Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 1059231, 11 pages
https://doi.org/10.1155/2018/1059231

Research Article

A Novel Query Method for Spatial Data in Mobile Cloud Computing Environment

Guangsheng Chen,[1,2] Pei Nie ,[1,2] and Weipeng Jing [1,2]

[1] *College of Information and Computer Engineering, Northeast Forestry University, Harbin 150040, China*
[2] *Heilongjiang Province Engineering Technology Research Center for Forestry Ecological Big Data Storage and High Performance (Cloud) Computing, Harbin 150040, China*

Correspondence should be addressed to Pei Nie; 15546012870@163.com

Received 25 January 2018; Revised 5 April 2018; Accepted 17 April 2018; Published 17 May 2018

Academic Editor: Kai Yang

Copyright © 2018 Guangsheng Chen et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the development of network communication, a 1000-fold increase in traffic demand from 4G to 5G, it is critical to provide efficient and fast spatial data access interface for applications in mobile environment. In view of the low I/O efficiency and high latency of existing methods, this paper presents a memory-based spatial data query method that uses the distributed memory file system Alluxio to store data and build a two-level index based on the Alluxio key-value structure; moreover, it aims to solve the problem of low efficiency of traditional method; according to the characteristics of Spark computing framework, a data input format for spatial data query is proposed, which can selectively read the file data and reduce the data I/O. The comparative experiments show that the memory-based file system Alluxio has better I/O performance than the disk file system; compared with the traditional distributed query method, the method we proposed reduces the retrieval time greatly.

1. Introduction

Under the background of the explosive growth of mobile data traffic and the emergence of various new business scenarios, the fifth-generation (5G) mobile communication network is proposed and becoming a hot topic in academical and industrial field. As a new generation of wireless mobile communication network, 5G is mainly used to meet the demand of mobile communication after 2020; driven by the rapid development of mobile Internet and growing demand for Internet of Things (IoT) services, 5G is required to have the features of low cost, low power consumption, and being safe and reliable [1, 2]; 5G will enable information and communication to exceed the time and space constraints, greatly shorten the distance between people and things, and quickly realize the interoperability of human and all things [3].

At present, the key technology of 5G network is still in the research phase; in addition to network architecture and transmission theory, mobile cloud computing is also one of the focuses of the future 5G network research [4]. Mobile cloud computing is a new model of delivery and usage of IT resources or information services; it is a product of cloud computing in the mobile Internet; mobile smart terminals in mobile networks are connected to remote service providers in an on-demand and scalable way to obtain the necessary resources, mainly including infrastructure, computing, storage capacity, and application resources [5, 6]. With the constant penetration of information technology into society, location-based services have been widely used in many fields such as military and transportation, for example, patient care in smart home and navigation service for mobile transportation; on the one hand, the positioning technology continues to evolve, providing increasingly accurate location information for mobile applications; on the other hand, with the increase in the number of users and the large increase in the number of mobile applications, the explosion of spatial data has brought tremendous pressure to upper-layer applications, especially for applications in mobile networks with huge traffic. Therefore, more and more researches combine mobile cloud computing with spatial data processing and use cloud platforms for data storage, calculation, indexing, and

querying to speed up processing and reduce response delay [7].

In this paper, we aim to provide a fast spatial data query interface in mobile cloud computing environment; for the reliable storage of massive spatial data, we first mesh the spatial data, fill the entire grid space by Hilbert curve, and then organize the data block using distributed memory file system Alluxio's KeyValueStore file and build a two-level index based on the file's internal index and file name. In order to provide real-time query, we use Spark, a distributed memory computing framework, to query for distributed data; in the meantime, we propose a Spark data input format based on the characteristic of Spark. The method eliminates disk I/O as much as possible and the entire query process from memory to memory and filtering through two layers. Experiments show that this method can well organize massive spatial data and has higher performance than traditional query methods.

The rest paper is organized as follows: Section 2 reviews the related work and in Section 3 we provide a background on spatial data queries and overview of Alluxio. Section 4 describes our proposed data indexing algorithm, storage structure, and parallel distributed retrieval algorithm. We discuss the experiment results in Section 5 and conclude in Section 6.

2. Related Work

Spatial data is a quantitative description of the geographical location of the world; based on computer technology, efficient use of spatial data in people's lives is of great significance. Spatial database emerged in the background of the rapid development of database and the rapid development of space application demand; it provides spatial data type definition interface and query language, which has very important pioneering significance in the early stage of GIS [8, 9]. However, with the rapid growth of spatial data, the traditional spatial database has been unable to meet the needs of real-time retrieval. Some works [10, 11] aimed at large-scale data query of spatial data and propose a parallel spatial database solution that distributes the data load and retrieval pressure of single computers to multiple servers. However, this method requires very expensive software licenses and dedicated hardware and requires complicated debugging and maintenance work [12].

As cloud computing has become a cost-effective and promising solution to computational and data-intensive issues, it is quite natural to integrate cloud computing into spatial data storage and processing. Wei et al. [13] applied the distributed NoSql database to the storage of spatial data and constructed efficient index to quickly retrieve spatial data. As MapReduce has become the standard solution for massively parallel data processing, more and more researchers apply Hadoop to GIS. Puri et al. [14] propose a MapReduce algorithm for distributed polygon retrieval based on Hadoop. Ji et al. [15] present a MapReduce-based method that constructs inverted grid index and processes k-NN query over massive spatial data sets. Hadoop-GIS [16] and Spatial-Hadoop [17] are two scalable and high-performance spatial data processing systems for running large-scale spatial queries in Hadoop. However, due to the limitations of MapReduce's own design, some researchers began to migrate spatial data processing to Spark. Wang et al. [18] use Spark for spatial range query; all the spatial data are stored in HDFS and propose a grid indexing method called Spark-Fat-Thin-Grid-Index. Cahsai et al. [19] propose a novel approach to process k-NN queries; this approach is based on a coordinator-based distributed query processing algorithm and uses Spark for data-parallel processing. At the system level, Yu et al. [20] introduce an in-memory cluster computing framework for processing large-scale spatial data, which efficiently executes spatial query processing algorithms and provides geometrical operations library that accesses Spatial RDDs to perform basic geometrical operations.

Figure 1: Range query.

The above researches have eased the contradiction between the current rapidly increasing data set and real-time retrieval. However, there are still some performance bottlenecks in current approaches. First of all, the data is stored on disk, and the query is based on memory; the mismatch between memory processing speed and I/O speed restricts the performance. Secondly, the existing distributed search algorithm does not distinguish data strictly while readings, so many data that are not related to the query conditions are also read into the memory, and the query load of the calculation layer is increased. Therefore, in this paper, we store the data based on the memory file system and build a two-layer index structure to accelerate random access; in view of the lack of current work, Spark is used for parallel data processing and a data input format for spatial query is proposed, which filters out the irrelevant data with the query conditions based on the index structure.

3. Background

In this section, we provide the required background and preliminaries about the spatial data queries and a brief overview of distributed memory filesystem Alluxio.

