Parallel Indexing for Past, Current and Future Locations of Moving Objects

Ying XIA\textsuperscript{a}, Zhen HUANG\textsuperscript{b}, Xu ZHANG\textsuperscript{c}, Hae-Young BAE\textsuperscript{d}

Research Center of Spatial Information System, Chongqing University of Posts & Telecommunications, Chongqing, China

\textsuperscript{a}xiaying@cqupt.edu.cn, \textsuperscript{b}cqhuangzhen@sina.com, \textsuperscript{c}zhangx@cqupt.edu.cn, \textsuperscript{d}hybae@inha.ac.kr

Keywords: Parallel Computing, Moving Object, Spatio-temporal Index, Location Querying, Trajectory Prediction.

Abstract. Facing huge amounts of location and trajectory data of moving objects, although cloud database systems based on Key-Value mechanism could perform better in scalability than traditional spatio-temporal database systems, it could not provide efficient access method to support querying the locations of moving objects in past, current and future. A parallel index method for past, current and future locations of moving objects named PIPCF is proposed. It splits the space into areas and uses Quad-Tree to manage them first, and then combines temporal property of the data to index the moving objects in each area by R-Tree. Furthermore, a hash table is used to help managing predicted trajectory unit to accelerate the index updating. The experiment shows that PIPCF could perform well in cloud computing environment and improve the querying performance of the locations of moving objects in past, current and future.

Introduction

In recent years, with the development of the location determination technology and the popularization of the mobile devices, variant location-based services produce mass location and trajectory data. Although traditional spatio-temporal database systems have mature storage, index and query scheme, they could not meet the requirement both in scalability for processing huge amounts of location and trajectory data and performance of the location query of moving objects in past, current and future.

To solve the scalability problem for processing huge amounts of data, cloud database systems represented by BigTable\cite{1} are released. This type of the databases partition the data into several parts and distribute them to each node of the cluster to improve the parallel processing performance. As one of the cloud data management technologies, Key-Value mechanism gains properties of high scalability, availability and fault tolerance, which can achieve the efficient storage management of massive data. However, cloud database systems based on Key-Value mechanism such as HBase only support efficient query on the row key, whole data set will be traversed if query is performed on other attribute. In order to support massive location data query in past, current and future, the spatial and temporal characteristics of location and trajectory data are considered, and a parallel index method for past, current and future (PIPCF) is performed.
Research Basis of Index for the Past, Current and Future Location in Parallel Mode

There are many research have conducted on the spatial and temporal indexing mechanism of moving objects in parallel mode. PMI (Partition based Multilevel Index)[2] bases on Hadoop platform, it uses MapReduce to split the spatial area and distribute data according to temporal character. MD-HBase [3] takes HBase as storage layer, uses Quad-Tree or KD-Tree to split space into areas, and then index the data in each area. In order to solve the problem of frequent updating of location data, ToSS-it (Throwaway Spatial Index Structure)[4] takes advantage of the concurrence computing of the distributed environment to reconstruct the index according to the latest location data.

Location query in past, current and future are necessarily for variant location-based services, but above parallel indexes do not concern more about it. By way of contrast, this category of index is more abundant in traditional spatio-temporal database systems. It can be divided into two types, one is based on Euclidean space and another is based on road network constraint. Indexes based on the Euclidean space [5-7] use the Euclidean trajectory unit as the basic to organize index. Each Euclidean trajectory unit is a line segment in X×Y×T dimensions. This kind of index requires high concurrence of performance because it needs a large amount of line segments to replicate the trajectory of the moving object. The index based on road network constraint [8-9] generally is two layer structures. The top layer is index for static road network, while the bottom layer is index for moving object on the corresponding road (road segment). This type of index has high requirements about the overall structure and integrity of the road network data, and it could not index the moving objects out of the road network.

Definitions

For simplicity, we use Euclidean space to describe the trajectory of moving objects. Related definitions are as follows:

Definition 1 Location of moving object \( p = (\text{lat}, \text{lng}, t) \). \( p \) describes the moving object at the location where its latitude is lat and longitude is lng at the time \( t \).

Definition 2 Motion vector of moving object \( \text{mv} = (p, v, \text{direction}) \). \( p \) is location of the moving object, \( v \) is the speed, and direction is GPS azimuth where its value range is \([0, 360]\).

