# Providing R-Tree Support for MongoDB

Longgang Xiang, Xiaotian Shao, Dehao Wang

[a] State Key Laboratory of Information Engineering in Surveying Mapping and Remote Sensing, Wuhan University, Wuhan 430079, China - (geoxlg, shaoxiaotian, wangdehao)@whu.edu.cn
[b] Collaborative Innovation Center of Geospatial Technology, Wuhan 430079, China

**Commission VI, ICWG IV/II**

**KEY WORDS:** R-tree, Spatial index, MongoDB

**ABSTRACT:**

Supporting large amounts of spatial data is a significant characteristic of modern databases. However, unlike some mature relational databases, such as Oracle and PostgreSQL, most of current burgeoning NoSQL databases are not well designed for storing geospatial data, which is becoming increasingly important in various fields. In this paper, we propose a novel method to provide R-tree index, as well as corresponding spatial range query and nearest neighbour query functions, for MongoDB, one of the most prevalent NoSQL databases. First, after in-depth analysis of MongoDB's features, we devise an efficient tabular document structure which flattens R-tree index into MongoDB collections. Further, relevant mechanisms of R-tree operations are issued, and then we discuss in detail how to integrate R-tree into MongoDB. Finally, we present the experimental results which show that our proposed method out-performs the built-in spatial index of MongoDB. Our research will greatly facilitate big data management issues with MongoDB in a variety of geospatial information applications.

## 1. INTRODUCTION

With the development of data acquisition technologies, the data needed to be stored expands strikingly in both volume and velocity. Under this circumstance, NoSQL (not only SQL) databases, which focus on high throughout and free scalability, start to emerge. NoSQL databases provide totally different storage patterns and query mechanisms compared with relational databases. According to their storage patterns, NoSQL databases can be divided into several types: Key-value database like Redis (Redis Labs, 2016), Document-oriented database like MongoDB (10gen, 2016), Graph database like Neo4j (Wikipedia, 2016) and the Column-Oriented database like HBase (Wikipedia, 2016).

Among various NoSQL products, MongoDB is extensively used in an increasing number of industries and companies (e.g., GitHub, Sourceforge, Taobao, etc.) due to its rich query language and high availability. And also it is acclaimed in geographic information fields. Relevant researches start with the discussion of methodology of storing spatial data in MongoDB (Zhang, 2014). After spatial data (encoded in GeoJSON) was officially supported by MongoDB, performance comparisons between MongoDB and relational databases (Santos, 2015 and Duan, 2015), which reveals MongoDB's capability in spatial tasks, has been made. Meantime, MongoDB is widely used in many spatial applications such as LIDAR data management (Boehm J, 2015), atmosphere environment Monitoring (Han, 2015) and sensor web (Liu, 2014).

In order to improve performance of querying geographical data, spatial indexes are indispensable in modern spatial databases. When it comes to accessing plane (non-geodetic) spatial data, R-tree (guttman, 1984) is definitely the most prevalent and extensively used one among a variety of spatial indexes (quad-tree, k-d tree, geohash,etc). Throughout the ages, several R-tree variations were developed by researchers. R*-tree (Beckmann, 1990) improves the R-tree's query performance by applying more powerful pruning roles, which reduces both coverage and overlap of the nodes. TPR-Tree (Saltenis, 2000) extends the R-tree capability of execute spatio-temporal query for moving objects. SD-Rtree (Du Mouza C, 2007a) brings the well-known R-tree structure into the scalable distribute environment. Besides the popularity in science committee, R-tree is also well-supported by many relational databases (e.g. ORACLE, PostGIS for PostgreSQL).

MongoDB's native spatial index, named 2dsphere (note: 2d index, supporting flat coordinate system, was no longer advocated by MongoDB), first partitions the earth surface at multiple resolution levels and then indexes the resulting cells with B$^+$-tree. That is, 2dsphere only supports the access of spatial data with geodetic coordinate (longitude and latitude). However, in geographical applications which focus on city/county scale, planar Cartesian coordinate is widely used, too. Thus, constant and time-consuming data transformation between two coordinate systems (Boehm J, 2015) is a necessity when applying 2dsphere index to these applications, not to mention that sophisticated spherical computation is much more costly than Cartesian computation in spatial operations (Oracle, 2015). To solve this problem, this paper introduces a way to integrate R-tree index into MongoDB and implements corresponding accessing methods. By taking both advantages of R-tree's wide suitability and NoSQL databases' high performance and scalability, the application areas of MongoDB will be significantly enlarged. Besides, traditional GIS applications will also benefit from it.