3.1. Spatial Data Queries. For most applications, there are two common approaches to geospatial query. As shown in Figures 1 and 2, Figure 1 is range query [21]; given a set of data points P and a spatial range R, query aims to retrieve all spatial points within the given R boundary; the result of the range query is $\{P1, P2, P3, P4\}$. Figure 2 is K-NN query [22]; the nearest

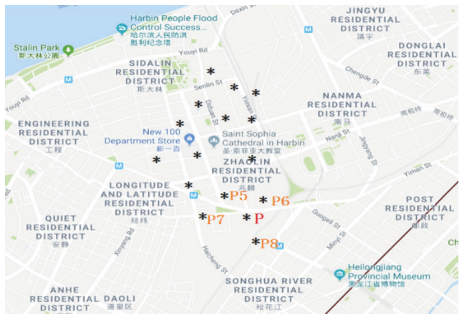


Figure 2: K-NN query.

neighbor query is the most common query in geography; it is another kind of query method different from range query. It is used to find out the nearest neighbor in space from a given point; the nearest neighbor can be one or more; as shown in Figure 2, the given point is P marked by the red color and $k = 4$, 4-NN query here is to search for four points nearest to P, and the search result of this query is $\{P5, P6, P7, P8\}$.

3.2. Alluxio Overview. Alluxio (former Tachyon) is a new project that was unveiled by UC Berkeley AMPLab Labs in 2013 as the world's first memory-centric virtual distributed storage system that unifies data access and becomes the key to connecting computing infrastructure and underlying storage; Alluxio's memory-centric architecture enables data access faster than existing solutions. In essence, Alluxio is a distributed memory file system deployed on computing platforms such as Spark or MapReduce and storage platforms such as HDFS or S3 and by globally isolating computing platforms and storage platforms from alleviating the memory pressure in the computing framework, also given the ability to quickly read and write large amounts of data memory framework computing. Alluxio separates the power of memory storage from Spark/MapReduce, allowing Spark/MapReduce to focus more on computing itself for greater execution efficiency through finer partitioning [23, 24].

Alluxio architecture shown in Figure 3, which uses a standard Master-Worker mode, running Alluxio system, consists of a Master and multiple Workers and Alluxio Master support ZooKeeper for fault tolerance, used to manage the metadata of all files; it is also responsible for monitoring the status of individual Alluxio Workers. Each Alluxio Worker starts a daemon and manages a local Ramdisk, and Ramdisk stores specific file data. So far, Alluxio has been updated to 1.7.1.

4. Parallel Spatial Data Retrieval Based on Memory

In this section, because the spatial data indexing and storage are closely related to retrieval, we first introduce the spatial indexing algorithm in this paper and illustrate the data storage method and a two-level index structure in Alluxio. After that, we introduce the specific spatial data-parallel retrieval algorithm.

4.1. Indexing Spatial Data. Geographic space usually includes three kinds of vector data, namely, points, lines, and polygons, as shown in Figure 4. Faced with massive spatial data, users often care about some local information; therefore, how to index spatial data and respond quickly to user's requests is a key issue when storing. Common spatial indexing methods include gridding, KD Tree, R-tree, quad-tree index, etc. [25–28]; among them, grid index has the characteristics of simplicity and convenience; compared with other methods, the construction of grid index in Spark/MapReduce parallel system is less complicated. So in this paper we build the grid index for spatial data.

In order to construct the grid index, the geographic space needs to be divided into different grids; different data blocks contain the data intersecting with the geographic location of this grid. For uniquely identifying each data block and taking into account the spatial proximity of the data, the data block is coded by using the Hilbert curve [29, 30]; the original space area is block-coded as shown in Figure 5.

After meshing the geospatial space, we need to locate the vector data to a grid block and give its grid ID. As shown in Figure 5, the coordinates of lower-left and top-right corner of the space plane are (Lon0, Lat0) and (Lon1, Lat1), respectively; as parameters, the width and height of a grid are w and h, respectively, so the number of rows and the number of columns can be calculated as follows:

$$\begin{aligned} \text{cols} &= \frac{\text{Lon1} - \text{Lon0}}{w} \\ \text{rows} &= \frac{\text{Lat1} - \text{Lat0}}{h}. \end{aligned} \tag{1}$$

For each spatial data, we need to extract its location information, locate it to a specific grid, and then organize the data within the grid together to speed up the query based on the index. This paper abstracts all the spatial data to a point, and for polygon we have its inscribed center point as its position, and for the line we have its midpoint as its position; for vector point, its position point is itself. The point (x, y) is the spatial location of the data; according to the location points and cols and rows, we can get the col and row for any spatial data; finally use the Hilbert curve algorithm to get the grid ID.

$$\begin{aligned} \text{row} &= \frac{(y - \text{lat0})}{h} \quad 0 \le \text{row} \le \text{rows} \\ \text{col} &= \frac{(x - \text{lon0})}{w} \quad 0 \le \text{col} \le \text{cols} \\ \text{gridID} &= \text{HilbertCurve}\,(\text{row}, \text{col}) \\ &= \text{HilbertCurve}\left(\frac{y - \text{lat0}}{h}, \frac{x - \text{lon0}}{w}\right). \end{aligned} \tag{2}$$

After the above process, each spatial data has its grid ID; the data with the same ID are organized together to form a data table as shown in Table 1.

4.2. Data Storage Structure. Alluxio is a memory-based distributed file system that has many similarities with HDFS [31].

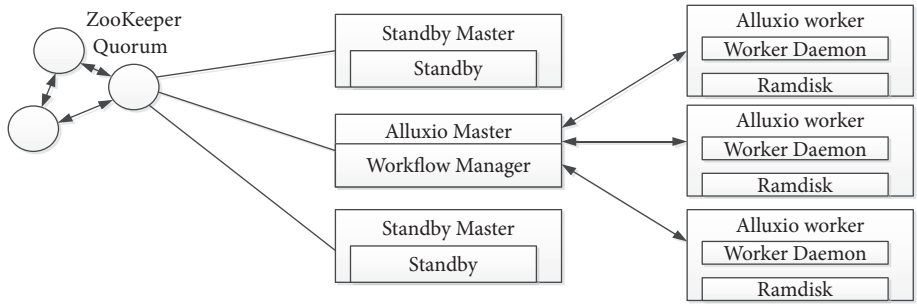


Figure 3: Alluxio architecture.

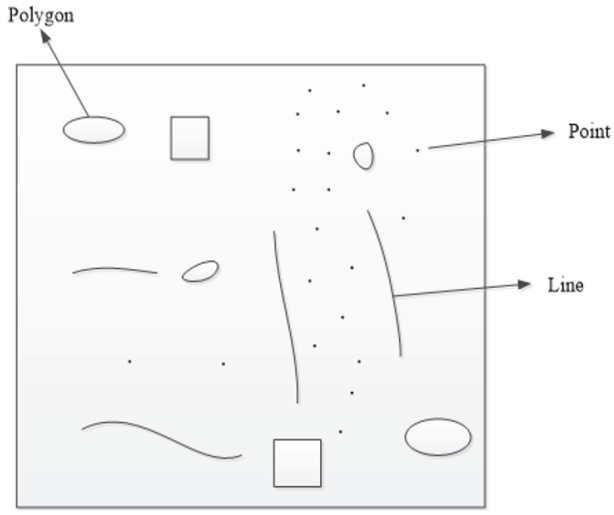


Figure 4: Space area.