Definition 3 Spatio-temporal trajectory of moving object \[ \text{trajectory} = (\text{mv}_i)_{i=1}^n (n \geq 2) \].

Definition 4 History trajectory unit of moving object marked as \( U_i = u(\text{mv}_{i-1}, \text{mv}_i)(1 \leq i < n) \). \( \text{mv}_{i-1} \) and \( \text{mv}_i \) are continuous motion vectors. The projection of the history trajectory unit on the spatial area is a line segment.

Definition 5 Predicted trajectory unit of moving object marked as \( U_p = u(\text{mv}_n, \text{mv}_p) \). \( \text{mv}_n \) is the latest motion vector and the \( \text{mv}_p \) is the predicted motion vector based on the \( \text{mv}_n \). The projection of the \( U_p \) on the spatial area also is a line segment.

PIPCF Data Structure and Algorithm Design

PIPCF Data Structure

PIPCF is two layer structure as shown in Fig.1. The top layer uses Quad-Tree to split space into several areas, and it is the global static index. The bottom layer consists of R-Tree forest. Each leaf node of the top layer Quad-Tree corresponds to one R-Tree of the bottom layer.
makes the R-Tree of the bottom layer processes the data in each area in parallel easily through splitting the spatial area.

The structure of the top layer Quad-Tree leaf node is \(<\text{nodeid, MBR, rootid}>\), \text{nodeid} is the identity of the leaf node. \text{MBR} is the minimum bounding rectangle of this node. \text{rootid} is the root node identity of the R-Tree corresponded to this node. According to properties of R-Tree, the root node of R-Tree might be changed during the data insertion. To avoid asynchronous and queuing problems caused by frequent modification of R-Tree root nodes in a parallel environment, we specify that once the R-Tree is generated, the root node will not be modified. We just need exchange the new node and root node when the root node is splitting. The structure of the Quad-Tree non-leaf node is \(<\text{nodeid, MBR, children}>\), children is the list of the identities of the node’s children.

\[ \text{Figure 1. PIPCF Structure.} \quad \text{Figure 2. Hash Table Structure.} \]

The structure of the bottom layer R-Tree leaf node is \(<\text{nodeid, parent, MBR, mid, U(U_p)}>\), parent is the identity of this node’s parent and mid is the identity of the moving object. The structure of the non-leaf node is \(<\text{nodeid, MBR, parent, children}>\). Furthermore, we fetch in a hash table based on the main memory to accelerate the processing of predicted trajectory unit removing. The hash table holds the predicted trajectory unit of each moving object. When the bottom layer removes the predicted trajectory unit, it could get the R-Tree leaf node by checking the hash table directly, while not traverse the whole tree. The structure of the hash table is as shown in Fig.2. The key of the hash table is the moving object identity mid and the value is the latest predicted trajectory unit \(U_p\) corresponded to the moving object whose identity is mid.

\section*{Trajectory Prediction}

In order to provide the query for short future location of moving object, trajectory need to be predicted. The method mentioned in article [9] is adopted to predict the trajectory. The method assume that the moving object will be keep in uniform rectilinear motion in the future time \(t\) and then calculate the motion vector at time \(t_c + t\) according to the current motion vector. \(t_c\) is the current time. For the example shown in Fig.3, we assume that point \(B\) is the current location of the moving object and its motion vector is \(mv_n\). The speed, speed direction and the time correspond to the motion \(mv_n\) are \(v\), \(d\) and \(t_n\). The moving object will be at point \(C\) after time \(t_c + t\). We can calculate the distance according to Eq.1.
\[ \text{distance} = v \cdot t \]  

Then we can estimate the location of the point \( C \) base on \( d \) as shown is Eq.2 and Eq.3.

\[
\text{lat}_C = \text{lat}_B + \frac{\text{distance} \cdot \cos\left(d \cdot \frac{\pi}{180}\right)}{\text{lengthperlat}}
\]

\[
\text{lng}_C = \text{lng}_B + \frac{\text{distance} \cdot \sin\left(d \cdot \frac{\pi}{180}\right)}{\text{lengthperlat} \cdot \cos\left(\text{lat}_B \cdot \frac{\pi}{180}\right)}
\]

\( \text{lengthperlat} \) is the length of one degree of latitude on one longitude line. According to the definitions, the location of point \( C \) is \( p_c = (\text{lat}_c, \text{lng}_c, t_{n+1} + t) \), and the motion vector \( \text{mv}_c = (p_c, v, d) \). Finally, the predicted trajectory \( U_p = (\text{mv}_n, \text{mv}_c) \).

