A Novel Spatio-temporal Data Storage and Index Method for ARM-based Hadoop Server

Laipeng Han[1] Lan Huang[1] Xueyi Yang[1] Wei Pang[2] Kangping Wang[1*]

[1]College of Computer Science and Technology, Jilin University, Changchun, Jilin Province, China, 130012
[2]School of Natural and Computing Sciences, Unversity of Aberdeen, Aberdeen, UK, AB24 3UE

Abstract. During the past decade, a vast number of GPS devices have produced massive amounts of data containing both time and spatial information. This poses a great challenge for traditional spatial databases. With the development of distributed cloud computing, many high-performance cloud platforms have been built, which can be used to process such spatio-temporal data. In this research, to store and process data in an effective and green way, we propose the following solutions: firstly, we build a Hadoop cloud computing platform using Cubieboards2, an ARM development board with A20 processors; secondly, we design two types of indexes for different types of spatio-temporal data at the HDFS level. We use a specific partitioning strategy to divide data in order to ensure load balancing and efficient range query. To improve the efficiency of disk utilisation and network transmission, we also optimise the storage structure. The experimental results show that our cloud platform is highly scalable, and the two types of indexes are effective for spatio-temporal data storage optimisation and they can help achieve high retrieval efficiency.

1 Introduction

In recent years, with the advancement of mobile technologies, smart phones are becoming more and more popular. This greatly enriches the way people access to information and thus huge amounts of data have been generated. A survey shows that the scale of China's mobile Internet users had reached 620 million by the end of 2015, and many enterprises have accumulated a vast amount of personal location information. It is also pointed out that 74% of the smart phone users use geographic information [1], which makes the related technologies and application model of LBS (Location -based Service) develop more and more rapidly [2].

With the development of LBS applications, the spatio-temporal data to be dealt with show a trend of rapid growth in terms of both quantity and complexity. Some researchers use NoSQL databases [3] (such as BigTable, HBase, and Cassandra) to store spatio-temporal data, but these databases do not consider the particularity of geographical information, and they do not support spatio-temporal query very well. Other researchers who study data storage hope to solve

the problem of massive storage of spatio-temporal data, such as PIST database [4] and TrajStore database [5], both of which can deal with spatio-temporal data. However, these systems analyse the spatio-temporal data in non-distributed storage environments, and they cannot process big data effectively.

Indexing is very important for massive data querying and manipulation. R-tree [6] is a highly balanced tree structure which can be effective when dealing with spatial data. On the basis of R-tree, some researchers put forward many improved approaches, such as R*-tree [7]. Beckman *et al.* [7] improved the R-tree algorithm in inserting and deleting. Singh S *et al.* [8] proposed a hybrid index method based on inverted index and R-tree. All of the above research built index and query methods in a single machine and they did not consider how to make use of such indexes and query methods in a distributed environment. Ahmed Eldawy *et al.* developed SpatialHadoop [9], a system that is based on Hadoop [10] and uses distributed storage space to greatly improve the efficiency of retrieval. This system changes at the HDFS level and has pioneering significance in building the Grid spatial data, R-tree, and R+-tree index. However, this system is only effective for the spatial data and it is not effective for spatio-temporal data.

In this paper, we built a green ARM-based spatio-temporal database which employs the distributed storage technology, distributed index technology, as well as the distributed computing algorithm based on the Hadoop system. We proposed two distributed indexing mechanisms for spatio-temporal data, and our research has the following three novel contributions: (1) we design two new spatio-temporal partition algorithms, TGrid (the spatio-temporal grid algorithm) and QaDTree (Quadtree-3Drtree algorithm), and apply these two algorithms to different distributions of spatio-temporal data; (2) in the local index phase, we improve the query efficiency by building a one-dimensional time index and a multidimensional R-tree index; (3) we improve the efficiency of network transmission by optimising the storage structure and reducing disk I/O.

The ARM board is cost effective and energy-efficient, but it has low performance and slow network speed. In order to adapt to such a platform, we use the column storage and a compression algorithm to process spatio-temporal data. This greatly reduces the amount of data to be processed and improves the utilisation efficiency of disks and the network transmission.

The rest of this paper is organised as follows: the mechanism of distributed spatio-temporal index is introduced in Section II, and Section III describes the storage optimisation of the two kinds of indexes for spatio-temporal data. The experimental procedure and the experimental results are reported in Sections IV. Finally, Section V concludes the paper.