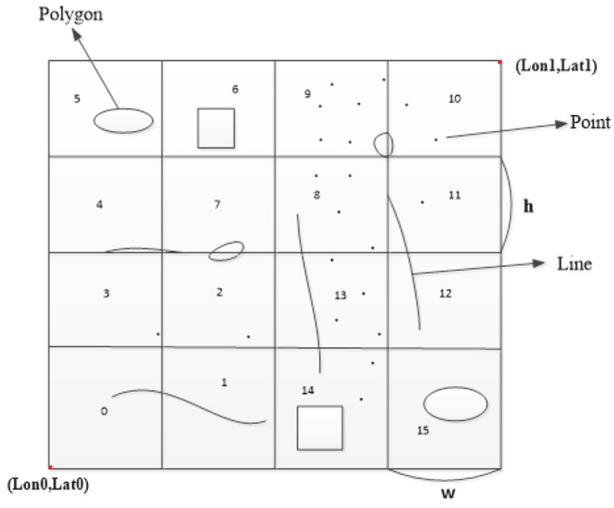


Figure 5: Grid index partitioning and coding.

Usually, data block tends to be smaller than file system block; if data blocks are directly stored on the file system, with the increase of data blocks, this will bring huge metadata storage pressure to the master node and small file problem [32]. Based on this problem, some researches use the key-value pair to structure data block together, solving the problem of small files [33, 34]. Therefore, this paper selected Alluxio built-in file structure KeyValueStore for data storage, the structure of the file in the form of key-value pairs, with the grid's Hilbert curve coding as the key; the corresponding data is stored as the value; KeyValueStore forces the key-value pairs to be written in ascending order because the internal index is built based on the key as the data is written.

Table 1: Data table.

Grid ID	Data List
1	Polygon 1, Line 2, Point 3, ...
2	Polygon 5, Polygon 7, ...
...	...
n	Polygon N, Line M, Point I, ...

This paper sets the size of each KeyValueStore file equal to the Alluxio block size, which is designed to optimize Alluxio storage and to avoid the problems of small files while improving the performance of file retrieval. Each filename contains the data range in the file, such as a file containing grid data with Hilbert code (0, 1, 2, 3); its filename is "0-3.kv," so that all the filename constitutes a global index. The internal index of all the KeyValueStore files and global index form a two-level index, as shown in Figure 6.

The shape of the vector data is irregular and there are many vector data in a single grid space. Therefore, it is very complex to determine the relationship between query conditions and data. So in this paper we introduce the thin-MBR and fat-MBR for the vector data in a single grid space [11]. Taking the data block with grid ID 14 as an example, the thin-MBR establishment process is as follows.

Step 1. Extract the center points of the vectors intersecting the space of the grid block; for the polygons, the center point is the incircle's center point; the line is its middle point, and the point is itself.

Step 2. Make the smallest circumscribed rectangle of all the vector center points in this grid area.

Step 3. The smallest circumscribed rectangle is the thin-MBR of vectors for the grid.

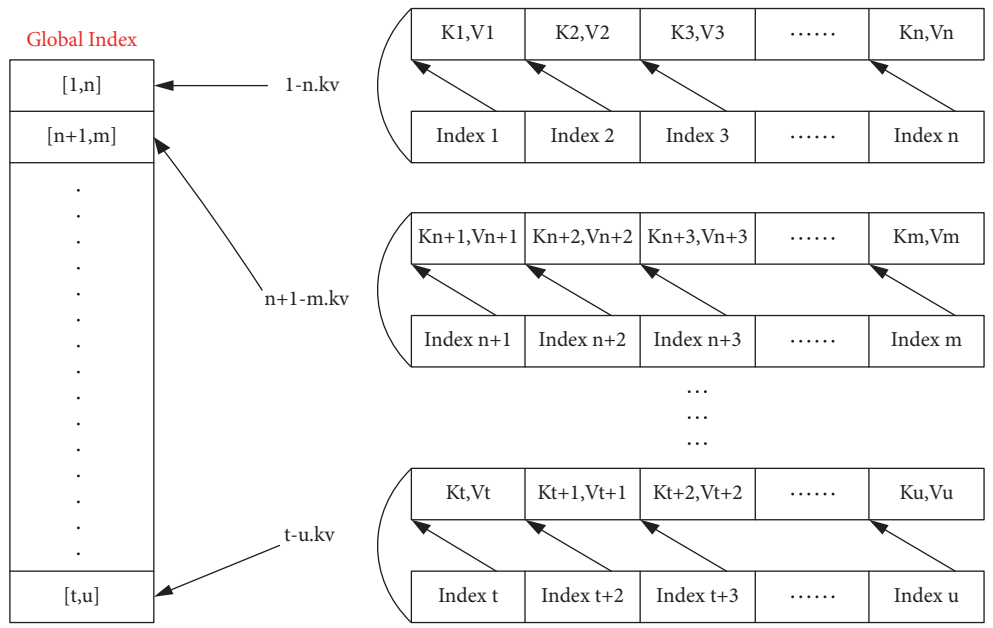


Figure 6: Data structure and two-level index.

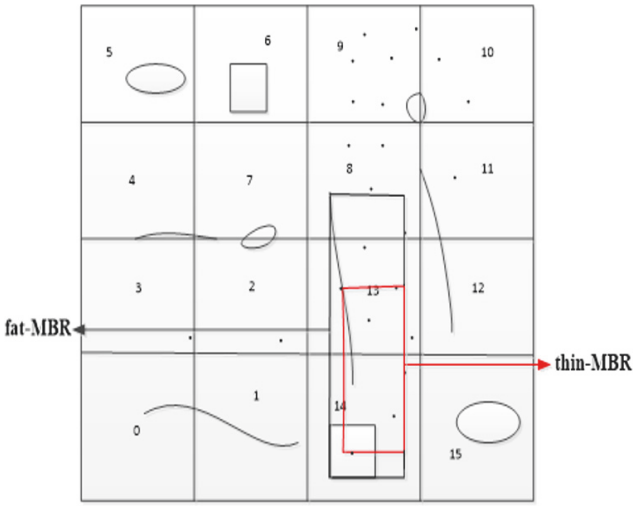


Figure 7: Fat-MBR and thin-MBR representation.

The establishment of fat-MBR is to traverse all the vectors in the space of this block; the minimum enclosing rectangle of all vectors is created, which is the fat-MBR of the space vector data of the grid block.

