Query Processing of Massive Trajectory Data based on MapReduce

Qiang Ma, Bin Yang
School of Computer Science
Fudan University
Shanghai, China
{maqiang, byang}@fudan.edu.cn

Weining Qian, Aoying Zhou
Institute of Massive Computing
East China Normal University
Shanghai, China
{wnqian, ayzhou}@sei.ecnu.edu.cn

ABSTRACT

With the development of positioning technologies and the boosting deployment of inexpensive location-aware sensors, large volumes of trajectory data have emerged. However, efficient and scalable query processing over trajectory data remains a big challenge. We explore a new approach to this target in this paper, presenting a new framework for query processing over trajectory data based on MapReduce. Traditional trajectory data partitioning, indexing, and query processing technologies are extended so that they may fully utilize the highly parallel processing power of large-scale clusters. We also show that the append-only scheme of MapReduce storage model can be a nice base for handling updates of moving objects. Preliminary experiments show that this framework scales well in terms of the size of trajectory data set. It is also discussed the limitation of traditional trajectory data processing techniques and our future research directions.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Design

Keywords

MapReduce, Spatio-temporal Queries, Location Based Services, Cloud Computing, Moving Objects, Indexing

1. INTRODUCTION

With the development of positioning technologies and the boosting deployment of inexpensive location-aware sensors, location based services are playing more and more important roles in both military issues and common daily life. From the outdoor location reported by GPS devices in longitude-latitude coordinate systems, to the indoor location observed by sensors in symbolic coordinate systems, diverse formats of trajectory data have emerged as well as spatio-temporal trajectory data sets in large volumes have been accumulated. For example, using radar to track an airplane is a typical application in military. In daily life, modern cars are always equipped with GPS positioning devices. Drivers are indicated for a considerable route in terms of its current location. The recorded historical trajectories, i.e. a set of routing sequence are also essential to the traffic management. Logistics companies are capable of making schedule plans for their fleets based on the historical trajectories.

A historical trajectory data set is usually in large volume and it exceeds the computation capacity of traditional centralized technologies. Nowadays, data intensive computing technology has been broadly applied in many applications. In these applications, the input data set, i.e. historical trajectory data is huge. Hence, processing the entire data set requires it to be divided into many subdivisions and dispersed across thousands of machines and give the result in acceptable time.

Facing trajectory data set in huge volumes and complicated formats, cloud computing will provide a promising paradigm to conquer the explosion of data. There are many research efforts on large-scale distributed storage and computing, such as Cooperative File System [6], GFS [8], MapReduce [7], Map-reduce-merge [5], Dryad [10], Pig latin [13]. UTab [14]. The efficiency of cloud computing has been shown by the performance of many data intensive applications. Recently, the frameworks of GFS and MapReduce have gained many attentions around the world. Although it is often used in massive data analysis and presents good performance, we argue that it provides a solid base for parallel trajectory data processing. MapReduce provides a framework that easily enables to process massive amount of data over large clusters. However, this framework does not directly support sequential data, which consists of ordered data elements. Trajectory data of moving objects, whose time dimension is monotonously increasing and whose latter location is dependent on the former one, is one of the most widely used sequential data in the world.

1.1 Challenges

Efficient query evaluation and data analysis on such massive trajectory data set remain a big challenge. There are some issues on the management of trajectory data over large clusters.

CloudDB'09, November 2, 2009, Hong Kong, China.

1. The volume of historical trajectory data set too huge to maintain, moreover it updates frequently since all objects' trajectories and current location must be traced, thus it leads to rapidly increasing volume of data. Simultaneously, frequent update costs too much, and it is inefficient that cluster writes disk for each updated information.

2. Trajectory data is 'continuous' and the ordered sequentiality must be reserved. The split method, often used in MapReduce, uniformly cuts the data set into several parts and puts them into different nodes, doesn't work here. As a typical kind of sequential data, trajectory data of moving objects has some important distinct characters. Specifically, trajectory data is continuously changing between any two successive updates of the location of the mobile objects. If the whole data set is divided into several parts, the continuity of some trajectories may be broken.

3. The trajectory data may be highly skewed. For example, more vehicles travel in the downtown than those in suburb. That is a catastrophe for distributed system and may lead to load imbalance. Another important fact is that the number of moving objects in different regions will change as time goes by. So traditional spatial static partition strategy is not suitable for ununiform distribution.

4. Traditional MapReduce framework is good at offline data analysis, but not efficient for online query. Usually, there are a large number of concurrent queries over large clusters. Technologies, which can speed up the query processing especially optimizing batch processing in cluster, are urgently needed.

1.2 Our Contributions

As stated before, the whole historical trajectory data set usually cannot be fit into a single machine. We propose our system called PRADASE(Query **P**rocessing of Massive T**RA**jectory **DA**ta ba**SE**d on MapReduce), which relies on a GFS-style storage which only supports appending data. The master node is only responsible for key to (data) nodes mapping and lookup (via hashing). So the corresponding data can be returned automatically if the key is given. The execution engine is built on Map-reduce model which can greatly simplifies many data intensive computational tasks. It is suitable for large-scale distributed systems such as GFS and Hadoop, since interactions among data on different nodes happens only in the automatic grouping, which is scheduled, and optimized.

