A Multilevel Indexing Method for Approximate Geospatial Aggregation Analysis

Luo CHEN†(a), Member, Ye WU†(b), Wei XIONG†, and Ning JING†, Nonmembers

SUMMARY In terms of spatial online aggregation, traditional stand-alone serial methods gradually become limited. Although parallel computing is widely studied nowadays, there scarcely has research conducted on the index-based parallel online aggregation methods, specifically for spatial data. In this letter, a parallel multilevel indexing method is proposed to accelerate spatial online aggregation analyses, which contains two steps. In the first step, a parallel aR tree index is built to accelerate aggregate query locally. In the second step, a multilevel sampling data pyramid structure is built based on the parallel aR tree index, which contribute to the concurrent returned query results with certain confidence degree. Experimental and analytical results verify that the methods are capable of handling billion-scale data.

key words: aggregation computation, approximate query, spatial index, online analysis

1. Introduction

Geospatial aggregation is a set of aggregate operations on collections of tuples based on granularity of the spatial domain, which returns a single value that summarizes the information contained in those tuples. Geospatial aggregation analysis applications are becoming more common with the increasing growth of demand for geospatial big data analysis and data mining. Examples of such applications include traffic flow monitoring, weather monitoring, natural resources management, tracking of mobile devices, public safety management and travel recommendation systems.

Geospatial aggregation query is time-consuming. In general, aggregation index is used to improve its performance. Aggregation R-tree (aR tree) [1] stores in each tree node the aggregation values of all the objects within the MBR of that node. So there is no need to re-calculate the aggregation value when querying. The structure of ECDF-B-tree [2] makes traditional structure disk-based and dynamic. The structure of aP tree [3] is used to process aggregate window query of planar points. However, due to the huge amount of spatial data, even if the algorithm is improved based on the aggregation index, its performance is still unable to meet the needs of online interaction. In the field of non spatial queries, approximate queries are an effective way to meet online interactive operations. There are two major solutions. The first is the non indexed online approximate aggregation. Hellerstein (et al) [4] extend a database system to return aggregate output in random order, and to calculate the confidence interval on the basis of the control over the relative rate until the result meets the requirements of the user. DBO system [5] can achieve approximate aggregate query, and return to a speculative result very quickly, followed with the continuous execution of the relational operations, thus making the result more and more close to the true value. Han et al [6] proposed an algorithm PAA to efficiently process approximate aggregation with an arbitrary confidence interval. Lazaridis et al [7] using selective traversal of a Multi-Resolution Aggregate (MRA) tree structure to answer non-spatial aggregation queries interactively. Although these methods are effective in the non spatial domain, the research work in the geospatial field is rarely seen at present. Wang et al. [8] propose multilevel indices for spatial online aggregation with sampling techniques. We feel that there is still much work for improvement.

In order to better improve the efficiency of online geospatial aggregation analysis of large scale data, this letter proposes a parallel multilevel index method under the distributed memory based cluster-computing framework, to optimize the efficiency of large scale spatial aggregation query. First, a data structure called HPaR tree is proposed for parallel optimization of the Hilbert aR tree. It improves the efficiency of index building through Hilbert curve based spatial data partitioning and improves the efficiency of parallel queries by building a distributed index structure. Then, a multilevel aggregation R tree called MLAR tree is proposed. It adopts the idea of random sampling technique of big data and multilevel structure of spatial pyramid index, and further improves the efficiency of aggregation computation and enhances the interaction experience through the progressive refinement of probabilistic query.

2. Parallel aR Tree Index for Aggregate Queries

The proposed methods is the Hilbert partition based parallel aggregation R tree (HPaR tree). It is designed to accelerate the accurate geospatial aggregation query (Fig. 1).

Let $D = \{d_j | j \in [1,N]\}$ be the input data, where $d_j$ is single tuple and $N$ is the total number of tuples.

First, we build a $P \times P$ grids $G = \{g_j | j \in [1,P \times P] \}$ for this area, each grid is denoted as $g_j$. Then, a Hilbert space filling curve is built on this grid. Each object is represented as key-value form $(H_{ci}, d_i)$, where $H_{ci}$ is the Hilbert code of
the object.

Second, the data set is sorted by $H_{ci}$, and is divided into $k$ parts, assuming that the parallelism of the processing is $k$. In partitioning processing, we assign all objects within each $g_j$ as a unit. A number of continuous $g_j$ are put together to form a block $D_i (i \in [1, k])$, and meet the following constraints:

1. The data volume in each block is roughly equivalent;
2. The overlapping of the minimum bounding box of each block is as small as possible.

Through the above processing, $D$ is divided into $k$ parts $D_i (i \in [1, k])$, and is distributed to $k$ computing nodes.

Third, in all computing node, local $AR$ tree indexes are built in parallel, as shown in Fig. 1. In each tree node of the local $AR$ tree, the aggregation information is recorded within it, and the aggregation value of the higher level node can be directly gathered from the sub nodes. For example, the aggregation value of area $A1$ is the direct accumulation of sub nodes $a1, a2, a3$.