2 Distributed spatio-temporal index

2.1 A spatio-temporal data storage model

In order to support efficient spatio-temporal index, we define a novel spatio-temporal data storage model. This model is very convenient for describing both

time and space dimension, and it can reduce the workload of index retrieval and data quering. We define a record Q in the following format:

Q=(Timestamp, Lon(longitude), Lat(latitude), Attr1, Attr2,..., AttrN)

The above format indicates that the record is at a certain time and a specific location. In Q, *Timestamp* represents the timestamp of the record which we also use T to represent later for ease of description, and *(Lon, Lat)* is the spatial location which is generally the longitude and latitude ; *Attr1, Attr2, ..., AttrN* are the other attributes of the record Q. The first three properties of Q are the core components of a spatio-temporal data object. As shown in Figure 1, a cuboid C (A, B) specified by A (t1, lon1, lat1) and B (t2, lon2, lat2) represents a time range between t1 and t2 and a spatial range between (lon1, lat1) and (lon2, lat2), and C is called MBC (the minimum bounding cuboid) which is similar to the minimum bounding rectangle (MBR) [10] in traditional spatial data representation.

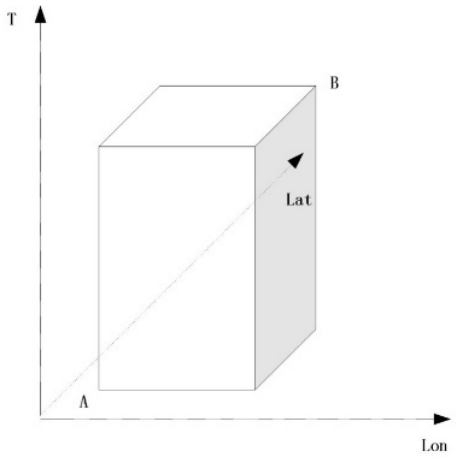

Fig. 1. MBC: The Minimum Bounding Cuboid

2.2 The TGrid algorithm

The traditional grid index partitions data into different subspace units in accordance with different lengths and widths of the space, and different spatial data are partitioned into different cells. But for spatio-temporal data, if the partitioning method is still in this way and in different time periods the degree of data aggregation is different, the data will not be distributed evenly and some nodes may have very heavy load. Considering this, we improve the segmentation

algorithm called spatio-temporal grid algorithm (TGrid), and the segmentation process has five steps, as detailed below.

Step 1 In the division of MBC, divide the time period. In this research, we will set the default size of time to 10s.

Step 2 Calculate the time period of MBC and the number of partitions. We use the following formula:

$$N = \left\lceil \frac{S(1+\alpha)}{B} \right\rceil \tag{1}$$

In the above, S is the total size of spatio-temporal data in the input file; B is the HDFS block size (the default value is 64 MB); α is the load factor, with a default value of 0.2, which makes the actual storage capacity of each data block less than the default size of a HDFS block.

Step 3 Calculate the partition boundaries. According to the number of the partitions N, we will uniformly divide the MBC in spatial range. Each unit is calculated by considering the space coordinates which are the coordinates of lower left corner and the coordinate of upper right corner.

Step 4 If there are still some space to be partitioned, return to Step 2; otherwise, go to Step 5.

Step 5 According to the partition method of spatio-temporal grids, store the spatio-temporal data into the corresponding partitions.

In the division process, some of the data grid cells may be larger than the size of the data blocks. So we need to set a threshold (a little less than 64 MB) during the actual division. When the data are beyond the threshold limit, the data will be assigned to a new block.

After obtaining the number of blocks, the next step is to start a MapReduce assignment. Then, the spatio-temporal data will be divided within the scope of each MBC grid into different partitions, and an index of one-dimensional time is created during the reduce phase. Finally, the results will be output into the HDFS, and the local indexes are put together to the master node to constitute a global index.

Because the grid index is a plane index, the contents of the grid cells in this index are stored disorderly. However, we will use one dimensional time index in the data block, essentially, which divides the minimum bounding rectangles of the data block into small MBCs and marks each MBC according to the time dimension. The format of storage is shown in Table 1. In this table, (T1, T2) is the time interval and (T1, T2) is marked C which represents the offset value of the data block in the MBC, such as (201402070223122, 3.21, 12334.34, C1), which represents the location of the spatio-temporal data at time 201402070223. We will introduce the methods of storing the spatio-temporal data in the next section.