The process of execution is: (1)Splitting input data into several pieces; (2)executing sub-task in each piece of data; (3)grouping all intermediate values with the same key; (4)reduction of each group. However, whether management of trajectory data is suitable for this framework still need further researches. In this paper we study the problems of implementing trajectory processing technologies over MapReduce-liked framework. Several sub-problems of trajectory processing in large clusters can be summarized as follows:

1. We extend MapReduce framework to implement massive sequential data processing like trajectory of moving objects. Furthermore, the requirements to process trajectory over MapReduce-liked framework are discussed in this paper.

2. Most traditional trajectory query processing methods designed for centralized environment are not directly applicable in MapReduce-liked framework. We study what kinds of method are appropriate for large clusters, and parts of them are improved to become a new partition strategy which can save the continuity of trajectories and support distributed environment storage such as GFS-style storage.

3. Moving objects are frequently updated, whereas it is inefficient that system update each location for every object. We propose that all new updated data are maintained in main memory at each data node until particular size of data is accumulated. Writing data to disk in term of batch can largely improve the performance of data update.

4. In order to optimize query processing over large clusters, we present two scalable indexing methods which can facilitate query processing efficiently and decouples spatial and temporal dimension. The first one is PMI(Partition based Multilevel Index) which is proposed to deal with spatio-temporal range queries. The second one is OII(Object Inverted Index) which can facilitates trajectory based queries.

The rest of this paper is organized as follows. Section 2 presents the assumed trajectory data model and two categories of queries. Section 3 gives the detail about trajectory processing in large cluster, in which framework, storage, indexing methods, update and query processing will be elaborated separately. Section 4 reports an comprehensive experimental study. Section 5 briefly reviews the related work. Section 6 concludes the paper and discusses directions of future research.

2. PRELIMINARY

2.1 Data Model

The popular line segments model [15] for representing trajectories is adopted in this paper. A trajectory of a moving object is a polyline in three-dimensional space, where two dimensions refer to space and the third dimension refers to time. It is represented as a sequence of position points $Tr(P_1, P_2, \ldots, P_n)$; where each position point P_i is in form of (x, y, t), where x and y represent the coordinates of the object in a given spatial coordinate system, and t indicates the corresponding timestamp of the report of the position. Each line segment in a trajectory is indicated as $S_i(P_i, P_{i+1})$, where S_i is the line segment identifier, P_i and P_{i+1} indicate two consecutive reported positions.

2.2 Query Types

The queries on historical trajectories can be generally classified into two different classes:
Spatio-temporal Range Query: Given a spatial extent E_s and a temporal extent E_t, the query $Q(E_s, E_t)$ returns all those trajectory line segments which are intersected with the spatio-temporal region defined by E_s and E_t.

$$Q(E_s, E_t) \rightarrow \{S_k\}$$

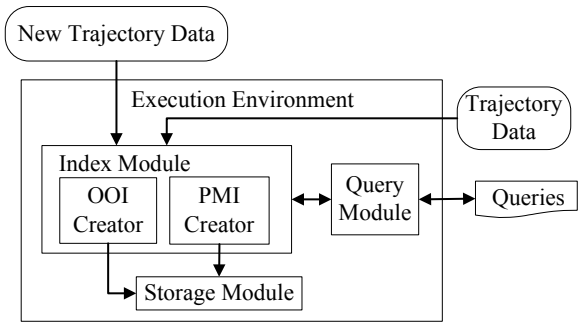

Figure 1: Execution Overview and Framwork

Trajectory-based Query: Given a object identifier o and a temporal extent E_s, Q(o, E_t) returns all the trajectory line segments of the given object o in the time period E_t.

$$Q(o,\ E_t) \rightarrow \{S_k\}$$

3. TRAJECTORIES PROCESSING

In this section, we will present a detailed analysis on our technologies of trajectories data processing in this framework.

3.1 Execution Overview

The framework of querying massive historical trajectory data is composed of index module and query module, which is shown in Fig. 1. When the trajectory data is imported into the system, index module will use PMI creator and OII creator to build two types of indices accordingly. After building the indices, all data is transferred to storage module. When queries are submitted, the results will be returned quickly by invoking index module.

1. Storage module: Historical trajectory data is always in a large amount and stay static. A single machine may not provide enough storage and processing capacity for the large amount of historical trajectory data. Thus, a distributed cluster or a cloud computing infrastructure is preferred to be employed to store and process the large amount of historical trajectories.

2. Index module: In the index module, historical trajectory data is divided into different partitions. All data in one partition is maintained in one chunk, and simultaneously stored at more than one data node as replications. Two different indices are employed in the indexing module in order to optimize the range query and trajectory based query.