The rest of this paper is structured as follows. In section 2, we introduce an efficient and tabular R-tree index structure tailored to MongoDB's document-orient data model. Section 3 describes R-tree index-related data schemas and mechanisms. In section 4, we discuss the method of integrating R-tree index with MongoDB. In the last 2 sections, we evaluate the system's performance, draw conclusions, and discuss future work.

## 2. STORING R-TREE INDEX STRUCTURE IN MONGODB

### 2.1 MongoDB's Storage Unit

Before further discussion of the documental-oriented index structure of MongoDB, a review of MongoDB's storage model is made. Unlike relational databases, MongoDB stores unstructured data in BSON (Binary JSON) documents rather than rows with fixed structure. Furthermore, a document can be nested to represent a complicated structure such as GeoJSON, an open standard spatial data format supported by MongoDB. For each stored document in MongoDB, a 12-bytes unique ID named "oid" is assigned. Querying through oid in MongoDB, when indexed by B+-tree, is very efficient. A MongoDB's dataset consisted of documents is referred as a collection (RDBMS's table counterpart). Comparisons between MongoDB and RDBMS are listed in Table 1.

| Items | MongoDB | RDBMS |
|---|---|---|
| Dataset | Collections | Tables |
| Record | documents | Tuples |
| Node identifier | 12-byte Objectid | Numerical unique ID |
| Schema in same dataset | free | fixed |

Table 1. Comparison between MongoDB and RDBMS

### 2.2 Document-oriented R-tree data structure

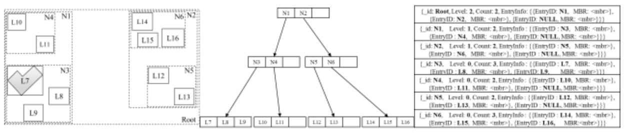
One documental R-tree node designed in this paper consists of four key-value pairs. The first one (oid) denotes node identifier. The second one denotes the node level counting from leaves to the root. The third one denotes the number of entries in this node. And the last one specifies those entries' information. Note that the fourth key-value element is nested and its value component is declared as an array of sub-documents, each of which denotes an index entry. Specified by its level, one such sub-document references either the document identifier of its child node (level>1), or the corresponding spatial object stored as a document in another collection (level=1). Figure 1 shows an example of an R-tree node document with three entries.

```
{
  "_id":              "53465e5fd2fa5e5bfe3dfc11",
  "Level":            2,
  "Count":            3,
  "EntryInfo":    [{"EntryID":" 5346b65e5fd2fa5e5bfe3dfc12", "MBR":[0.0,0.0,3.0,3.0]},
                   {"EntryID":" 5346b65e5fd2fa5e5bfe3dfc13", "MBR":[4.0,4.0,5.0,5.0]},
                   {"EntryID":" 5346b65e5fd2fa5e5bfe3dfc14", "MBR":[2.0,5.0,3.0,6.0]},
                   {"EntryID":" 000000000000000000000000", "MBR":[0.0,0.0,0.0,0.0]}]
}
```

Figure 1. An example of R-tree node document

There are three main merits of this tabular design. First, each node stores the minimum bounding boxes (MBRs) of all its entries. Thus, when received a query request, the database can quickly rule out irrelevant entries and consequently does not have to access their descendants. As a result, performance deterioration is avoided in node retrieval procedure. Second, the fixed document structure is used for storing all nodes with different numbers of actual entries (up to a specified value M). A "Null" value of Entry's oid indicates that the branch is currently empty (e.g. 4th entry of the node in Figure 1.). Comparing with variable structures, using fixed structure will pre-allocates space for empty entry and avoid constantly data moving in hard disc during node expansion or shrinking, and therefore improving efficiency (MongoDB, 2016). The last but not least, served as a redundant short-cut pair, "Count" can avoid frequent scan through "EntryInfo" during the stage of inserting and rebalancing.

With this documental structure, R-tree can be flattened as a collection into MongoDB. Thus the tree navigation through document identifiers (oid) can be performed. Figure 2 illustrates an example. There are seven documents in the right collection and each of them exactly corresponds to one node in the middle R-tree.



