, and the local index constitutes
the data within each node. Consequently, while processing a query operation, it can take advantage of
information about which nodes are mapped to which nodes and which blocks of that node are relevant.
This can speed up query processing.

Parallel SECONDO [4] is a parallel and distributed version of SECONDO [20] 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 spatiotemporal 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 spatiotemporal queries and analyses on the large amount of moving object data and sets of
trajectory data in the cloud. Like Hadoop GIS [2], Parallel SECONDO uses HDFS as the communication
way between data and tasks.

GeoSpark [7] is an in-memory cluster computing framework for processing large spatial data. It
extends Apache Spark to support spatial data types and operations. It uses Quad-Tree, R-Tree, Voronoi
diagrams, and Fixed-Grid to efficiently partition spatial data among cluster nodes. Quad tree and
R-tree indexing techniques are used to index the data on each node. SpatialSpark [8] implements
several spatial operations on Apache Spark to analyze large-scale spatial data. A broadcast join is used
to join a large data set to a small data set and supports two spatial join operations where partition joins
are used to join two large data sets. Spatial data can be segmented using FixedGrid, BinarySplit, and
SortTile partitioning techniques and indexing using R-trees.

LocationSpark [9] is an efficient spatial data processing system based on Apache Spark. Its query
scheduler includes an efficient cost model and a query execution plan that can mitigate and handle data
partition and query skew. Global indexes (grid and local quadrrees) partition spatial data between the
various nodes and local indexes (R-tree, Quadtree transform, or IRTree) that are used to index the data
on each node. LocationSpark also uses a spatial bloom filter to reduce the cost of communication for
global spatial indexes, which can determine whether a spatial point is within a spatial extent. Finally,
to efficiently manage main memory, frequently accessed data is dynamically cached in memory and less frequently used data is stored on disk, greatly reducing the number of IO operations.

Reference [5] proposes an in-memory distributed indexing method for moving objects based on Apache Spark. The basic technique of Reference [5] is a simple grid index. Reference [5] 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.

Reference [6] proposes distributed an in-memory moving object management system based on Spark. It consists of a data and query collector, an index manager, and a data manager. Data and query collectors which are designed based on Apache Kafka receives location data and time from vehicles and queries from users. Index manager creates grid-based spatiotemporal index structures, and it is an enhanced version of that in Reference [5], 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 RDD(Resilient Distributed Dataset) of Apache Spark. Data manager is to store old index structures to HBase and to load index structures.

GeoMesa [12] provides spatiotemporal indexing using space-filling curves to transform multidimensional spatiotemporal data into the one-dimensional data. It is designed to run based on distributed storage systems such as HDFS, Apache Accumulo, and so on. GeoMesa creates indices on the geospatial attributes (point and spatial) of spatiotemporal data. These indices are implemented by creating a space-filling curve based on a GeoHash index. GeoMesa uses Zcurve and XZ space-filling curve, respectively, for point data and spatial data.

3. Parallel Insertion and Indexing Method for Proposed Spatiotemporal Data

Figure 3 shows the overall architecture of the proposed parallel ingestion and indexing method of big spatiotemporal data stream based on Apache Accumulo [16]. As shown in the figure, the spatiotemporal data generated by moving objects are transmitted periodically to the ingest manager in the form of a data stream through Apache Kafka. Ingest manager stores the transmitted spatiotemporal data in a data buffer of fixed size. The spatiotemporal data in the data buffer is distributed to tablet servers of Apache Accumulo to be stored in a data table. Before storing the data to the data table, indexing process is performed.

![Figure 3. Architecture of the proposed parallel spatiotemporal data ingestion method.](image-url)

The index data created from the spatiotemporal data in the data buffer is stored in an index buffer of fixed size, which may be greater than the size of the data buffer. The index buffer is flushed whenever the buffer is full. This process is performed by an index manager. We use Hilbert Curve technique for mapping the spatiotemporal properties of data to one-dimensional data and the Grid
A technique to distribute spatiotemporal data to tablet servers. Spatiotemporal data and index data are inserted in parallel into the tablet servers in charge of the mapped value to which the respective data belong. The indexing and insertion of the data are implemented as Kafka’s consumer, and the number of Consumers is equal to that of table servers. Each consumer can simultaneously insert the data into each tablet server to maximize parallelism.