2.3 The QaDTree algorithm

In order to make uniform distribution of the spatial data and ensure the node load balance, we improve the quadtree algorithm [11] and design a bet-

Table 1. One-dimensional Time Index And Data Storage Structure

Temporal interval (T1,T2)	Offset value
(201402030112, 201403050223)	C1
(201403050223, 201404050645)	C2
.....	.....

ter partitioning strategy for spatio-temporal data, and we term this algorithm the QaDTree algorithm, which generates the global index using the improved quadtree algorithm to partition the spatio-temporal data and the local indexes using 3DR-tree index to manage the spatio-temporal data [12]. The tree non-leaf node of the global index contains four children, each of which represents a sub-space area. Then each child space area is divided into four again and this is done recursively until the end of the division. The leaf node points to the local index tree. In order to partition efficiently in large-scale spatio-temporal data, we run a MapReduce assignment and build the quadtree according to the following five steps.

Step 1 According to different time intervals, the MBCs of spatio-temporal data are divided and the total size of MBCs is recorded.

Step 2 If the size of the spatio-temporal objects is lower than the threshold value (the default value is 60MB) in each MBC, a partition is output directly.

Step 3 We divide data into smaller MBCs in accordance with the extent of space. If in an MBC the total size of the spatio-temporal objects is below the threshold value (the default value is 60MB), all the data in the MBC are output to the same partition. And we continue to divide the MBC region into four smaller MBCs uniformly.

Step 4 Divide each child MBC as in Step 3 until all the data in the MBC meet the divided value.

Step 5 Repeat Step 2 until all the spatio-temporal data objects have been processed.

The size (60MB) which we set is smaller than the default size of HDFS block (64MB) so that we can have enough room in the data block to store the local index file. But the dividing value cannot be too small because it will cause the time of dividing spatio-temporal data to be long, and it will also increase the number of data blocks and decrease the retrieval efficiency. Through the above steps, the data divided are physically adjacent in the HDFS blocks of the same database, and at the same time the distribution of data is more uniform than others.

Finally, we construct local indexes for each block of the data file. Once the job of MapReduce is completed, we initialise the HDFS command and all the local indexes are written to a file and this file is the global index. The Master node then records this global index. The global index is constructed by the bulk

load and is stored in the memory of the master node. Once the master node fails or restarts, global indexes can be recreated through the local indexes.

In order to improve the throughput of I/O and reduce the limit of I/O and network bandwidth, it is very important to use a good index technology and store the data which are physically adjacent and improve the retrieval efficiency. Multi-dimensional indexing technology can provide technical support for the spatio-temporal index and the time serves as another dimension of space objects. 3DR-tree index can query historical data effectively and it is very suitable for one-time writing and reading a lot of spatio-temporal data. The index structure is simple and it is superior to the other index structures when dealing with a long time period of the query. In the job of MapReduce, spatio-temporal data are divided into different partitions after the Map tasks, and then we start to build the local 3DR-tree index in each partition in the Reduce task. The leaf node of the index is the MBC and the non-leaf node is a rectangle containing all the children nodes. At the end, we construct a tree structure which is similar to the 3DR-tree.

2.4 The storage structure of spatio-temporal data

The column-based database is different from the traditional relational database which stores data in row. In the table of a typical relational database, each row contains a record of field values, so every piece of spatio-temporal data sequence is stored together. However, the column-based database will store a column with the same attribute. In this research we use the column storage, which has the following advantages: (1) we query historical spatio-temporal data, so we slightly modify the data. Although it is time consuming to store spatio-temporal data by splitting the records of each row, the retrieval is more efficient than others and the data redundancy is reduced. (2) Due to the fact that the data attribute stored in each column is the same, it is highly beneficial for data analysis. Compared to the storage in row, we need to analyse multiple attributes of each record. (3) Each column attribute is the same and the compression algorithm is more efficient.

Each column has similar properties. If we want to find certain spatio-temporal data, such as Id, we can just scan the Id column. And it also makes the compression ratio very high. It can use less space for storage to reduce the disk I/O requirements, and at the same time it has more advantages in terms of the network transmission. Similar to RCFile [13], in the spatio-temporal grid index, we will store the data of the same period of time in column and will store the data of the different periods of time in row. In the QaDTree index, we will write the MBC data of leaf nodes according to the column into data block. Each MBC writes into the file according to the column and each MBC is stored in normal order.