{_id: **Root**, Level: **2**, Count: **2**, EntryInfo: {{EntryID: **N1**, MBR: <mbr>}, {EntryID: **N2**, MBR: <mbr>}, {EntryID: **NULL**, MBR: <mbr>}}}
{_id: **N1**, Level: **1**, Count: **2**, EntryInfo : {{EntryID : **N3**, MBR: <mbr>}, {EntryID : **N4**, MBR: <mbr>}, {EntryID: **NULL**, MBR: <mbr>}}}
{_id: **N2**, Level: **1**, Count: **2**, EntryInfo : {{EntryID : **N5**, MBR: <mbr>}, {EntryID : **N6**, MBR: <mbr>}, {EntryID : **NULL**, MBR: <mbr>}}}
{_id: **N3**, Level: **0**, Count: **3**, EntryInfo : {{EntryID : **L7**, MBR: <mbr>}, {EntryID : **L8**, MBR: <mbr>}, {EntryID: **L9**, MBR: <mbr>}}}
{_id: **N4**, Level: **0**, Count: **2**, EntryInfo : {{EntryID : **L10**, MBR: <mbr>}, {EntryID : **L11**, MBR: <mbr>}, {EntryID : **NULL**, MBR:<mbr>}}}
{_id: **N5**, Level: **0**, Count: **2**, EntryInfo : {{EntryID : **L12**, MBR: <mbr>}, {EntryID : **L13**, MBR: <mbr>}, {EntryID : **NULL**, MBR:<mbr>}}}
{_id: **N6**, Level: **0**, Count: **3**, EntryInfo : {{EntryID : **L14**, MBR: <mbr>}, {EntryID : **L15**, MBR: <mbr>}, {EntryID : **L16**, MBR:<mbr>}}}

Figure 2. Flattening a R-tree into a MongoDB collection of documents

## 3. INDEX-RELATED SCHEMA AND MECHANISM

In this section, we introduce the index-related schemas, as well as their relations, that are needed to support all R-tree related operations, including index management (create, drop), spatial queries (range, approximate) and maintaining R-tree's balance.

### 3.1 Index-related Schema

Four kinds of collections, which are spatial data collections (SDC), index collections (IC), geometric metadata collection (GMC) and index metadata collection (IMC), are devised with different purposes but all contribute to our index mechanism. Their functions are described in detail as follows.
First, spatial data encoded in GeoJSON (GeoJSON, 2008) are stored in SDC, along with some other non-spatial attributes (SDC are ordinary collection with one field encoded in GeoJSON to store spatial data). Then, an accessorial index collection (IC), as introduced in section 2, is assigned for a target SDC. Literally, an index collection is named by its corresponding spatial data colloction's name appended with the suffix "_RTreeIndex" (e.g. the index collection name of spatial data collection "building" is "building_RTreeIndex"). Under this naming rule, one can easily identify the index collection for a certain spatial data collection. Besides index collection, spatial metadata collection (GMC) is designed to record each indexed collection, along with its geometric properties, while index metadata collection (IMC) records the index's parameters. According to the schema in Figure 3, "namespace" records the indexed spatial dataset in form of "A.B", in which A donates the database name and B donates the collection name. Meantime, "Fieldname" keeps a record of the spatial field on which the R-tree index is created. "GTYPE" specifies the data layer's geometry type, limited to one of the seven natively supported geometry typies in MongoDB. "MBR" is the layer's minimum bound rectangle and also the MBR of R-tree's root node. "CRS" specifies the Cartesian coordinate system used by the data layer and "IndexInfo" references to a document located in index metadata collection (IMC), which contains the parameters of R-tree, such as the fan factor and the oid of the root node.

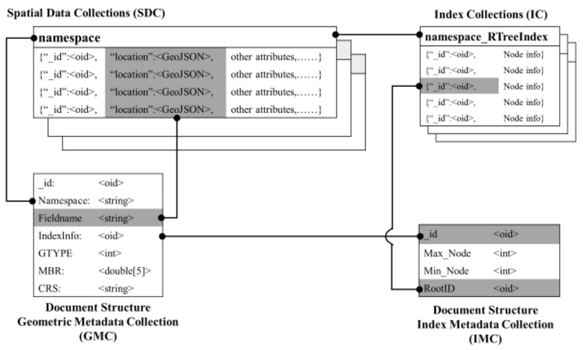


Figure 3. Index related schema

### 3.2 Index-related mechanism

Following the schema proposed in section 3.1, an R-tree operation can be easily turned into a sequence of manipulations performed on the collections discussed above.