3. Query module: In the query module, two kinds of queries are efficiently processed. Receiving a spatio-temporal range query, the spatial extent of the range is checked against the partition of PMI. For trajectory-based query, given an object identifier, the corresponding address of storage can be directly located.

3.2 Storage

PRADASE is built on GFS-style storage with one-master and multiple-data-server. Data are grouped with key and organized in chunks. Each chunk has duplications in multiple nodes, so that single-point failure would not affect the availability of the system. Failure of master node can be recovered from data nodes' information. PRADASE is consisted of many nodes. The whole data set is divided into several parts, and each part is called a partition and assigned to one chunk to store. A good spatial partitioning is the one in which the size of data per chunk is fairly uniform, which can make the efficiency of the whole cluster maximize, if not the load imbalance may happen. In this section, we will introduce how to store huge trajectory data on different chunk.

Recently, some research works are proposed on spatio-temporal space partitioning, while most of them are static or dynamic strategies. In conclusion, static partitioning strategies are easy to control and suitable for distributed scheduling. The static uniform partitioning method which is adopted in SETI [3], is one of the most popular static partitioning strategies and suitable for uniform distributed moving objects. However, in the real world the number of moving objects in each partition is always changing and trajectory data is always highly skewed, which may lead to load imbalance. Dynamic strategies can resolve this problem. In order to ensure the size of each partition is fairly uniform, dense region should be divided into more partitions. However, when the space is repartitioned, the data set will be resplitted. The cost is too complicated even can cause distantly migration of large volume of data, which is a disaster for data management in large cluster. In summary, single static or dynamic strategy is not suitable for the data management of massive moving objects in large clusters. So a hybrid method is proposed in this paper.

The number of moving objects will change with the increasing time and leads to data distribution change accordingly. In order to ensure better load balance on highly skewed trajectory data, we propose a hybrid method which uses individual static partition strategy for each time period. In PRADASE, a chunk monitor is designed to record the size of all chunks. If the system finds the size of data for each chunk is imbalanced, which means partition strategy is not suitable, thus a new partition strategy will be trained. Historical data which has already been imported into the system keeps unchanged, whereas the new updated data will follow the new partition principle. As a result, the partitioning is dynamic from the point of whole time dimension, but is static within each time period. Fig. 2 shows an example of this process, and the performance of load balance is supported by experiments introduced in section. 4.2.

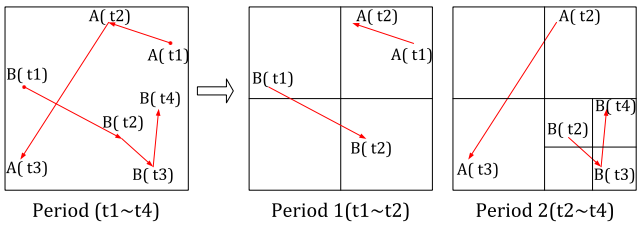

Figure 2: Individual static partition strategy for each time period

After partitioning the temporal or spatio-temporal space, all trajectory data is assigned to at least one partition ac-

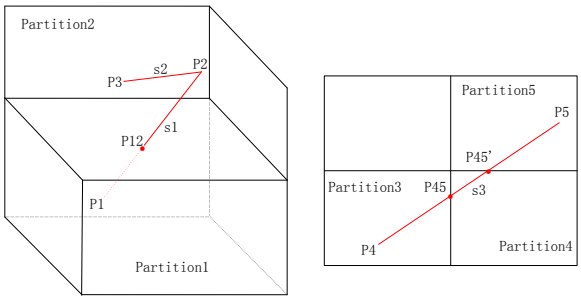

Figure 3: Examples of Splitting Segment

cording to spatio-temporal information. It can be summarized as follows, if a line segment is fully covered by one partition, the line segment is assigned to this partition. If a line segment spans more than one partitions, such a line segment is divided into several sub line segments. Each sub line segment is fully covered by one partition, and is assigned to this partition. The new end points of such sub line segments are the intersections with the original line segment and the partition boundaries. The corresponding time for each end point is determined by interpolating.

Fig. 3 shows two examples of this process. The spatio-temporal division is illustrated in the first one. Line segment $s_2(P_2, P_3)$ is transformed into (Partition2, $s_2(P_2, P_3)$). Line segment $s_1(P_1, P_2)$ is divided into two sub line segments and mapped to (Partition1, (P_1, P_{12})) and (Partition2, (P_{12}, P_2)). The spatial division is illustrated in the second one. Line segment $s_3(P_4, P_5)$ is divided into three sub line segments and mapped to (Partition3, (P_4, P_{45})), (Partition4, (P_{45}, P'_{45})) and (Partition5, (P'_{45}, P_5)).

3.3 Indexing Methods

A space division method which splits the whole trajectory data set to several parts is introduced in section. 3.2. All data in each part is stored in one chunk. Further more, the partition can also speeds up range queries by reducing the scale of all spatial space into a candidate partition list, which is called PMI(Partition based Multilevel Index) and will be introduced in section. 3.3.1.