The thin-MBR and fat-MBR of grid 14 are shown in Figure 7; therefore, key-value pairs should contain more information, the grid code is used as the key, and the value includes the spatial data in the grid and the thin-MBR and the fat-MBR.

4.3. Spatial Data Retrieval. Based on the data-reading characteristics of Spark computing framework, this paper presents an input format called query-KVInputFormat for spatial data query, which filters the data when reading the memory filesystem. After the first layer of filtered data, each Spark task performs data query in memory, which is also called second-layer filtering; two layers of filtering are distributed; the first is to read distributed filesystem data, the second is that the parallelism tasks are handled distributedly.

4.3.1. The First Layer of Filtering. We cover two common distributed queries in Section 3, which can be retrieved after geospatial meshing and coding. For range queries, the range can be transformed into a set of grid numbers spanned by the range. For K-NN queries, record the grid number where the current position is located, such as y, and then record all the grids adjacent to y to form a set of grid numbers. As shown in Figure 8, the grid blocks intersecting the query range are (2, 7, 8, 13), current position is $P2$, and the K-NN query conditions are transformed into grid block group (6, 7, 8, 9, 10, 11). It can be seen that we have narrowed down the data range related to the spatial data query to a set of grid blocks so that we only need to read this set of grid blocks into memory for distributed retrieval and speed up data retrieval.

When interacting with MapReduce/Spark, Alluxio complies with the input and output formats of computational models. However, input formats of big data computing models are designed for batch processing and are read sequentially one by one; each task cannot be randomly read. Therefore, this paper designs a spatial-oriented query input format: query-KVInputFormat—this input format is based on KeyValueInputFormat; the implementation details are as follows.

Overrides protected List ⟨FileStatus⟩ listStatus (JobContext job): this method is used to filter out key-value files that do not contain data to be queried by using grid block groups, such as range queries that transform into grid block group (x, y), where $1 < x < n, n+1 < y < m$, building a global index in memory by traversing the filename, as shown in Figure 6 and

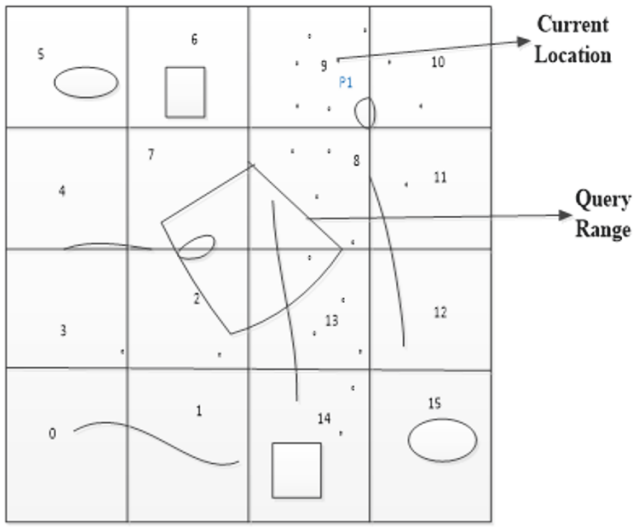


Figure 8: The grid block group representation.

retrieving the eligible files based on the global index, which returns $[1 - n.\text{kv}, n + 1 - m.\text{kv}]$.

Defined class query-KVRecordReader extends RecordReader $\langle K, V \rangle$; this class is internal class; the class defines the way of task read, in the traditional RecordReader class and its subclasses, and the order of each task to read records sequentially. But in the query we need to filter out some records; the sequential read method is no longer applicable, so rewrite the class; each task uses the block group as the query conditions to read assigned file (the file here is filtered after listStatus). Firstly, load the internal index of KeyValuetore into memory and use the query conditions for inquiries. If the record in the query condition exists in the file, the corresponding data block will be read out, if not, to the next record, the entire grid block group will be traversed. This process is concurrency on each task.

4.3.2. The Second Layer of Filtering

(1) Range Query. After the first level of retrieval, the data that meets the query is loaded into memory, forming RDDs, assigned to each task. For the range query, the data obtained through the first level query may be redundant; as shown in Figure 8, the query range is transformed into a grid block group, which reduces the complexity of the query but increases the scope of the query; data not in the range is also loaded into the memory. So in this layer we use Spark for further retrieval.

In a grid area, there are various types and styles of vector data; for the range to be queried, according to the conventional search method, traversing the vector data one by one in memory to determine whether there is intersection with the range, this method is quite consuming and unfavorable to extend. In this paper, data is filtered based on thin-MBR and fat-MBR; at first, whether the range contains the thin-MBR of the current grid region is judged; if yes, keep all the data in the area and jump to the next grid area; if no, judge the relationship between the range and the fat-MBR; if they do not intersect, discard the data in the area; otherwise, traverse the vector data in this area to determine the relationship between the vector and the range; if there is intersection, then retain this vector data; otherwise discard the vector. Combined with two filters, range query algorithm is shown in Algorithm 1.

```
Input: query range
Output: all data in the range
(0) Result set S = ϕ
(1) Calculate the grid block group corresponding to
    the query range
(2) for each task of spark do
(3)     Reading KeyValueStore to form RDDs based
        on the grid block group
(4) end for
(5) for each Partition of RDDs do
(6)     Determine the relationship between the
        query range and thin-MBR/fat-MBR
(7)     if the range contains the thin-MBR then
(8)         Add all data of this grid to S
(9)     else
(10)        if the range don't intersect with fat-MBR
then
(11)                Discard this grid
(12)        else
(13)                Traverse the data of this grid and
                    add eligible data to S
(14)        end if
(15)    end if
(16) end for
(17) return S
```

Algorithm 1: Range query.

(2) K-NN Query. At the first layer of filtering, the grid where query point lie and its bordering data block composed of the grid block group, each task reads the data into memory based on the grid block group, forming RDDs. For each task, we maintain a local queue, calculate the distance between each point of RDD-partitions and the query point, and add the nearest K points to the queue. After the task completes the calculation, the local queue is collected to the master node, and the master node recalculates and selects the nearest K vector points as the query results. K-NN query algorithm is shown in Algorithm 2.

5. Experimental Evaluation

In this section, we present the experimental evaluation of our retrieval algorithm. We divide this section into three parts; firstly, we introduce the dataset and the computing environment used in our experimental study; secondly, we evaluate and compare with existing solutions to see how the time is consumed by the query as the query area grows; after that, we compared Alluxio with HDFS. Finally, we evaluate our algorithms on various sizes of Spark cluster to measure the efficacy of our approach across various cluster sizes.

Input: query point
Output: the nearest K points
(0) Result set $\mathbf{S} = \phi$, local queue $\mathbf{Q} = \phi$
(1) Calculate ***the grid block group*** corresponding to the query point
(2) **for** *each Task* **do**
(3) Reading KeyValueStore to form RDDs based on ***the grid block group***
(4) **end for**
(5) **for** each *Partition of RDDs* **do**
(6) Calculate the distance between each point and the query point
(7) Add the nearest K points to **Q**
(8) **end for**
(9) **for** *each* **Q do**
(10) Collected to the master node
(11) **end for**
(12) Pick out the nearest K points add to **S** on master node
(13) **return S**

Algorithm 2: K-NN query.