Taking spatial query operations as an example, a three-step algorithm is depicted as following. The first step is to verify the existence of target index in GMC and then load the index parameters from IMC. Considering the fact that spatial query is frequently called in a practical application, we cache metadata of both GMC and IMC into system memory to optimize the I/O access. The second step is to filter, which navigates the R-tree nodes stored in IM and excludes spatial objects whose MBRs are not intersected with query range. Consequently, we get a potential candidate set. Finally, after the precise refining upon these candidates is done, the final result set is obtained.

Unlike query operations, inserting or deleting spatial objects in SDC may induce a deformation of the corresponding R-Tree structure, leading to modifications to all SDC, IC and IMC. However, we modify the SDC ahead of other collections for better data protection against unforeseen accidents, such as hardware failure. Note that the R-tree nodes might change during the rebalancing process of R-tree, which may lead to a root oid update in IMC.

Index management operations like creating and dropping an index can be transformed into inserting or removing metadata documents located in GMC and IMC. After that, a corresponding index maintaining process will be in operation. For a better understanding, figure 4 illustrates how four different kinds of collections (SDC, IM, GMC and IMC) are involved during the life span of the R-tree related operations.

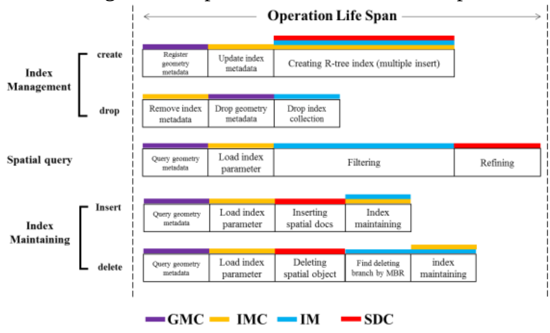


Figure 4. Procedures of R-tree related operations

## 4. 4 INTEGRATION OF R-TREE INDEX INTO MONGODB

Obviously, there exists two ways to implement R-tree operations into MongoDB: an agency to shield MongoDB or an in-built module plugged into MongoDB. Represented by ESRI™ ArcSDE, the former implementation serves as a middleware between the client and database system. Though easier to be achieved, middleware will increase the learning cost of end users who might have to set up environment, import data and be familiar with a new script language. In this paper, an in-built R-tree index module written in C++ was plugged in to MongoDB's system, which allows users to use R-tree index as easily as 2dsphere index. Before further discussion of the implementation, a detailed introduction of MongoDB's architecture is given.

### 4.1 4.1 MongoDB's Architecture

Three key compounds consist the MongoDB cluster, they are shard server, config server and route server. Collections with large volumes are usually horizontally partitioned and stored in multiple shard servers (mongod instance). Router servers, accordingly, interface with client applications and direct operations to the appropriate shard or shards. Config servers (mongos instance) store the cluster's metadata, which contains a mapping of the cluster's data set to the shards. Based on this architecture, MongoDB is able to provide auto-sharding capability, which automatically balances I/O load among multiple shards, thus guarantees flexible horizontal extension again rapid growth of data. Based on the architecture, the integration of R-tree index (see Figure 5) is designed as following:

- **Storage allocation:** we shard spatial data collection (SDC), and index collection (IC) if necessary, into multiple shards, on which a hash based shard-key is assigned. By making the most use of distributing memories, I/O efficiency over sharded large collections are greatly improved. Unlike SDC and IC, two metadata collections (GMC and IMC) are stored in config servers for better protection of the metadata from unexpected user interferences.
- **Module-embedding:** an R-tree module is plugged into mongos, in which the upper level CRUD commands are provided. Using these commands with R-tree algorithm greatly simplify the implementation by ignoring the partition details of sharded collections.