For moving objects, efficient and scalable indices which can facilitate both spatial-temporal range queries and trajectory based queries should be designed. So, we design another index called (OII)Object Inverted Index by collecting each object's all trajectory data to store them together in order to optimize trajectory based query. A hybrid index combines two indexing methods: partition based Multilevel Index (PMI) for spatio-temporal range queries, and Object Inverted Index(OII) for trajectory based queries is adopted. In this chapter, we will introduce the two indexing methods respectively, and the synergy of them will be elaborated in Figure. 3.3.3

3.3.1 Partition based Multilevel Index

As is introduced in section 3.2, the whole space can be partitioned into several logical nonoverlapping partitions by space partitioning. If a trajectory segment crosses a spatial partitioning boundary then that segment is splitted at the boundary, and is inserted into both partitions. Each partition can be viewed as a bucket which contains almost the same volume of trajectory data. At the same time, each partition contains only trajectory segments that are completely within the partition and all data which covers this partition is maintained in one chunk. With the restriction that each data chunk only contains trajectory segments that belong to the same partition, we can take advantage of this character to build a hash index to speed up range query. Given any spatial-temporal range, we can generate all candidate partitions where the query range covers by invoking space partition strategy. The query results must be contained in these candidate partitions, then the system directly locates the address where data chunk store by hashing the partition identifier. The system scans only a few candidate chunks instead of all data chunks.

For further extensions, the system can build a multilevel index for each chunk in local node where this chunk store. Some traditional centralized methods, such as creating R-tree for each partition [3], can be adopted in each data node. However, it may increase the cost of maintenance in some cases. This is beyond the scope of key content and not introduced in this paper, and we only use the first level index in our experiments.

3.3.2 Object Inverted Index

The system can access data quickly based on PMI when a range is provided, however if some objects need to be traced, every partition in the storage system have to be examined. In order to speed up trajectory based query of some moving objects, we designed the index for each moving object which is called object inverted index. We collect each object's all historical trajectories and store these data together. We use a triple<OID, P_l, T> to present the data model of OII, where OID is the identifier of the object, P_l is the current location of the object including spatial location and corresponding time stamp, T is the historical trajectory of this object represented as line segments model. So given any object identifier, the last known location and all corresponding trajectories will be returned quickly.

The distribution of objects identifier is discretely distributed and different objects are not constant, so all objects' historical data can be uniformly assigned to different nodes by hashing object identifier. We design a two-level hash index for object oriented queries. By using the first level the data node where the data chunk stores at is returned, and then trajectory data can be addressed by invoking the second index. It is worth noting that similar method is adopted in bigtable [4] and not introduced in this paper.

3.3.3 Data Insertion

We provide a MapReduce-liked model to facilitate massive trajectory data set insertion.

As can be seen from Figure. 4, first of all, the appropriate space partition strategy should be trained. When the data insertion starts, two kinds of data processing, PMI and OII are invoked. In this framework, there are two kinds of MapReduce subprocess.One is the process of PMI creation, the other is the process of OII creation.

For PMI creation, in map phase each data node reads pieces of historical trajectory data set, and transforms each segment $s_k(P_i, P_j)$ to a list <partitionID, $s_k(P_i, P_j)$> by invoking trajectory split strategy introduced in previous section, then output as pairs of <key, value>. The intermediate output with the same partition identifier will be grouped and collected together to obtain <partitionID, $\{s_l(P_m, P_n)\}$ >