Figure 4 shows the schema of data table and index table and the overall data ingestion process of the proposed method in this paper. As shown in the figure, the key of the data table is a combination of the ID (the moving object ID) and the timestamp of a spatiotemporal record. The key of the index table is a combination of cellID (Hilbert Curve value) and the timestamp of the record. Our indexing procedure begins with moving objects. All the moving objects have indexing information such as grid size and time interval for Hilbert Curve mapping. As shown in the figure, a moving object maps timestamp and location of a record to a Hilbert Curve value (cellID) and then transmits the record with the cellID to the ingest manager.

| Data Table | Key | Column Family | Column Qualifier | Value |
|------------|-----|---------------|------------------|-------|
| ID + timestamp | “Data” | “Data” | Record |

| Index Table | Key | Column Family | Column Qualifier | Value |
|-------------|-----|---------------|------------------|-------|
| cellID | “Data” | ID or ObjectID | Data Table Key |

![Data Ingestion Process Diagram](attachment:image)

**Figure 4.** Data ingestion process and schemas for the index and data table.

Ingest manager stores the input spatiotemporal data stream from moving objects in the data buffer of fixed size. Concurrently, the index manager creates index records with the records in a data buffer. The data buffer consists of a hash table. An index record consists of a key (cellID and timestamp) and a value (its key of the data table). The index records are stored in an index buffer which has a KD-tree [21] structure. Spatiotemporal data and index data in both buffers are flushed into Apache Accumulo. Apache Accumulo has multiple tablet servers, and cellIDs (Hilbert Curve values) are assigned to tablet servers. Thus, the flush operations for both buffers are performed in parallel by the tablet servers.

As described earlier, Apache Accumulo enables to split data in advance and to assign key ranges to tablet servers. We use this feature to assign cellIDs (Hilbert Curve values) to tablet servers. Figure 5 shows an example of the proposed method. Index manager creates a grid for a given area on a time interval. In this figure, time interval is 10, i.e., the first TI (time interval) is T0–T9 and the second TI is T10–T19. The Hilbert Curve value of the grid for TI, where i (0–k) means the order of TI starts at \([i \times \text{row size} \times \text{column size}]\), where i means TI and \(\text{row size}\) and \(\text{column size}\) mean the row size and the column size of the grid, respectively. In this figure, \(\text{TI}_0\) starts at 0 and \(\text{TI}_1\) starts at 16 when the grid size is \(4 \times 4\). Then, a mapped Hilbert Curve value (cellID) is assigned to a tablet server, for example, cellIDs 0–3 and...
16–19 are assigned to tablet server1. The assignment depends on the number of servers and the size of the grid.

Generall y, locations of moving objects can be skewed to a specific area and the area may be changed with time. The indexing method described above cannot process efficiently the skewed location data. Therefore, we propose a dynamic grid technique that can be adapted to the skewed location data. Figure 6 shows the proposed dynamic grid indexing method. In our method, multilevel grid technique is used. Initially, multilevel grid starts with only level 1. Then, when the number of records contained in a grid cell exceeds a given threshold value, we create lower level grids for the cell. As shown in Figure 6, grid cell 7 in level 1 exceeds a threshold and then level 2 is created for the cell. CellID is calculated by Equation (1). In the figure, $HCV_i$ means the Hilbert Curve value of $i$th level.

\[
\text{cellID} = HCV_1 + \sum_{i=2}^{l} (HCV_i + 1) \times 10^{i-1}
\]

Figure 6. Dynamic grid indexing for skewed data.

In Figure 7, there is an example of the proposed indexing method. The threshold value for the number of data records for a cell is 3 in that example. O11, O31, and O41 are inserted sequentially into the area for the grid cell 7. According to the Equation (1), cellIDs of the newly created grid are 7.1, 7.2, 7.3, and 7.4. Then, level 2 grid for the grid cell is created, and after that, O12, O13, O42, and O32
are inserted. In this example, the grid cell 7.4 exceeds the threshold, so the level 3 grid is created for the cell.

![Diagram of grid indexing](image)

**Table 1.** CellIDs for data objects.

| CellID | Data Records |
|--------|--------------|
| 7      | O_{11}, O_{31}, O_{41} |
| 7.4    | O_{12}, O_{13} |
| 7.3    | O_{32} |
| 7.2    | O_{42} |
| 7.44   | O_{21} |
| 7.42   | O_{23} |

**Figure 7.** Example of dynamic grid indexing.

In the above example, cellIDs are assigned to the data records like Table 1. As shown in the table, data records inserted before a new level of grid is created have cellIDs assigned to it. Consequently, all levels of cellIDs must be considered to process a range query. In Figure 8, we show an example to process range queries. Figure 8a shows an example of range query processing on one level grid. Range queries Q1 and Q2 overlap the grid cell 7 so to process the queries we need to compare all data records in the cell. In Figure 8b, range queries are processed on dynamic multilevel grid indexing method. In this example, to process Q1 and Q2, retrieve 4 records and 6 records only, respectively.