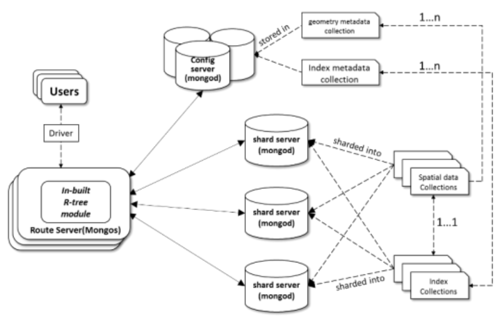


Figure 5. MongoDB's architecture with R-tree integration

### 4.2 4.2 R-tree module integration

First, we modify the message parsing module of mongos, so as to pinpoint the newly added R-tree commands from received messages. During this stage, a metadata exchange with config servers is necessary to verify if R-tree index exists in a certain collection. If an R-tree index has been built, the corresponding R-tree commands will be run. Otherwise, a native database command will be executed.

In order to support these R-tree commands, a two-level R-tree module is designed. The above level consists of a set of storage detached R-tree algorithms such as insert, delete and query. The ground level, correspondingly, provides I/O interface features on data retrieval, R-tree node manipulation and index metadata management, for the above algorithms. Connected into the command center of mongos, the I/O interface is able to translate basic R-tree function into a sequence of CURD operations, which will be dispatched into target shards. Although designed for MongoDB, this two-level R-tree module can be easily ported into other databases and file system, with a tweak of the I/O interface.

In spatial query operations, the filtered candidates from R-tree will be handed into a refining module, which uses Cartesian computation powered by a famous open source geometry library GEOS. Figures 6 shows us a diagram of relevant modules in mongos.

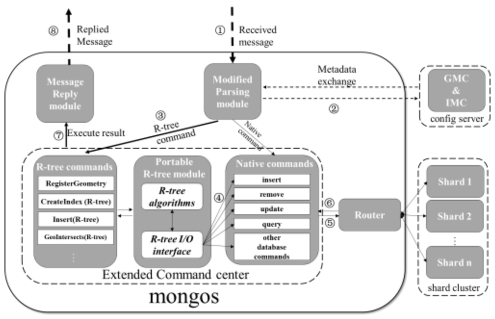


Figure 6. Integration detail of R-tree module

## 4.3 R-tree related Command design

In our case, seven commands listed in Table 2 are involved in R-tree index operations. However, six of these commands already exist in MongoDB's command set. We reuse these native command interface to ensure backward compatibility. Unlike geodetic coordinate system, the metadata like CRS is very important for applications based on Cartesian coordinate system. So we introduce a new command--"RegisterGeometry", which brings the concept of layer to MongoDB and registers the geometric properties of geospatial data stored in SDC. As described in the last section, every R-tree related command is based on a series of native commands. Table 2 details their composition.

| Command Name | Command Type | Base command(s) | | | | | |
|---|---|---|---|---|---|---|---|
| | | drop | create | insert | remove | update | find |
| ensureIndex | Native | | √ | √ | | | √ |
| createIndex | Native | | √ | √ | | | √ |
| registerGeometry | Newly Added | | √ | √ | | | √ |
| dropIndex | Native | √ | | | √ | | √ |
| insert | Native | | | √ | | √ | √ |
| remove | Native | | | √ | √ | √ | √ |
| find | Native | | | | | | √ |

Table 2 R-tree related command design

## 5. PERFORMANCE EVALUATION

This section focuses on the performance evaluation of the proposed plug-in R-tree index by comparing our method with 2dsphere index through range queries.

### 5.1 Evaluation Environment

Our experiments are carried out on a PC with 3.1GHz E3 1231 v3 CPU, Windows Server 2008 platform and 8 GB RAM. The version of MongoDB is 2.6. A city scale (20km*20km) dataset is chosen in this test. Clipped from open street map (OSM), the test data consists of 195823 building polygons located in the center of Washington DC. A copy of original data (geostatic coordinate system) projected into Cartesian coordinate system is prepared to facilitate R-tree queries. A snapshot of the tested dataset is shown in Figure 7.