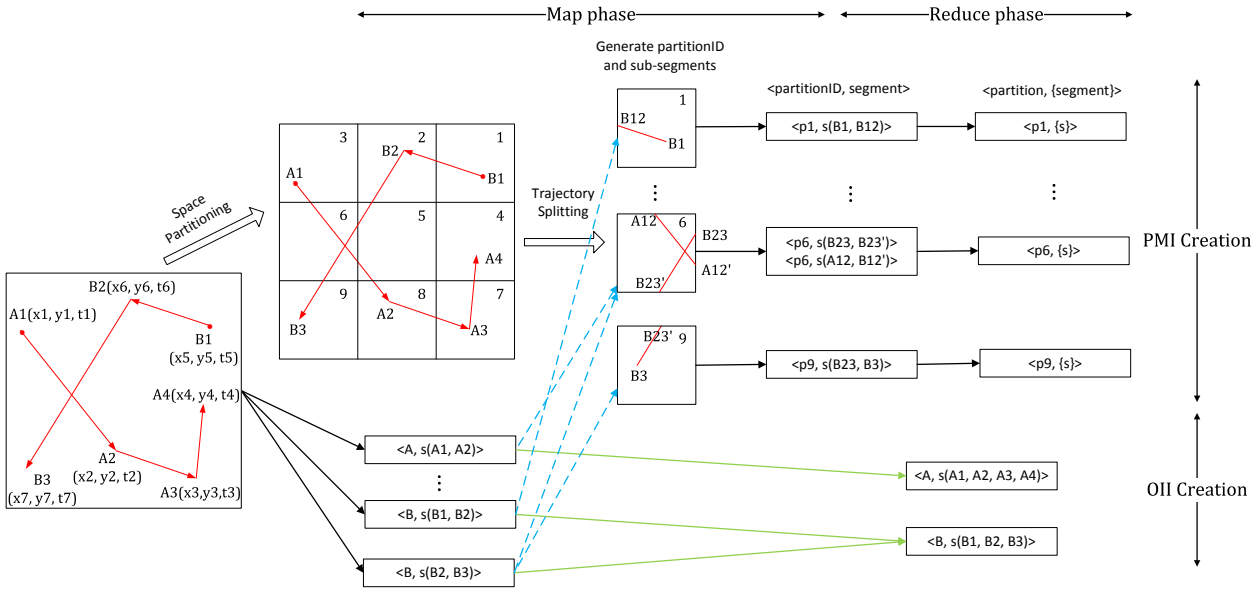

Figure 4: PMI and OII based Trajectory Data Insertion

in one reduce node. Then in reduce phase multilevel index can be created at each group and stored at local. At the same time of creating PMI, OII is created. In map phase, each segment $s_k(P_i, P_j)$ is transformed to <objectID, $s_k(P_i, P_j>$ and output as a pair. After grouping all intermediate output, all trajectory segments with the same object identifier are collected to obtain <objectID, {T}> in reduce phase, where T represent one object's all historical trajectories.

For initial insertion which is the first time to import data, historical trajectory data has accumulated for a long time so that volume is very huge. In this case, the utilization of all nodes in the cluster is very high, and the whole cluster is highly efficient. However for new reported locations which need to be updated, writing disk for each new reported location of every object is too complicated. In order to improve the efficiency of updating, each reduce node can maintain a buffer to collect new data, then write to disk until certain size of new data are accumulated.

For any new updated trajectory of moving objects, if this object has existed in OII, the system needs to read last known location P_l' at first, then update its' historical trajectory T and last known location P_l. If this object has not existed in OII, that means it is the first time to import this object into the system. Thus we need to create a new object identifier and add corresponding data to historical trajectories T and last known location P_l. Finally the corresponding PMI is updated. To be noted, for different time periods, if the partition strategy changes, the new chunks will be created.

3.4 Query Processing

In PRADASE, two indices are provided to optimize range queries and trajectory based queries.

For range queries PMI index can speed up querying, as can be seen from Figure. 5. In map phase, Each data node adopt query parser to generate a candidate partition list which consists of all possible partitions stored at local and corresponding sub queries. Then each data node reads data

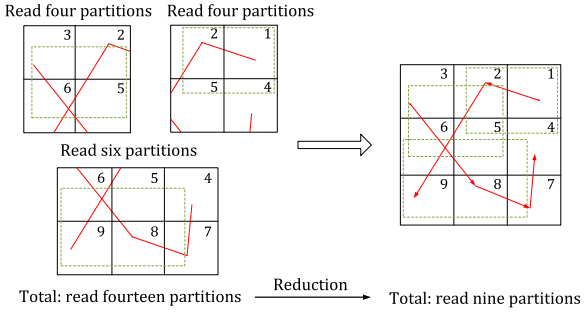

Figure 6: Batch Enhancement of Spatio-temporal Range Query

from local and executes sub queries, then outputs results for each sub query in the form of <objectID, {segment}>. After grouping by object identifier, reduce phase collects sub results with the same object identifier, then executes join operation for each object's all sub segments and finally returns the results <objectID, {trajectory segments}>. OII index can speed up trajectory based queries. Given any object ID, the system can locate the address where the data store and read this object's all historical trajectories.

3.4.1 *Batch Enhancement*

PRADASE can largely optimize the batched querying. For trajectory based queries, if only a small quantity of objects are searched from OII, resource utilized ratio will be low and most of nodes in cluster will be idle, whereas batched query is good for resource utilization. For range queries, the system can parse batched queries, and reduce the cost of querying common range. As can be seen from Figure 6, the green curves present query range and the red curves represent trajectories of moving objects. Three range queries are submitted to the system. Each query needs to read trajectories from data chunk and the corresponding number are three, three and six. If they are batched execution, the com-