5.1. Datasets and Experiment Environment. For the experimental dataset, we selected 43200 × 20880 global high-resolution images as the base map and a total of 11 layers' vector data of national borders, coastlines, ports, provinces, lakes, rivers, roads, and airports of all countries in the world as experimental data; the size of a single data grid is 512 × 512, encoded using Hilbert curve; the Alluxio block is set to 64 MB and the single KeyValueStore file is also 64 MB. We conducted our experiment on a cluster of 5 Inspur Yingxin I8000 blade servers, one of which served as the master node and the other four served as compute nodes. Each node was configured as a Xeon E5-2620 v2 6-core 2.10 GHz processor with 32 GB of memory and a 200 GB hard disk drive using Tenda TEG1024G Gigabit Ethernet switch, with Red-Hat 6.2 installed on each node and 2.6.32 Linux kernel, running Spark-1.5.0, Hadoop-2.5.2, and Alluxio-1.6.0.

5.2. Time Cost versus Query Size. We first demonstrate how the time costs grow as the size of the query area changes. In this experiment, we use all datasets as input, the cluster uses 4 compute nodes with 24 cores and 4 cores. For range queries, we randomly generated a polygon region for the query. For K-NN queries, we generate a pair of latitude and longitude coordinates using a random function to locate a point on the map as a query point. In order to compare our results with existing distributed techniques, we chose GeoSpark [17] as the benchmark. As shown in the previous section, we store the data in the Alluxio memory file system and use Spark to read the data for distributed data query.

Figures 9 and 10 show the relationship between the query time cost and the query size in 4 cores' environment, and Figures 11 and 12 show that in 24 cores' environment. For range queries, the size of the query range is measured in terms of acreage, and for K-NN queries, the number of query points is used for measuring. From the figure, we note that as the query size grows larger the time cost generally increases due to more data being processed. From the results, we also deduce that our algorithm is on average 1%–50% faster than the current technology in GeoSpark. As we discussed earlier, in the traditional approach data is stored in HDFS; a large amount of spatial data is directly loaded into the memory,

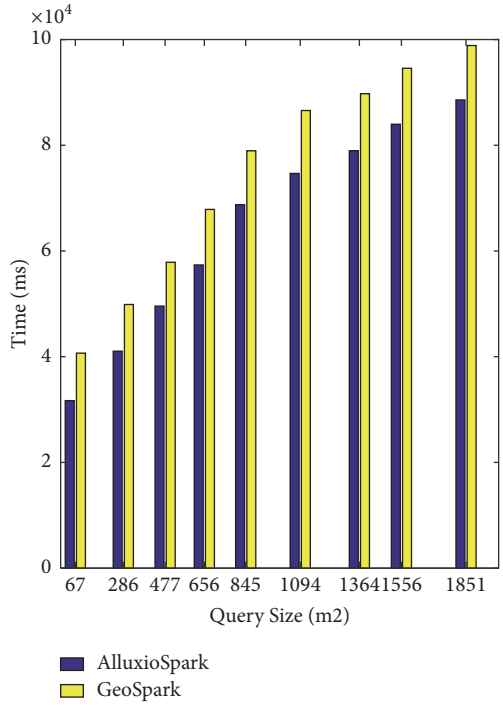


Figure 9: Range query execution time with 4 cores.

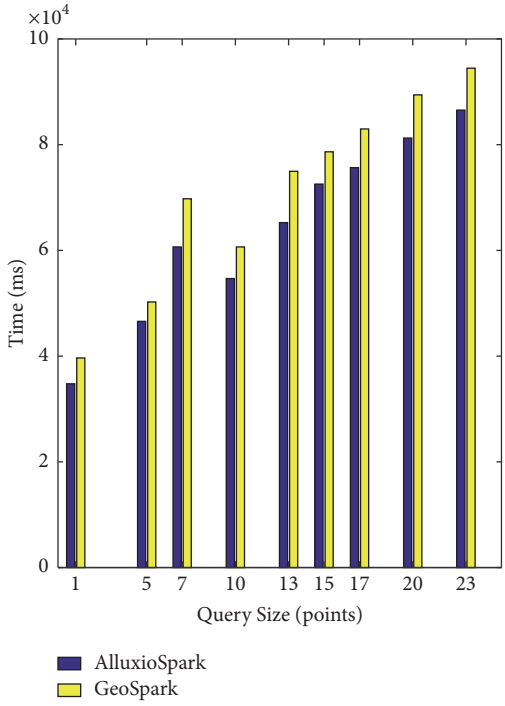


Figure 10: K-NN query execution time with 4 cores.

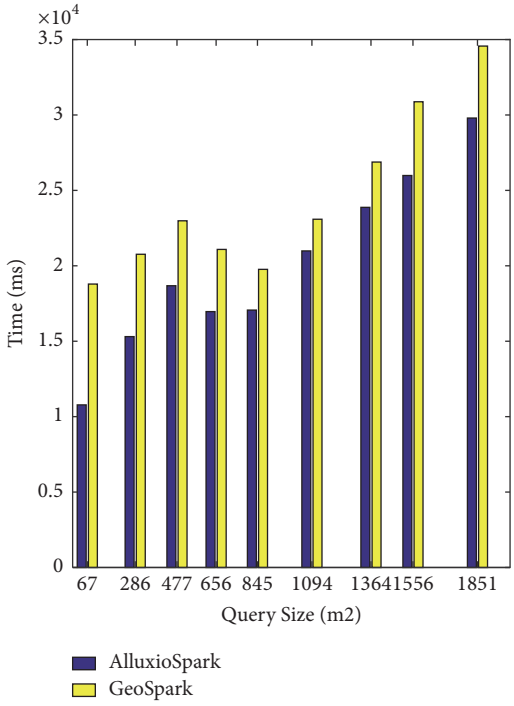


Figure 11: Range query execution time with 24 cores.

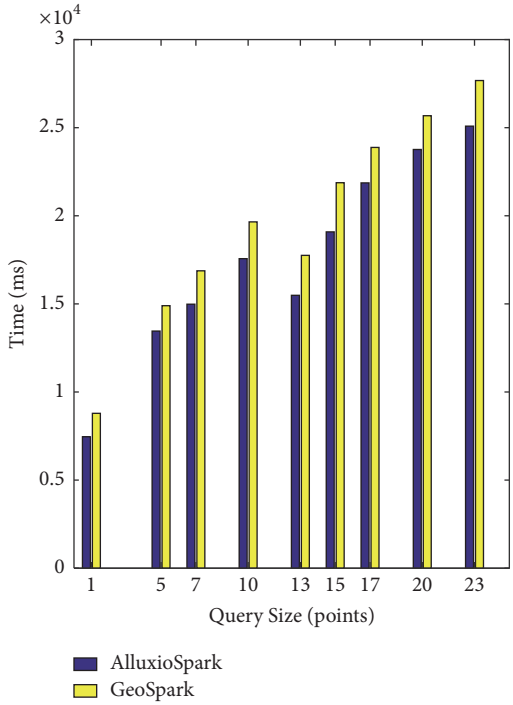


Figure 12: *K*-NN query execution time with 24 cores.