**Trajectory Unit Splitting**

As spatial area is split into areas by the index, we need to split the trajectory unit to make sure the trajectory unit can be indexed by the corresponding R-Tree if the trajectory unit cross more than one area. As shown in Fig.4, the trajectory corresponding to trajectory unit \( u(AB) \) is the line segment \( AB \). \( AB \) crosses 3 areas, so we need to split \( AB \) into 3 line segments according to the intersection points between the boundary of the areas and \( AB \). The line segments are \( AC, CD \) and \( DB \) as shown in the Fig.4. After splitting, these trajectory units corresponding to the line segments can be inserted into the R-Tree of the bottom layer correctly.

**PIPCF Creation, Maintenance and Query Algorithm**

At the beginning of index creation, we need to set range of the spatial area and the minimal size of the area, the spatial area will be split into several areas and the R-Trees corresponding to each area are empty.

The creation and maintenance algorithm see algorithm 1.
Algorithm 1 PIPCF creation and maintenance algorithm

Input:
- mid //moving object identity
- time //the information collection time
- lat //the latitude of the location
- lng //the longitude of the location
- v //the speed of the moving object
- direc //the direction of the speed

Output:
1. If(quadtree not exist) Then
2. quadtree start to split the spatial area.
3. EndIf
4. \( \text{mv}_n = \text{genCurMV}(\text{time}, \text{lat}, \text{lng}, v, \text{direc}); \) //Generate the current motion vector
5. \( \text{u}(\text{mv}_{n-1}, \text{mv}_{p-1}) = \text{getPreTrajUnit(mid)}; \) //Get the latest predicted trajectory unit
6. If(\( \text{u}(\text{mv}_{n-1}, \text{mv}_{p-1}) \) exist) Then //If the latest predicted trajectory unit exists
7. \( \text{rtree} = \text{getRtreeByTrajUnit(u(mv_{n-1}, mv_{p-1}))}; \)
8. delete\text{PreTrajUnit(tree,u(mv_{n-1}, mv_{p-1}));} //Deleting it from the R-Tree
9. EndIf
10. \( \text{mv}_{n-1} = \text{getLatestMV}(); \) // Get the latest motion vector
11. \( \text{mv}_p = \text{predictMV(mv}_{n}); \) // Predicting trajectory
12. If(\( \text{mv}_n\text{.time-mv}_{n-1}\text{.time}< \text{maximal duration} \)) Then
13. \( \text{u}(\text{mv}_{n-1}, \text{mv}_n) = \text{genTrajUnit(mv}_{n-1}, \text{mv}_n); \) //Generating the trajectory unit
14. EndIf
15. \( \text{u}(\text{mv}_n, \text{mv}_p) = \text{genTrajUnit(mv}_{n}, \text{mv}_p); \) //Generating the trajectory unit
16. If(\( \text{u}(\text{mv}_{n-1}, \text{mv}_n) \) exist) Then
17. list\text{TrajUnit}=split\text{TrajUnit(quadtree, u(mv}_{n}, mv_{n-1});} // splitting the trajectory unit
18. EndIf
19. list\text{TrajUnit} = list\text{TrajUnit} \cup \text{split\text{TrajUnit(quadtree, u(mv}_{p}, mv_{n});}}
20. For(trajUnit: list\text{TrajUnit}) Do // insert all of the trajectory unit into the R-Tree
21. \( \text{rtree} = \text{getRtreeByTrajUnit(trajUnit}); \)
22. insert(\text{rtree, trajUnit});
23. EndFor

PIPCF mainly supports range query. For the range query \( Q(\Delta x, \Delta y, \Delta t), (\Delta x, \Delta y) \) is the range of the spatial and \( \Delta t \) is the range of the time. \( \Delta t \) could be the period of the past or future. During the query processing, PIPCF will get the areas according to the spatial area firstly, and then get trajectory unit satisfied \( \Delta t \) in each R-Tree corresponding to the areas obtained by the first step. If some of areas cover the spatial range partly, it is necessary to filter the results again.