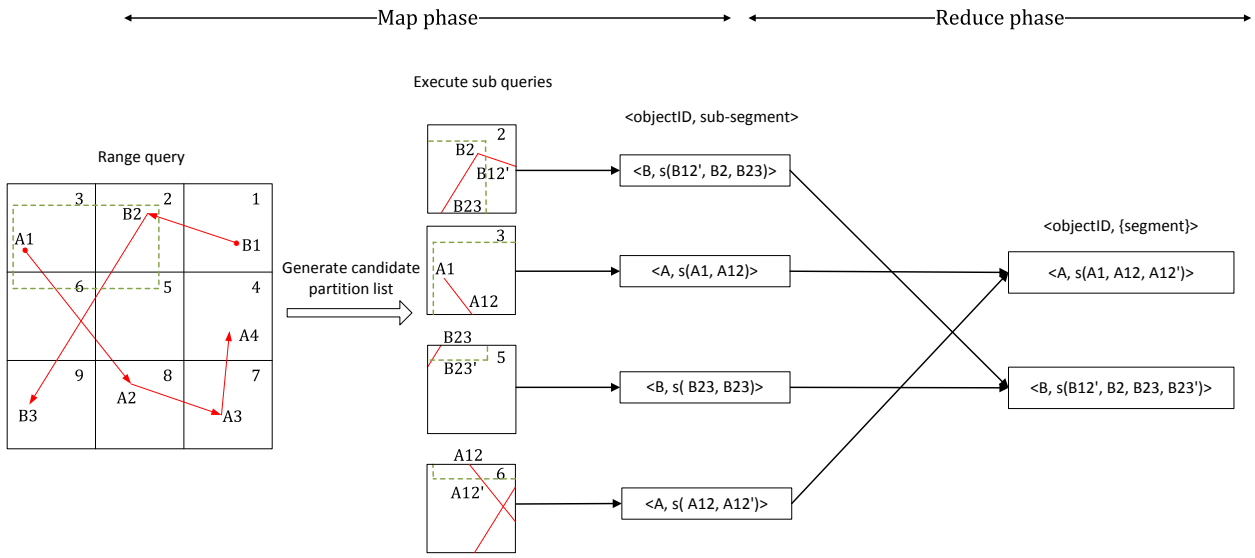

Figure 5: Processing of PMI based Spatio-temporal Range Query

mon data chunk can be read only once. So the number of data chunk which need reading reduces to nine from fourteen which is the summation of three single queries.

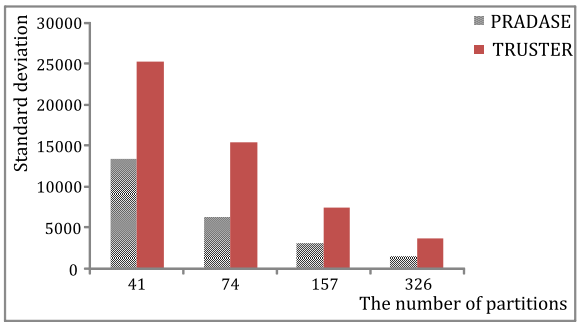

Figure 7: Standard Deviation of Partitioning

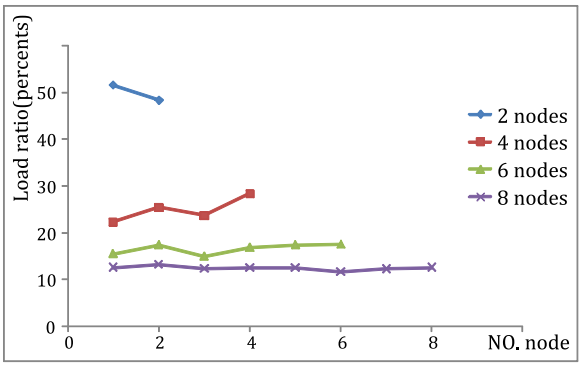

Figure 8: Load Balance of PRADASE

4. EXPERIMENTAL STUDY

In this section, the experimental evaluation for PRADASE is presented. The experiments mainly focus on load balance and scalability of PRADASE system. Moreover, data importing, indexing creation and query processing for the trajectory data are also taken into account in terms of their performance respectively. It is worth noting that most existing traditional centralized technologies are not suitable for massive trajectory data over large clusters. Thus, they are not suitable to compare with PRADASE in terms of performance.

4.1 Experiment Setting

PRADASE is built on Hadoop [1] version 0.19.0, and consists of eight nodes. Each node runs Ubuntu Linux version 8.04 on Pentium IV 1.7GHz CPU and 512M memory. In order to compact with Hadoop platform, the experiments is implemented in Java SDK 1.42. All trajectory segments with the same PMI or OII partitions are stored in one data chunk which is deployed on top of Hadoop distributed file system.

Network-based Generator of Moving Objects [2] is adopted to generate the massive trajectory data set. This data set is generated to simulate moving objects in a network of paths such as roads, and consists of 300,000 objects moving in two dimensional geographical spaces.

4.2 Load Balance

Load balance of the whole cluster is very important for data processing in large clusters. For trajectory data processing on clusters, the load balance highly depends on the partition strategy. TRUSTER [18] is a system specialized in evaluating trajectory based query over massive data set on large clusters. We compare the balance of splitting data set between TRUSTER and PRADASE, which is indicated in Figure 7. The abscissa represents the number of partitions, and the ordinate represents the value of standard deviation which is computed from data size of each partition. In this experiment, a quadtree based partition method is adopted, which executes by recursively subdividing the underlying denser partition into four sub-partitions. The quadtree based partition method can't accurately control