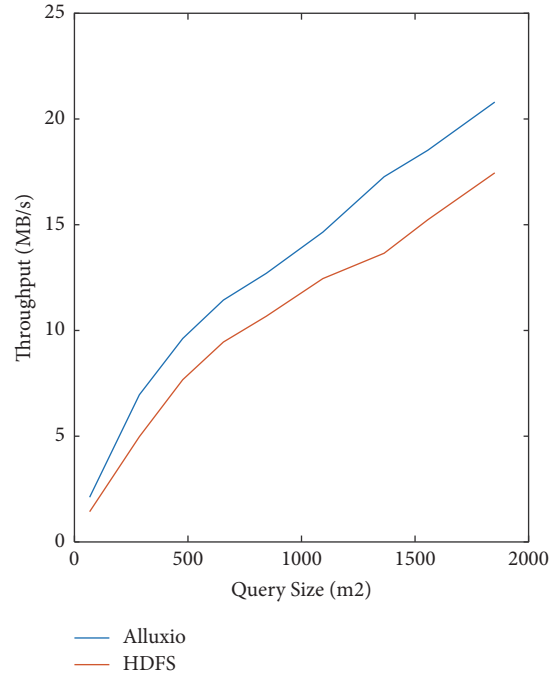


Figure 13: Throughput of range query with 4 cores.

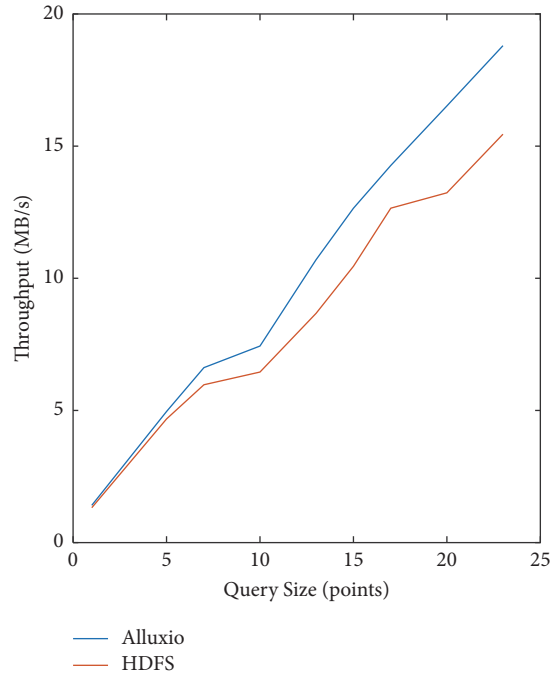


Figure 14: Throughput of *k*-NN query with 4 cores.

resulting in a large amount of data I/O and CPU load, but in the method proposed in this paper data is stored in memory; the entire retrieval process data flows from memory to memory, filtered while reading, making the irrelevant data read as little as possible into memory, reducing the CPU load and speeding up the query.

5.3. Alluxio versus HDFS. The underlying file system used by GeoSpark is HDFS, and Alluxio is used in this paper. We recorded the throughput of file system in experiment 5.2 to evaluate Alluxio's performance. Figures 13 and 14 show the throughput of Alluxio and HDFS in 4 cores' environment, and Figures 15 and 16 show that in 24 cores' environment. From these four figures, we can see that Alluxio has higher

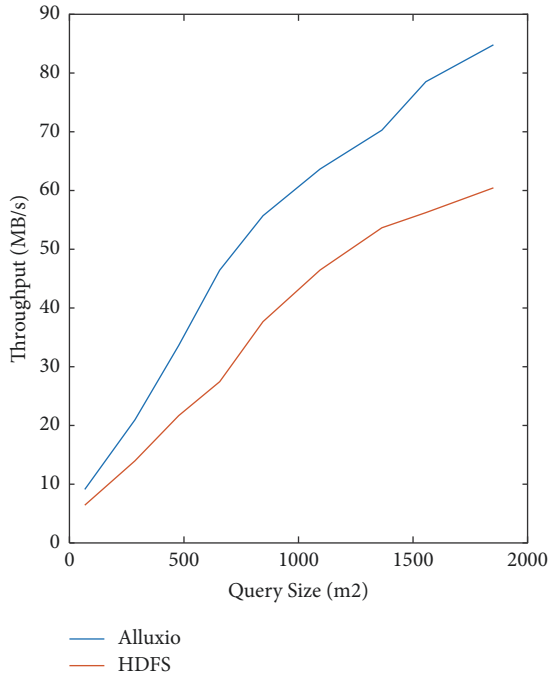


Figure 15: Throughput of range query with 24 cores.

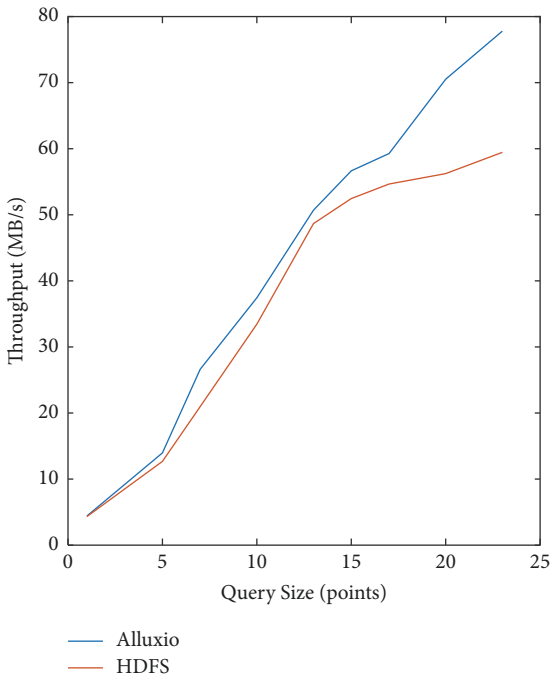


Figure 16: Throughput of k-NN query with 24 cores.

throughput than HDFS in the same experimental conditions, especially when it comes to range query; the main reason for this is that when executing a K-NN query, only the vector point is read into the memory, and the volume of vector point is small, so the difference between Alluxio and HDFS will not be obvious. And through this experimental result, we can see

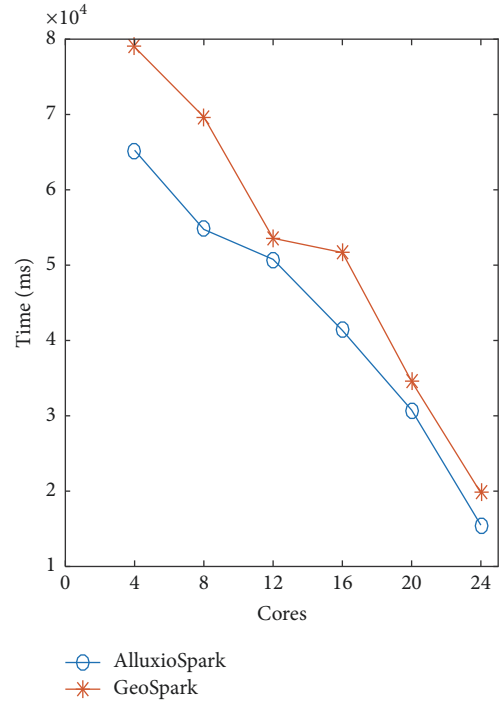


Figure 17: Impact of number of cores for range query.

that Alluxio has higher I/O rate than HDFS, which improves the overall job execution speed.