We add the gzip data compression method when constructing a column to store and write to a file. This data compression algorithm has the following advantages:

(1) The HDFS stores spatio-temporal data files by compressing data, which can reduce the storage space.

(2) The communication between a Hadoop cluster is dependent on the network. We can transmit the compressed data to improve the network traffics.

(3) Based on spatio-temporal data storage, the compression effect is better.

We start the MapReduce and need to partition the data which will be queried in HDFS when we query spatio-temporal data. Due to the fact that the type of data which are stored based on column is consistent and the characteristics are similar, it can efficiently compress data. When the data in storage are stable or change a little, the compression of data can greatly reduce the storage space. In this paper, we use the generic gzip compression algorithm. After storage, all the data use the gzip compression algorithm.

3 Experimental Results

In the experiments, we adopt fifteen ARM-based Cubieboards2 [14] development boards to set up the Hadoop cloud computing platform. Cubieboard2 is developed by the Zhuhai Cubietech team. It adopts the A20 processor development board and runs on Android, Ubuntu, and other Linux systems. Cubieboard2 adopts the CortexTM - A7 dual-core processor and the memory of the board is 1 GB, so its function is more powerful. This cluster has Hadoop 1.2.1 deployed on it and running on Lubuntu. Unless mentioned otherwise, we use the data set based on the format of the raw GPS taxi data in a region. We design the process which generates simulation data: we generate 100GB simulated GPS spatio-temporal data randomly. We then perform the data pre-processing according to the definition of the spatio-temporal data model.

3.1 Storage performance

Figure 2 shows the size of the data after using different storage methods for the 2GB GPS data. In this figure, the size is the sum of the data size and index size. The spatio-temporal grid construction method has used 0.641GB storage space for the 2GB data. We take the storage solution which is based on column and use the compression algorithm to compress data. This greatly reduces the storage space. The size of QaDRtree index is bigger than the size of grid index, so the storage space of QaDRtree index is relatively larger but there is no much difference. Compared with the storage methods, the size of SequenceFile is about 5 to 6 times larger than the size of our storage solution. While compared with the traditional PostGIS database, the cost of space for storing 2GB spatio-temporal data is 10 to 12 times higher than the column-based database. So the advantages of our storage solution are very obvious. The storage space of 100GB spatio-temporal data is about 31.3GB and the compression ratio is 31.3%. So our solution has great advantages on saving storage space when dealing with the huge amounts of data.

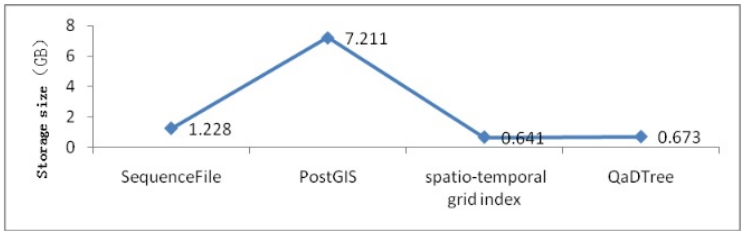

Fig. 2. The Comparison of Different Storage Solutions in Terms of Storage Used

3.2 The time for spatio-temporal data index generation

Figure 3 shows the analysis of two index structures. The data have been generated in advance. The construction process of the index is as follow: (1) read the data from the HDFS, (2) perform one-time construction after executing two MapReduce tasks. The index can be read many times to retrieve the spatio-temporal data contents. As data ranges from 1GB to 20GB, the construction time for the indexes increases with the increase of data size. Due to the hardware limitation, such as CPU, memory, and network card performance, we use the compression mechanism in network transmission of the clusters. So the time of disk I/O transmission and network transmission is greatly reduced when the index is built and the data are retrieved. The construction method of the spatio-temporal grid index is simple and the cost of time is less than that of constructing the QaDTree index. However, with the increase of the amount of data, the number of map tasks and reduce tasks which execute concurrently also increase in the MapReduce tasks and thus the efficiency of the index building is improved.