**Figure 8.** Query processing of dynamic grid indexing method.
4. Performance Evaluation

In this paper, we compare the proposed method with Geomesa in terms of ingestion and range query throughput through experiments. Geomesa is one of the well-known big spatiotemporal data management systems. It is currently maintained and professionally supported by CCRi. The most recent version of Geomesa is 2.3.1 released in July 2019. We use the Geomesa 2.3.1 version in our experiments for the comparison. Table 2 shows the experimental environment of this paper. Nine nodes are used for Geomesa and the proposed method, and 8 nodes are used for clients that request queries and data insertion. Client HW(Hardware) specifications are higher than server HW specifications. The reason is to run multiple client processes on each client node to provide enough workload for Geomesa and the proposed method.

Table 2. Experimental environments.

| Server HW (9 Nodes) | Client HW (8 Nodes) | SW(Software) | Data Set 1 | Data Set 2 | Query Set 1 | Query Set 2 |
|---------------------|---------------------|--------------|------------|------------|-------------|-------------|
| Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30 GHz, 4 cores, HDD 50 GB, RAM 16 GB | Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40 GHz, 16 cores, HDD 20 GB, RAM 64 GB | Ubuntu 16.04 LTS, Apache Kafka 2.11-2.2.0, Apache Hadoop 2.8.4, Apache Accumulo 1.9.1, Apache Zookeeper 3.4.10 | Synthetic spatiotemporal data 100,000,000 with uniform distribution generated from the GPS area (37.2125, 128.1361111–36.79444444, 127.6611111) | Synthetic spatiotemporal data 100,000,000 with hot spot generated from the GPS area (37.2125, 128.1361111–36.79444444, 127.6611111) | Range queries 5,000,000 with uniform distribution | Randomly generated range queries 5,000,000 |

We generate a couple of synthetic spatiotemporal data sets from the GPS coordinate area (37.2125, 128.1361111–36.79444444, 127.6611111), as shown in Figure 9. The first dataset is 100,000,000 spatiotemporal data with a uniform distribution. The second dataset is a 100,000,000 spatiotemporal dataset with a hot spot where 80% of the total data places 20% of the area. We also generate two query sets. Like the data set, the first query set has a uniform distribution of query ranges and the second query set has the same hot spot as that of the second data set. The average number of returned objects of the range queries is about 120. To compare the performance of the proposed method and Geomesa, we measure ingestion and query throughputs with varying the number of nodes.

Figure 9. GPS coordinate area used for data generation.
4.1. Experiments with Uniform Distribution Data Set (Data Set 1)

In our first experiments, we execute 40 client processes in 8 client nodes that send 100,000,000 (uniform distribution) insertion workloads to Geomesa and our proposed spatiotemporal data management system with varying the number of server nodes from 3 to 9. While performing experiments, we measure the number of completed insertion operations in each server node and the total execution time. Figure 10 shows the experimental results, i.e., ingestion throughput of Geomesa and the proposed method as nodes increase. As shown in the figure, the ingestion throughput of our proposed method scales up well as nodes increase while that of Geomesa does not increase well when the number of nodes is greater than 6. Also, the throughput of the proposed method is about 4.5 times higher than that of Geomesa.

![Figure 10. Ingestion throughput of Geomesa and the proposed method (Ingestion throughput: number of insertion operations per second).](image)

In our second experiments, we also execute 40 client processes in 8 client nodes that send 5,000,000 range query (uniform distribution) workloads to both systems with varying number of server nodes from 3 to 9. While performing experiments, we measure the number of completed range queries and their results in each server node and the total execution time. The results of range queries of both systems are used to compare the accuracy of range queries. In Figure 11, the experimental results are shown. As shown in the figure, the throughput difference between the proposed method and Geomesa is small. When the number of nodes is 6, the range query throughput of both methods are almost the same, and when the number of nodes is 3 and 9, the throughput of the proposed method is about 1.3 times higher. In terms of scalability, the range query throughput of both systems scale up well as node increases.

![Figure 11. Range query throughput of Geomesa and the proposed method (Range query throughput: number of range queries per second).](image)
4.2. Experiments with Hot Spot Data Set (Data Set 2)

We also perform experiments with the hot spot data set (Data Set 2 in Table 1) and the hot spot query set (Query Set 2 in Table 1). As described earlier, the second data set has hot spots. The experimental process is the same to that of the experiments using Data Set 1. Figure 12 shows the experimental results, i.e., ingestion throughput of Geomesa and the proposed method as nodes increase. As shown in the figure, the ingestion throughput of our proposed method scales up well as nodes increase while that of Geomesa does not increase well when the number of nodes is greater than 6. Also, the throughput of the proposed method is about 4.8 times higher than that of Geomesa.

![Figure 12. Ingestion throughput of Geomesa and the proposed method (Ingestion throughput: number of insertion operations per second).](image)

In Figure 13, the experimental results are shown. As shown in the figure, the range query throughput of the proposed method scales well while that of Geomesa does not. The throughput of the proposed method is about 1.7 times higher than that of Geomesa. Specifically, when the number of nodes is 9, the throughput of the proposed method is about 2.2 times higher.