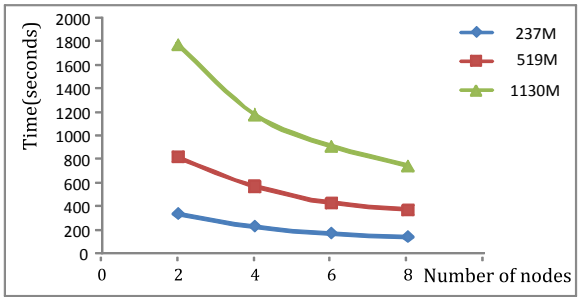

Figure 9: Data Importing with PMI

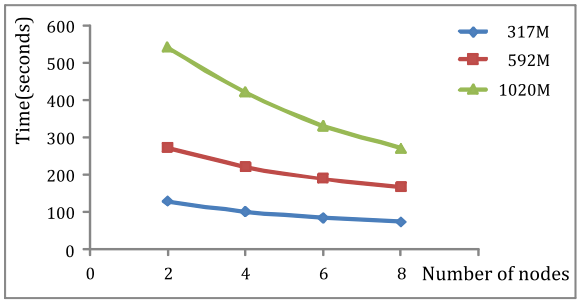

Figure 10: Data Importing with OII

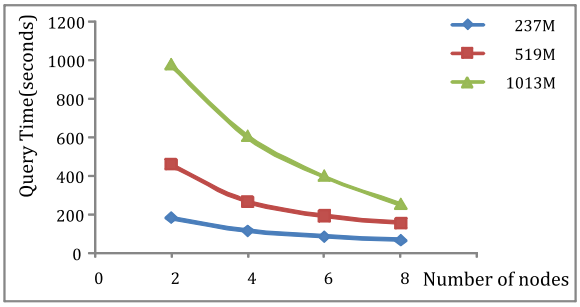

Figure 11: Spatio-temporal Range Query Processing with PMI

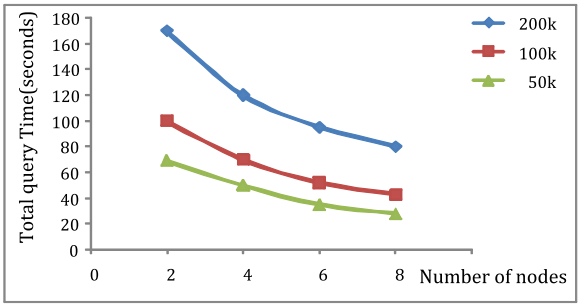

Figure 12: Trajectory Base Query Processing with OII

the number of partitions in the process of training partitioning, while it is only controlled in a range. In the experiment, the variation of partition number in Figure 7 is controlled in [-5,+5] based on he specific point. The standard deviations in PRADASE are less than those in TRUSTER in terms of all four groups of partitions. It evidently indicates that the partition method of PRADASE is better than that of TRUSTER's. On the other hand, the standard deviation decreases with the increasing number of partitions.

The load balance of the PRADASE cluster is shown in Fig. 8. The abscissa represents the identifier of each node in the cluster, and the ordinate represents the load ratio of the corresponding node. The smoother curves indicate what the better load balance of the whole system gains. We can find out that more nodes the PRADASE system has, the better load balance the PRADASE system gains. This suggests that the PRADASE system is highly scalable.

4.3 Data Importing and Index Creating

The efficiency of trajectory data importing and index creating is evaluated in this set of experiments. When the data set is imported into the system, two kinds of indices, PMI and OII are created correspondingly. The performance of importing data with PMI and OII is shown in Figure. 9 and Figure. 10 respectively. The abscissa represents the number of nodes, and the ordinate represents the total response time. Three data sets with different sizes are imported into the system separately. The number of nodes in PRADASE system varies from 2 to 8. Larger data set results in higher execution time. The performance of data importing with both PMI and OII improves by increasing the number of nodes in PRADASE system, whereas the rate of growth decreases. More nodes in the cluster make the whole system much more efficient, especially for large data set. This experiment shows that the data importing and index creating on PRADASE is highly scalable so that we can resolve the explosion of data by increasing the number of computer in clusters.

4.4 Query Processing

Finally, the performance of query processing on PRADASE system is evaluated. The performance of spatio-temporal range query on PMI is shown in Figure. 11. The abscissa represents the number of nodes in the cluster, and the ordinate represents the total query response time. In the experiment on range queries, we choose the largest range (i.e. the whole spatio-temporal space as the query range). The range query is evaluated on three data sets with different sizes, 237M, 519M and 1130M. Since the query range is the whole spatio-temporal space, all the trajectory records are fetched as the final result, which is the worst case of a range query. As we can seen, the query response time decreases as the number of nodes increases.

To evaluate the trajectory based query, we choose the largest data set with 1020M which is used in experiment for OII creation (Figure 10). As for the parameters, 50k, 100k, 2000k, and the entire objects in the temporal space are randomly chosen. The performance of trajectory based query is shown in Figure. 12. The abscissa represents the number of nodes in cluster, and the ordinate represents the total query time. Each query is executed on the cluster with different number of nodes. Obviously, the total response time of queries decreases when the number of nodes increases. These two experiments prove that query processing on PRADASE is highly scalable and can reduce execution time by increasing number of nodes. Both MPI and OII index methods are efficient and scalable, thus it provides foundations for trajectory processing in MapReduce framework.