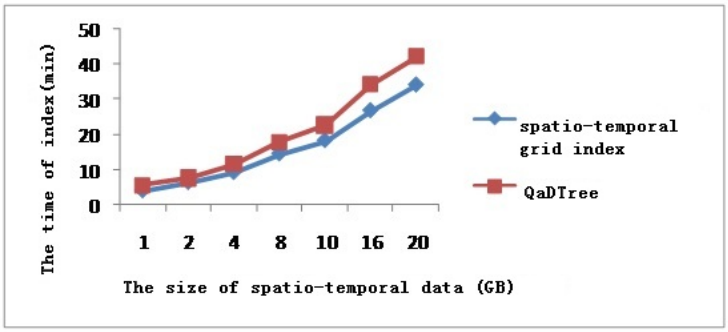

Fig. 3. The Time Used for Index Construction of Different Data Sizes

3.3 Spatio-temporal data retrieval performance

In order to evaluate the retrieval performance of the indexes, we use the simulated taxi data of 100GB and select multiple retrieval scopes of space and time range to carry out a series of experiments. In Figure 4, we retrieve different scopes of space within a fixed time 100s. The horizontal space area increases from 0.001% to 1% of the total space with respect to all the data. In terms of the spatial retrieval, the performance of QaDTree is better than taht of the spatio-temporal grid index. But with the decrease of the search space, the performance of these two kinds of retrievals is almost unanimous. In Figure 5, the x-axis represents that the time range that we retrieve data from increases from 0.01% to 0.2% of the whole range of time spanned by all data. We found that when the time interval was small, the retrieval performance of using spatio-temporal grid index is better than that of using the QaDTree index. This is because the spatio-temporal grid index prunes the time more efficiently and the performance is higher. But with the increase of the time interval and the number of spatio-temporal objects, we use the spatio-temporal grid index to retrieval the spatial attributes is slowly. In contrast, QaDTree is more applicable in retrieving data from a bigger time interval.

Figure 6 demonstrates the scalability of our cloud computing platform and the stability of the index. The number of cluster nodes increases from 15 to 30 and the scope of retrieval is 1% of the total time spanned by all data and 0.1% of the total spaces occupied by all data. With the increase of the number of nodes, the performance of the system in terms of the time used for retrieval is improved significantly, especially under the cloud computing platform. Because the performance of single node is too low and the processing ability is limited, we need to share the MapReduce tasks among multiple nodes.

4 Conclusion and Future Work

In this research, we proposed a novel storage and retrieval solution for spatio-temporal data based on the ARM development boards, which are cost effective and green. In this solution we use one-dimensional time index and 3DR-tree index. The former can quickly locate the query time interval, and the latter is based on the R-tree. Thus we proposed the concept of MBC which is based on the minimum bounding rectangular (MBR). Considering the fact that the hardware performance of the cloud computing platform we used is low and the network transmission speed is slow, in order to adapt to such a platform we use the column storage and a compression algorithm to deal with the spatio-temporal data. This greatly reduces the amount of data to be processed and thus improves the utilization efficiency of disk and the transmission efficiency of networks.

In future work, our research can be improved in the following three aspects: (1) we will further improve the temporal data retrieval functions. (2) We need to further strengthen the spatio-temporal indexing learning. In addition, we will learn the latest technology, such as Storm and Spark. (3) We can study the

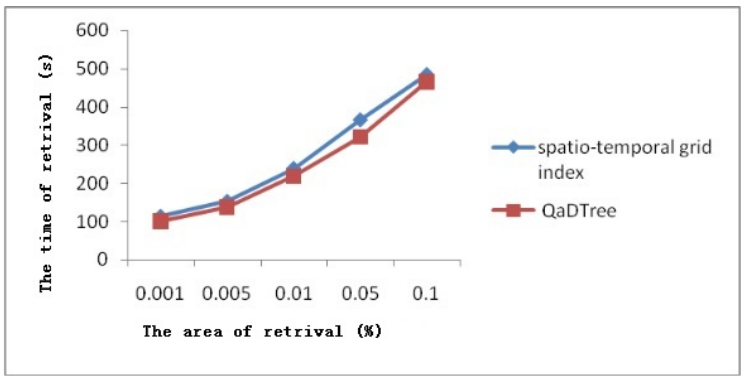

Fig. 4. The Time Used for Retrieval under Different Space Areas with the Same Fixed Time Interval