In the last step, we build the global index based on each local indexes. The global index is a grid index, each index node containing a pointer to a local index, a $MBB$ (Minimum Bounding Box) of the local index, and a precomputed aggregate value.

In the case of a data object crossing multiple grids, we assign the object to a grid, thus to ensure the correctness of the results.

Algorithm 1 shows the basic query processing, which return the accurate result of aggregation query. It first uses global index to implement pruning, and then use local index to carry out an iterative DFS query. After all results in distributed nodes are obtained, the final result is merged and output.

3. Multilevel HPaR Index for Approximate Online Query

In applications of large-scale geo-spatial online analysis, people expect to get the results on the whole in the least amount of time, rather than waiting for a long time to get an accurate result, and there is great interest in methods that provides approximate answers to aggregate queries at a significantly lower computational cost and quick response time, by providing a relatively good estimation of the result without fully accessing large portion of the data set or the index.

Our proposed method uses a multilevel structure. By sampling recursively of the original data, a set of multilevel sample data is obtained. Like a pyramid, the bottom layer is the most detailed, the top layer is the most abstract. Each level of the data is indexed with $HPaR$ tree as depicted in Sect. 1. In order to distinguish from $HPaR$ tree, we call this structure Multi-Level aggregation R tree ($MLaR$ tree).

First, we build a pyramid structure based on multilevel sampling data, as depicted in the left of Fig. 2. Let the height
of the pyramid be \( l \). Layer \( i \) is composed of dataset of sampling \( DS_i (i = 0,..., l) \). \( DS_0 \) is the original data. \( DS_i \) is sampled from \( DS_{i-1} \) with sample rate \( S \). With the construction of the multi-level structure, the amount of data will also increase. \( \sum_{i = 0}^{l} N_l = \frac{1 - S^l}{1 - S} N \). Suppose \( S = 0.5, l = 10 \), then
\[ \sum_{i = 0}^{10} N_l = \frac{1}{1 - S} N = 2N. \]
Then, with each \( DS_i \), we build \( HPaR \) tree \( HPaR_i \) as described in Fig. 1. With the increase of \( i \), \( N_l \) decreases, and the partition number \( k_l \) also decreases.

Afterwards, we use a dynamic cache strategy that combines memory and disk. The high level \( HPaR \) trees are cached in memory as their size is relatively small. Low level \( HPaR \) trees are cached in disk, and will be loaded into main memory when needed.

Among above, \( CD_i \) is the confidence degree, which means the probability that the difference between result and the true value is less than confidence interval \( t \).

Taking the average aggregation function \( AVG(x) \) as an example, we treat each tuple \( d_i \) as an independent sampling of real world events, and \( DS_0 \) can be treated as a set of \( N \) samplings. The expected mean of \( D \) is expressed as \( E[D] \). The average aggregation value of \( D \) is \( AVG(D) = \frac{1}{N} \sum_{i=1}^{N} d_i \).

Then normalize it and make \( AVG(D) \in [0..1] \). According to Hoeffding’s inequality
\[
CD_i = P(|AVG(D) - E[D]| \leq t) \geq 1 - 2e^{-2Nt^2} \tag{1}
\]
\[
CD_0 \geq 1 - 2e^{-2Nt^2} \tag{2}
\]

Therefore, given \( t \in [0..1] \), \( CD_i \) can be estimated based on the current level \( i \) from Eq. (1). We noticed that in each level, there is a lower-bound of confidence degree. Let \( T_i \) be the lower-bound of confidence degree of level \( i \). Then
\[
T_i = 1 - 2e^{-2Nt^2} \tag{3}
\]
\[ \frac{1 - T_i}{1 - T_{i+1}} = \frac{2e^{-2Nt^2}}{e^{-2Nt^2}} = e^{-2Nt^2} = e^{-2N(t(1-S))^2} \tag{4} \]

It can be inferred that when the sampling rate \( S \) is small, the intermediate data set obtained is smaller and the retrieval speed will be faster, but the confidence degree will decrease rapidly.

4. Experiments

4.1 Experimental Setup

The experiments are conducted on the stand-alone environment and the cluster environment. The stand-alone environment is a server with a 32-core Intel CPU and 256GB memories. The cluster environment contains 5 servers, each with 20 CPU cores and 256GB memories. The operating system are both CentOS7.1. We implemented 6 methods in Scala: \( HPaR \), MLaR, GeoSpark [9], Simba [10], Random Sample, Range Query.

Two OpenStreetMap\(^\dagger\) data sets are used in the experiments. One is the OSM data of China, which contains about 50 million vertexes and is about 2.1GB. The second is the OSM data of the whole world, which contains about 1.7 billion vertexes and is about 49.7GB. The two data sets includes nodes and ways. In order to focus on the performance, only latitude and longitude of each nodes in the data sets are used in the experiments.