5. RELATED WORK

Various techniques have been proposed for processing trajectory data in moving object environments. Most existing works solve the problem in a centralized manner. In particular, the locations of different moving objects are reported to a single server, and these location data are recorded as trajectories in the central server. Receiving different query requests, the single server searches the whole trajectory data set, and returns the eligible trajectories as query result. In order to facilitate the trajectory data processing efficiently, different kinds of indexing methods are proposed in the centralized manner. The R-tree [9] family is the most often used indexing structures. Different proposals on R-tree index the historical positions of moving objects in order to support spatiotemporal range queries. The 3D R-tree [17] treats the time dimension as another spatial dimension in order to process spatial and temporal queries uniformly. The HR-tree (historical R-tree) [12] maintains a R-tree each timestamp. Only the updated data node needs to be stored separately. The MV3R-tree (Multi-version 3D R-tree) [16] integrates the HR-tree and the 3D R-tree.

Besides the R-tree indexing family, some grid based indexing methods are also proposed. SETI [3] is based on a static partitioning of the spatial dimensions. Trajectories are partitioned into different cells according to spatial dimensions, and temporal dimension of all the trajectories in the same cell are indexed by an 1D R-tree. Since SETI just employs a static uniform partitioning methods, it cannot deal with the skewed trajectory data efficiently. The size of grid cell is also critical to the performance of such indexing structure [11]. As the trajectories are partitioned into different cells, the grid based indexing method can be used in a distributed environment. In our previous work [18], we propose a distributed indexing method. Trajectories are distributed into different nodes according to the spatial extent, and temporal indexing is built on each node. Accordingly, spatio-temporal range queries also needs to be distributed to different nodes to process.

6. CONCLUSIONS AND FUTURE WORK

This paper presents massive trajectory data procession based on MapReduce framework. Large amount of historical trajectory data set is partitioned into data chunks according to two different strategies, (i.e. MPI and OII). MPI partitions the trajectory based on the spatial extent, while OII partitions the data set based on object identifiers. These data chunks are distributed to different nodes in the cluster, thus it makes the whole system more available and reliable. Based on the two indexing methods, spatio-temporal range query and trajectory based query can be processed efficiently in the framework. A comprehensive experimental study is also conducted relaying on the proposed framework. The results suggest that our proposal is efficient and scalable, and the MapReduce-based framework is a promising paradigm for the trajectory data process.

Several interesting research directions exist. More heuristic partitioning methods may improve the performance of the whole clusters. Too many communications between computers may lead to low efficiency, thus reducing data migration is a very urgent problem in MapReduce-liked framework. Exploring the opportunities for data analyzing trajectory data in the framework of PRADASE is much more useful. It is not efficient for real time queries, therefore better technologies with or without index need much more researches.

7. REFERENCES

[1] Apache. Hadoop. http://hadoop.apache.org/core/.

[2] T. Brinkhoff. A framework for generating network-based moving objects. GeoInformatica, 6(2):153-180, 2002.

[3] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with seti. In CIDR, 2003.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data (awarded best paper!). In OSDI, pages 205-218, 2006.

[5] H. chih Yang, A. Dasdan, R.-L. Hsiao, and D. S. P. Jr. Map-reduce-merge: simplified relational data processing on large clusters. In SIGMOD Conference, pages 1029-1040, 2007.

[6] F. Dabek, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. In SOSP, pages 202-215, 2001.

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, pages 137-150, 2004.

[8] S. Ghemawat, H. Gobioff and S.-T. Leung. The google file system. In SOSP, pages 29-43, 2003.

[9] A. Guttman. R-trees: A dynamic index structure for spatial searching. In SIGMOD Conference, pages 47-57, 1984.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In EuroSys, pages 59-72, 2007.

[11] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch. Main memory evaluation of monitoring queries over moving objects. Distributed and Parallel Databases, 15(2):117-135, 2004.

[12] M. A. Nascimento and J. R. O. Silva. Towards historical r-trees. In SAC, pages 235-240, 1998.

[13] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In SIGMOD Conference, pages 1099-1110, 2008.

[14] B. C. Ooi, B. Yu, and G. Li. One table stores all: Enabling painless free-and-easy data publishing and sharing. In CIDR, pages 142-153, 2007.

[15] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches in query processing for moving object trajectories. In VLDB, pages 395-406, 2000.

[16] Y. Tao and D. Papadias. Mv3r-tree: A spatio-temporal access method for timestamp and interval queries. In VLDB, pages 431-440, 2001.

[17] Y. Theodoridis, M. Vazirgiannis, and T. K. Sellis. Spatio-temporal indexing for large multimedia applications. In ICMCS, pages 441-448, 1996.

[18] B. Yang, Q. Ma, W. Qian, and A. Zhou. Truster: Trajectory data processing on clusters. In DASFAA, pages 768-771, 2009.