![Figure 13. Range query throughput of Geomesa and the proposed method (Range query throughput: number of range queries per second).](image)
4.3. Analysis of Experimental Results

GeoMesa may suffer a performance degradation during data ingestion because its indexing method is not pipelined. However, our proposed method inserts asynchronously data records and index records. Our proposed method uses lazy insertion policy for index records and always ensures the data records are inserted ahead their index records. If index records are lost due to some failures, since data records are stored, the lost index records can be recovered.

Also, the proposed dynamic grid indexing method can partition spatiotemporal data evenly across each node to increase the parallelism. Consequently, it can increase the ingestion throughput and range query throughput. Figure 14 shows the performance improvement rates of the range queries and insert operations of the proposed method compared to Geomesa. As shown in figure, the performance improvement rates are higher in the experiments with hot spot data and queries.

![Figure 14. Performance improvement rates of range queries and ingestion of the proposed method compared to Geomesa.](image)

5. Conclusions

In this paper, we proposed a method to parallel ingest and query method for big spatiotemporal data in a cluster computing environment. The proposed method includes a dynamic multilevel grid index scheme to process queries efficiently for the skewed spatiotemporal data. Through experiments, we showed the proposed method has high scalability in throughput in data ingestion and range query processing through experiments. In our future work, we will perform experiments with real spatiotemporal data sets and compare with other recent spatiotemporal data management systems.

Author Contributions: Conceptualization, methodology, formal analysis, investigation, data curation, writing—original draft preparation, S.P. and S.S.; conceptualization, methodology, formal analysis, writing—review, editing and supervision S.S.; writing—review, project administration, funding acquisition, D.K.

Funding: This work was supported by the ICT R&D program of MSIT/IITP (B0101-15-0266, Development of High Performance Visual BigData Discovery Platform for Large-Scale Realtime Data Analysis). Also, this work was carried out with the support of R&D Program for Forest Science Technology (Project No. 2017063B10-1719-AB01) provided by Korea Forest Service (Korea Forestry Promotion Institute).

Conflicts of Interest: The authors declare no conflict of interest.
References

1. Alarabi, L. ST-Hadoop: A Mapreduce Framework for Big Spatio-temporal Data. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017.

2. Aji, A.; Wang, F.; Vo, H.; Lee, R.; Liu, Q.; Zhang, X.; Saltz, J.H. Hadoop-GIS: A High Performance Spatial Data Warehousing System over Mapreduce. In Proceedings of the 41st International Conference on Very Large Data Bases, Kohala Coast, HI, USA, 31 August–4 September 2015.

3. Eldawyand, A.; Mokbel, M.F. Spatial Hadoop: A Mapreduce Framework for Spatial Data. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Korea, 13–17 April 2015.

4. Lu, J.; Guting, R.H. Parallel Secondo: A Practical System for Large-scale Processing of Moving Objects. In Proceedings of the 2014 IEEE 30th International Conference on Data Engineering, Chicago, IL, USA, 31 March–4 April 2014.

5. Lee, Y.; Song, S. Distributed Indexing Methods for Moving Objects based on Spark Stream. Int. J. Contents 2015, 11, 69–72. [CrossRef]

6. Lee, H.; Kwak, Y.; Song, S. Implementation of Distributed In-Memory Moving Objects Management System. Adv. Sci. Lett. 2017, 23, 10361–10365. [CrossRef]

7. Jia, Y.; Jinxuan, W.; Mohamed, S. Geo Spark: A Cluster Computing Framework for Processing Large-scale Spatial Data. In Proceedings of the 23rd SIGSPATIAL GIS, Bellevue, WA, USA, 3–6 November 2015.

8. You, S.; Zhang, J.; Gruenwald, L. Large-Scale Spatial Join Query Processing in Cloud. In Proceedings of the 2015 IEEE 31st International Conference on Data Engineering, Seoul, Korea, 13–17 April 2015.

9. Mingjie, T.; Yongyang, Y.; Qutaibah, M.; Mourad, O.; Aref, W.G. Location Spark: A Distributed In-memory Data Management System for Big Spatial Data. Proc. VLDB Endow. 2016, 9, 1565–1568.

10. Hulbert, K.; Hughes, F.; Eichelberger, C.N. An Experimental Study of Big Spatial Data Systems. In Proceedings of the 2014 IEEE 30th International Conference on Data Engineering, Seoul, Korea, 31 March–4 April 2014.

11. Lawder, J.K.; King, P.J.H. Querying Multi-dimensional Data Indexed using the Hilbert Space-filling Curve. ACM Sigmod Record 2001, 30, 19–24. [CrossRef]

12. Bentley, J.L. Multidimensional Binary Search Trees used For Associative Searching. Commun. ACM 1975, 18, 509–517. [CrossRef]

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).