The range query algorithm see algorithm 2.

Algorithm 2 PIPCF range query method

Input:
- \((\Delta x, \Delta y, \Delta t)\) //Spatial-temporal range

Output:
- \( \text{resultSet} \) //moving object identity set
1. resultSet←Φ
2. According to the spatial range(Δx, Δy), getting the pair through query the Quad-Tree $p=(\text{rootid}_i, \text{isFullContain}_i, \text{rect}, \Delta x, \Delta y, \Delta t)^n_{i=1}$
3. For($i=1$ to $n$) Do
4.  $r\text{tree} = \text{R-Tree corresponding to rootid}_i$;
5.  $\text{tmpRstSet=search(rtree, } \Delta t);$ // Querying the bottom layer R-Tree forest.
6.  If   (¬ $\text{isFullContain}_i$ ) Then //if there are some areas cover the spatial range partly.
7.  resultSet $\cup =$refiltrate(tmpRstSet, $\Delta x, \Delta y$); // Filter the result set again.
8.  Else
9.  resultSet $\cup =$ tmpResultSet;
10.  EndIf
11.  EndFor
12.  Return resultSet;

**Experimental Evaluation**

**Experiment Setting**

The experiment choose the NDTR-Tree for comparison, it has similar structure and function with PIPCF. The comparison is focus on the query response time and update time in high concurrence environment.

The experiment use HBase to organize and store data. The HBase deploy on the 10 nodes cluster. One node of the cluster is the Master, the others are RegionServer. The experimental environment is shown in Table 1. And the data set is produced by 12000 Beijing taxies in one month (http://www.datatang.com/data/45888/). Each record contains 9 attributes, such as the latitude, longitude, speed and the direction of the speed etc.

<table>
<thead>
<tr>
<th>CPU</th>
<th>OS</th>
<th>Memory</th>
<th>Hadoop</th>
<th>HBase</th>
<th>ZooKeeper</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2GHz</td>
<td>CentOS6.5</td>
<td>8G</td>
<td>2.5.2(64bit)</td>
<td>1.1.2</td>
<td>3.4.7</td>
</tr>
</tbody>
</table>

**Experimental Result Analysis**

Fig.5 is the comparison between PIPCF and NDTR-Tree about the update time in different number of moving objects concurrence insertion. We can find that PIPCF performs a little faster than NDTR-Tree. The reason is that PIPCF uses the hash table to accelerate delete predicted trajectory units while NDTR-Tree need to traverse the whole tree to delete those units.
The increase of the number of the moving objects, the update time of the NDRT-Tree increase rapidly while the PIPCF appears more stable. That is because it has pre-split the region into two regions. It means that the PIPCF is running on two nodes of the cluster at the same time. We can see that the PIPCF has more advantages in high concurrency insertion.

Fig.6 shows different number of the moving objects influence on the query response time of the PIPCF and NDTR-Tree. To ensure objectivity, we also generate 5400 groups of spatial range randomly and unify the time range in 100s to 200s. We can see from the Fig.6 that there is no obvious difference between PIPCF and NDTR-Tree when the number of the moving objects below 6000, because PIPCF just works on one node of the cluster. With the number of moving objects increase, PIPCF works on two nodes to process the query. So PIPCF much faster than the NDRT-Tree when the number of the moving objects becomes bigger. In addition, we also fetch in a hash table based on the main memory to accelerate the update of the bottom layer R-Tree.

Conclusion

In order to support query for the location of moving objects in past, current and future efficiently in cloud computing environment, a two layer index PIPCF is proposed. PIPCF uses the Quad-Tree split the spatial area into several areas as a static index and then use R-Tree index the data in each area. The experiment shows the PIPCF could improve the performance of the query for the location of moving objects in past, current and future especially in high concurrence environment.

Acknowledgement

This work is funded by Chongqing Natural Science Foundation (cstc2014kjrc-qnrc40002), Scientific and Technological Research Program of Chongqing Municipal Education Commission (KJ1500431), and WenFeng Creative Foundation of CQUPT (WF2014-05).

References