5.4. Time Cost versus Cluster Size. We next evaluate the effectiveness of our retrieval algorithm by varying the size of the Spark cluster in terms of the number of cores. For this experiment, we generated range query and K-NN query and used them to run queries on different cluster sizes. Figures 17 and 18 show the time cost on various cluster sizes when the range query size is 845 square meters and with K-NN query as 13-NN query. We infer from Figures 17 and 18 that the execution time decreases gradually as the cluster size becomes larger. Overall, we find that the proposed technique scales well with the number of nodes in the Spark cluster, showing a significant reduction in job execution time with increase in cluster size.

6. Conclusion and Future Work

With the development of mobile networks and the rapid growth of spatial data, traditional data storage and query models seem to be inadequate when dealing with massive spatial data. In this paper, the distributed memory file system Alluxio is used for data storage and indexing; at the same time, a big data input format for query of spatial data is proposed based on Spark computing framework, the entire retrieval process from memory to memory and read data selectively, reducing I/O load and CPU load. Through comparative experiments, the distributed retrieval method proposed in this paper has better query performance than the traditional method on the premise of efficient data

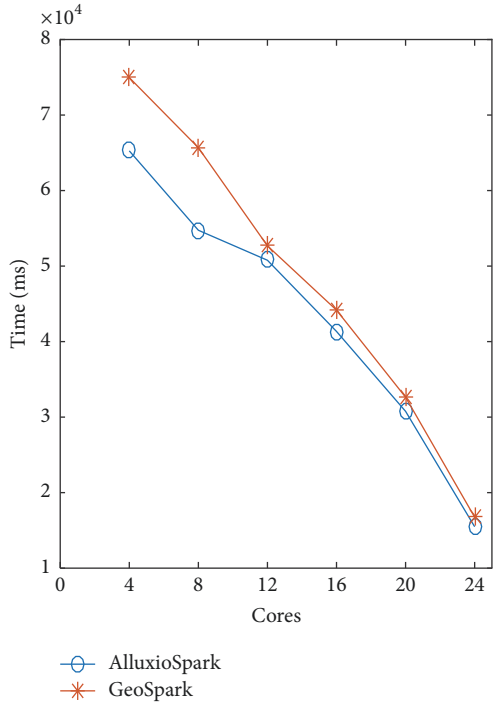


Figure 18: Impact of number of cores for k-NN query.

organization. The next step is to build more efficient spatial index and optimize Spark processing details to improve distributed query efficiency.

Data Availability

The spatial data (including global high-resolution remote sensing images and vector data of various layers) used to support the findings of this study comes from ENVI's own data set. Anyone can install ENVI and find experimental data in the "/data" folder.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported in part by National Natural Science Foundation of China (no. 31770768), Harbin Science and Technology Innovation Talent Research Project (no. 2014RFQXJ132), and China Forestry Nonprofit Industry Research Project (no. 201504307).

References

[1] Z. Wang, Z. Luo, and K. Wei, "5G Service Requirements and Progress on Technical Standards," *Zte Technology Journal*, vol. 20, no. 2, pp. 2–4, 25, 2014.

[2] 4G AMERICAS, 4G Americas' Recommendations on 5G Requirements and Solutions, white paper (EB/OL) 2014, http://www.4gamericas.org.

[3] IMT- 2020(5G) Promotion Group, 5G Vision and Requirements, white paper [EB/OL], 2014, http://www.IMT-2020.cn.

[4] X. Wang, G. Han, X. Du, and J. J. P. C. Rodrigues, "Mobile cloud computing in 5G: Emerging trends, issues, and challenges [Guest Editorial]," *IEEE Network*, vol. 29, no. 2, pp. 4-5, 2015.

[5] R. Kumar and S. Rajalakshmi, "Mobile cloud computing: Standard approach to protecting and securing of mobile cloud ecosystems," in *Proceedings of the 2013 International Conference on Computer Sciences and Applications, CSA 2013*, pp. 663–669, December 2013.

[6] N. Gupta and A. Agarwal, "Context aware mobile cloud computing: Review," in *Proceedings of the 2nd International Conference on Computing for Sustainable Global Development, INDIACom 2015*, pp. 1061–1065, March 2015.

[7] Y. Cui, J. Song, C.-C. Miao, and J. Tang, "Mobile Cloud Computing Research Progress and Trends," *Jisuanji Xuebao/Chinese Journal of Computers*, vol. 40, no. 2, pp. 273–295, 2017.

[8] R. H. Güting, "An introduction to spatial database systems," *The VLDB Journal*, vol. 3, no. 4, pp. 357–399, 1994.

[9] T. Devogele, C. Parent, and S. Spaccapietra, "On spatial database integration," *International Journal of Geographical Information Science*, vol. 12, no. 4, pp. 335–352, 1998.

[10] F. Wang, J. Kong, L. Cooper et al., "A data model and database for high-resolution pathology analytical image informatics," *Journal of Pathology Informatics*, vol. 2, no. 1, article 32, 2011.

[11] F. Wang, J. Kong, J. Gao et al., "A high-performance spatial database based approach for pathology imaging algorithm evaluation," *Journal of Pathology Informatics*, vol. 4, no. 1, p. 5, 2013.

[12] A. Pavlo, E. Paulson, A. Rasin et al., "A comparison of approaches to large-scale data analysis," in *Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems, SIGMOD-PODS'09*, pp. 165–178, July 2009.

[13] L. Y. Wei, Y. T. Hsu, W. C. Peng et al., "Indexing spatial data in cloud data managements," *Pervasive & Mobile Computing*, vol. 15(C), pp. 48–61, 2014.

[14] S. Puri, D. Agarwal, X. He et al., *MapReduce Algorithms for GIS Polygonal Overlay Processing*, 2013.

[15] C. Ji, T. Dong, Y. Li et al., "Inverted Grid-Based kNN Query Processing with MapReduce," in *Proceedings of the 2012 Seventh ChinaGrid Annual Conference (ChinaGrid)*, pp. 25–32, Beijing, China, September 2012.

[16] A. Eldawy and M. F. Mokbel, "A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data," *VLDB Endowment*, pp. 1230–1233, 2013.