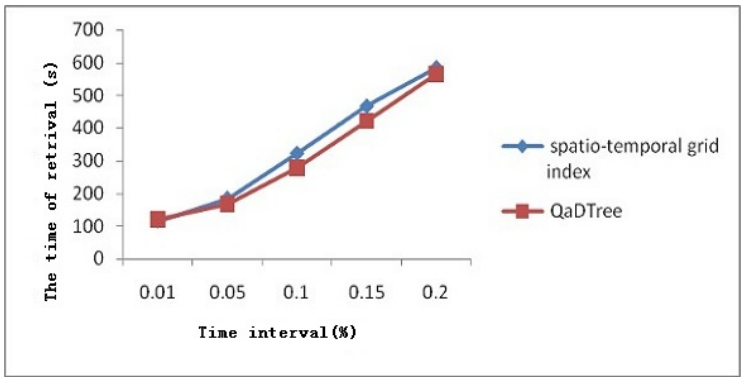

Fig. 5. The Time Used for Retrieval under Different Time Intervals with the Same Space Area

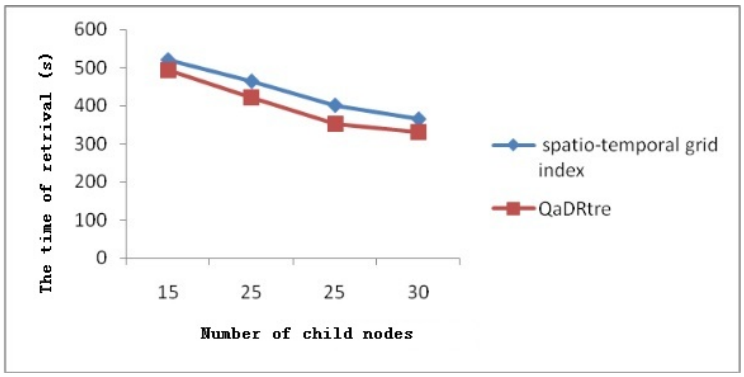

Fig. 6. The Scalability Of The Cloud Computing Platform

visualization modules of spatio-temporal data, and analyse the spatio-temporal data distribution. We will work on the further improvement of the distributed spatio-temporal index. In addition, we will aim to mine more valuable information through the use of various query algorithms.

Acknowledgements

This project is supported by Science and Technology Development Plan of Jilin Province (20140204010SF) and Chinese National Natural Science Foundation (61472159). WP is supported by the PECE bursary from The Scottish Informaticsc and Computer Science Alliance(SICSA).

References

1. Zickuhr K, " Three-quarters of smartphone owners use location-based services ", Pew Internet and American Life Project, 2012.
2. Dhar S, Varshney U, " Challenges and business models for mobile location based services and advertising ", Communications of the ACM, 2011, 54(5). 121-128.
3. Cattell R, "Scalable SQL and NoSQL data stores ", ACM SIGMOD Record, 2011, 39(4). 12-27.
4. Botea V, Mallett D, Nascimento M A, et al. " PIST: an efficient and practicalindexing technique for historical spatio-temporal point data ", GeoInformatica, 2008, 12(2). 143-168.
5. Cudre-Mauroux P, Wu E, Madden S, " Trajstore: An adaptive storage system for very large trajectory data sets ", Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010. 109-120.
6. Guttman A, " R-trees: A Dynamic Index Structure for Spatial Searching ", Proceedings of Meeting, Boston, Massachusetts, June 1984. 47-57.
7. Schneider R, Seeger B, Beckmann N, et al. " The R*-tree: an efficient and robustaccess method for points and rectangles ", Proc. ACM SIGMOD Symposium on Principles of Database Systems, 1990. 322-331.
8. Singh S, Mayfield C, Prabhakar S, et al. " Indexing uncertain categoricaldata ", Data Engineering, 2007. ICDE 2007. IEEE 23rd InternationalConference on. IEEE, 2007. 616-625.
9. Apache Hadoop, http://wiki.apache.org/hadoop/last accessed 28.11.2015.
10. Eldawy A, Mokbel M F, " SpatialHadoop: A MapReduce framework for spatialdata ", Proceedings of the IEEE International Conference on Data Engineering(ICDE'15). IEEE. 2015. 1352-1363.
11. Eldawy A, Alarabi L, Mokbel M F. "Spatial partitioning techniques in SpatialHadoop", Proceedings of the Vldb Endowment, 2015, 8(12). 1602-1605.
12. Gong J, Zhu Q, Zhong R, et al. " An efficient point cloud management method based on a 3D R-tree ", Photogrammetric Engineering and Remote Sensing, 2012, 78(4). 373-381.
13. He Y, Lee R, Huai Y, et al. " RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems ", Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011. 1199-1208.
14. http://docs.cubieboard.org/products/last accessed 30.3.2015.