4.2 Query Performance Evaluation

Suppose \( R \) is the data set to be queried, which has \( N \) vertexes. We sample \( k \) nodes from the raw data to make the query data set. Let \( Q \) be the max bounding box of \( R \) and \( QR \) be the query window. We study the performance of the algorithms by varying \( k \) up to \( N \) for fixed \( QR / Q = 10\% \).

The result in Fig. 3 shows the average of 10 random queries with the same \( k \). The dataset was loaded into main memory completely.

In standalone environment (Fig. 3a), the four methods show the similar performance trends with \( k \) increases. When \( k \) is small, all the four algorithms were able to quickly return the aggregation result because it is fast to scan a small dataset in the memory. The cost of MLaR is lowest out of the algorithms tested. It’s largely because of the two-level index, which prunes lots of unnecessary visit.

Then same experiment is repeated on the full OSM data with 1.7 billion nodes. The result is presented in Fig. 3b. Because the dataset is so large that we can not compute it in acceptable time. The experiment was performed on a cluster

\(^\dagger\)https://www.openstreetmap.org
with 10 Gigabit network. When $k$ is small, the four algorithms can return result quickly. However, it cost more time on cluster than on a single machine, because of the additional network overhead, compared to their corresponding ones.

It is also interesting to study how the size of query window $Q_R/Q$ affects the performance of the algorithms when $k$ is fixed. We fixed $k$ to the full size of the two datasets, respectively, in standalone and cluster environments. For each $Q_R/Q$, ten $Q_R$s are randomly chosen. Figures 4a and 4b present the results on both datasets.

For both datasets, $MLaR$ and $HPaR$ stay mostly constant, and show a gradual decline cost when $Q_R/Q$ increases. This is expected because a larger $Q_R$ will cover more MBRs of the dataset, resulting less scanning in local index. $GeoSpark$ presents first decreased and then an upward trend, because every partition of $GeoSpark$ R-tree contains the aggregation information of the whole partition. Once again, $MLaR$ has the best performance on both environments and datasets, and has roughly outperformed other methods by 1-2 orders of magnitude.

The aggregation result may be estimated as time elapses. In order to evaluate the quality of our methods, we compare the result on $MLaR$ tree with standard range counting queries on R-trees.

Ten random queries were chosen and the average result were given. For each query, the exact aggregation result is determined with a range query, and the time used is measured. Figure 5 shows the relative error of the estimation. We can discover vast error variations at the beginning because the query on top level of $MLaR$ is quite inaccurate. However, the estimation can be quite accurately with a small amount time and tends to the truth-value. We believe the quality of the estimation is acceptable in practice.

5. Conclusions

In this letter, we show that it is possible to produce progressive estimating aggregation result for large spatio-temporal datasets. We first design a parallel aR tree for exact aggregation analysis, and then we design a multilevel aR tree for approximate geospatial aggregation analysis, and estimate the confidence degree based on Hoeffding’s inequality. Experimental results show that our methods are more efficient and scalable than existing methods.

Acknowledgments

This work is supported in part by the National High Technology Research and Development Program of China (Grant No. 2015AA123901) and National Science Foundation of China (Grant No. 41471321).

References

[1] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, “Efficient OLAP operations in spatial data warehouses,” International Symposium on Spatial and Temporal Databases, vol.2121, pp.443–459, 2001.
[2] D. Zhang, V.J. Tsotras, and D. Gunopulos, “Efficient aggregation over objects with extent,” Acm Sigmod-sigact-sigart Symposium on Principles of Database Systems, pp.121–132, 2002.
[3] Y. Tao, D. Papadias, and J. Zhang, “Aggregate processing of planar points,” International Conference on Extending Database Technology: Advances in Database Technology, vol.2287, pp.682–700, 2002.
[4] J.M. Hellerstein, P.J. Haas, and H.J. Wang, “Online aggregation,” Acm Sigmod Record, vol.26, no.2, pp.171–182, 1997.
[5] C.M. Jermaine, S. Arumugam, A. Pol, and A. Dobra, “Scalable approximate query processing with the DBO engine,” ACM Trans. Database Syst., vol.33, no.4, pp.1–54, 2008.
[6] X. Han, J. Li, and G. Hong, “Paa: An efficient approximate aggregation algorithm on massive data,” Journal of Computer Research & Development, vol.51, no.1, pp.41–53, 2014.
[7] I. Lazaridis and S. Mehrota, “Progressive approximate aggregate queries with a multi-resolution tree structure,” ACM SIGMOD Record, vol.30, no.2, pp.401–412, 2001.
[8] L. Wang, R. Christensen, F. Li, and K. Yi, “Spatial online sampling and aggregation,” Proc. VLDB Endowment, vol.9, no.3, pp.84–95, 2015.
[9] J. Yu, J. Wu, and M. Sarwat, “Geospark: A cluster computing framework for processing large-scale spatial data,” Sigspatial International Conference on Advances in Geographic Information Systems, pp.1–4, 2015.
[10] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo, “Simba: Efficient in-memory spatial analytics,” International Conference on Management of Data, pp.1071–1085, 2016.