[17] A. Aji, F. Wang, H. Vo et al., "Hadoop gis: a high performance spatial data warehousing system over mapreduce," in *Proceedings of the VLDB Endowment*, pp. 1009–1020, 2013.

[18] F. Wang, X. Wang, W. Cui, X. Xiao, Y. Zhou, and J. Li, "Distributed retrieval for massive remote sensing image metadata on spark," in *Proceedings of the 36th IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2016*, pp. 5909–5912, July 2016.

[19] A. Cahsai, N. Ntarmos, C. Anagnostopoulos, and P. Triantafillou, "Scaling k-Nearest Neighbours Queries (The Right Way)," in *Proceedings of the 37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017*, pp. 1419–1430, June 2017.

[20] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: a cluster computing framework for processing large-scale spatial data," in *Proceedings of the Sigspatial International Conference on Advances in Geographic Information Systems*, vol. 70, ACM, 2015.

[21] L. Chen, Y. Tang, M. Lv, and G. Chen, "Partition-based range query for uncertain trajectories in road networks," *GeoInformatica*, vol. 19, no. 1, pp. 61–84, 2014.

[22] H. D. Chon, D. Agrawal, and A. E. Abbadi, "Range and k NN query processing for moving objects in grid model," *Mobile Networks & Applications*, vol. 8, no. 4, pp. 401–412, 2003.

[23] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the 5th ACM Symposium on Cloud Computing, SOCC 2014*, November 2014.

[24] Z. Li, Y. Yan, J. Mo, Z. Wen, and J. Wu, "Performance Optimization of In-Memory File System in Distributed Storage System," in *Proceedings of the 2017 International Conference on Networking, Architecture, and Storage (NAS)*, Shenzhen, China, August 2017.

[25] D. Šidlauskas, S. Šaltenis, C. W. Christiansen, J. M. Johansen, and D. Šaulys, "Trees or grids? Indexing moving objects in main memory," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS 2009*, pp. 236–245, November 2009.

[26] A. Guttman, "R-trees: a dynamic index structure for spatial searching," *ACM SIGMOD Record*, vol. 14, no. 2, pp. 47–57, 1984.

[27] Z. Liu, "A k-d tree-based algorithm to parallelize Kriging interpolation of big spatial data," *Giscience & Remote Sensing*, vol. 52, no. 1, pp. 40–57, 2015.

[28] M. Demirbas and X. Lu, "Distributed Quad-Tree for Spatial Querying in Wireless Sensor Networks," in *Proceedings of the IEEE International Conference on Communications*, pp. 3325–3332, IEEE, 2007.

[29] J. McVay, N. Engheta, and A. Hoorfar, "High Impedance Metamaterial Surfaces Using Hilbert-Curve Inclusions," *IEEE Microwave and Wireless Components Letters*, vol. 14, no. 3, pp. 130–132, 2004.

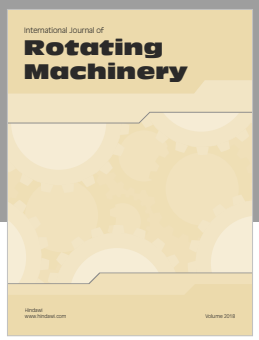
[30] T. Su, W. Wang, Z. Lv, W. Wu, and X. Li, "Rapid Delaunay triangulation for randomly distributed point cloud data using adaptive Hilbert curve," *Computers and Graphics*, vol. 54, article no. 2631, pp. 65–74, 2016.

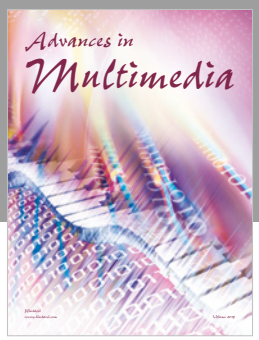
[31] F. Azzedin, "Towards a scalable HDFS architecture," in *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, pp. 155–161, May 2013.

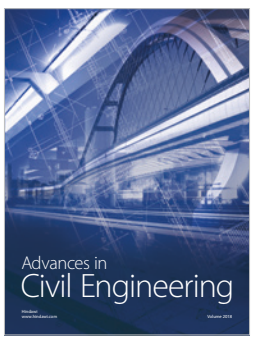
[32] X. Li, B. Dong, L. Xiao, L. Ruan, and Y. Ding, "Small files problem in parallel file system," in *Proceedings of the 2011 International Conference on Network Computing and Information Security, NCIS 2011*, pp. 227–232, May 2011.

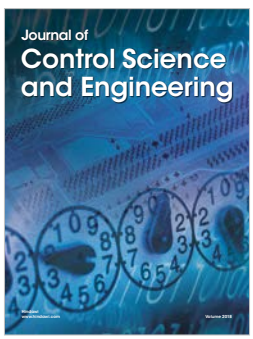
[33] B. Dong, Q. Zheng, F. Tian, K.-M. Chao, R. Ma, and R. Anane, "An optimized approach for storing and accessing small files on cloud storage," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1847–1862, 2012.

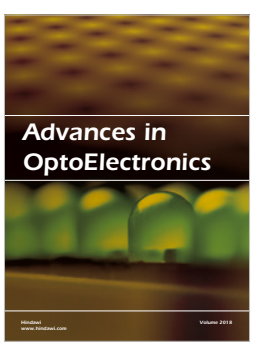
[34] Z. Chi, F. Zhang, Z. Du, and R. Liu, "A distributed storage method of remote sensing data based on image blocks organization," *Journal of Zhejiang University, Science Edition*, vol. 41, no. 1, pp. 95–112, 2014.

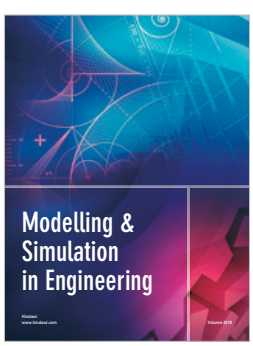


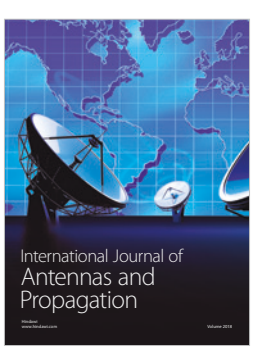


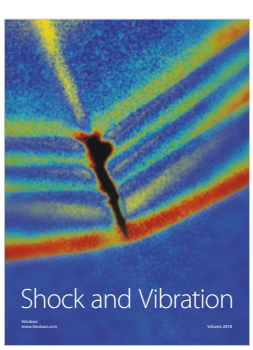


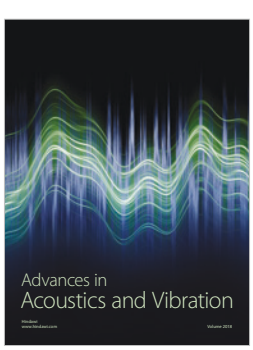












Hindawi

Submit your manuscripts at
www.hindawi.com