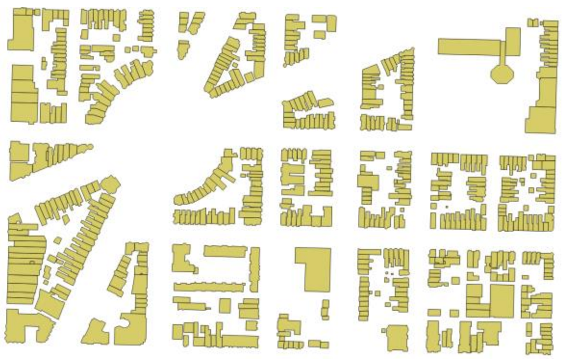


Figure 7. An example of tested data

### 5.2 Comparing setup

Before the performance evaluation, a group of query windows scaling from 0.5% to 10% over the whole area of input data were generated. A 2dshpere index is created on original dataset, and we also create five R-tree indexes with fan factors of 8,16,32,64,128 for the projected dataset. Considering the fact that MongoDB will continue caching frequently used data into main memory which will significantly affect the system's performance, a systematic database warm up before evaluation is required.

The evaluation workload is designed as follows. First, we start MongoDB and import the dataset. Second, some query operations are performed for database warming up. Then, query operations with the whole set of generated windows will be done by using certain indexes. And the run time of each query is recorded. Finally, the database system will be shut down to release memory and prepare for another round of evaluation.

### 5.3 Evaluation result

In the first evaluation experiment, the performance of R-tree with different fan factors is compared. Note that in this experiment, the time to read geometries from spatial data collection was excluded, resulting in a better discrimination of run time caused by various fan factors. As shown in the top of Figure 9, the R-tree with fan factor 32 outperforms the others. This is because the size of each node document (3888Byte) is approximately equal to the system's page size (4096Byte), the basic unit of data swapping between physical memory and hard disk. In the second experiment, fan factor is fixed to 32, and the result is presented in the bottom of Figure 8. It can be noticed that R-tree shows a significant improvement against 2dsphere on the efficiency of range query processing. And more importantly, this improvement becomes more obvious as query scale increases.

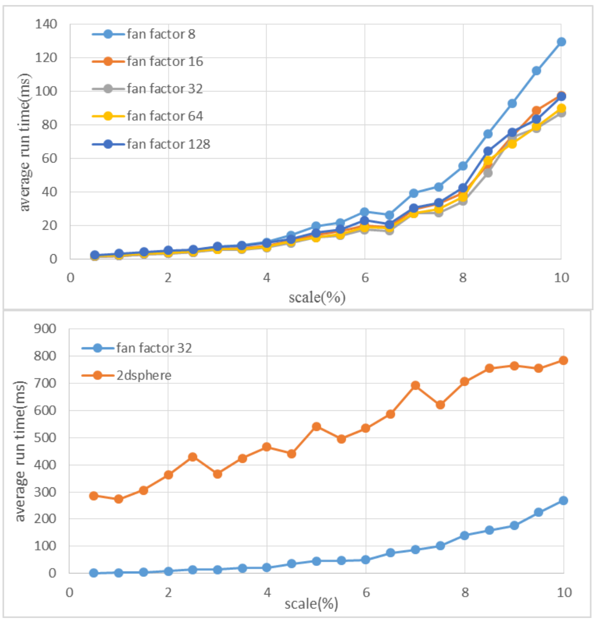


Figure 8. Average run time of range queries

## 6. CONCLUSION AND FUTURE WORK.

In this paper, we presents a new version of MongoDB that integrates with R-tree index. Therefore we can manage projected spatial data and perform R-tree related operations like index management, spatial queries over scalable MongoDB cluster. The performance evaluation results suggest our method is a better option than 2dsphere at least in city/county scale GIS applications.

In the near further, we plan to implement and test more query types for our flattened R-tree index within MongoDB, including point query, circle query and nearest query. After that, we will open our module in the internet and hope to benefit more people in the GIS-related fields.

## REFERENCES

10gen, 2016, “Introduction to MongoDB”, https://docs.mongodb.org/manual/introduction/ (25 Mar. 2016).

Boehm J, Liu K. 2015, NoSQL for storage and retrival of Large LIDAR data collections [J], ISPRS 2015, Commission III, WG III/5.

Beckmann N., Leriegel H., Schneider R., etc. 1990, The R*-tree: an efficient and robust access method for point and rectangles [C], SIGMOD’s 90

Duan, M.R., Chen, G., 2015, Assessment of MongoDB’s spatial retrieval performance[C], Geoinfomatics 2015.

Du Mouza C, Litwin W, Rigaux P, 2007, SD-Rtree: A Scalable Distribute RTree [C]. ICDE 2007: 296-305

Guttman A., 1984, R-Tree: A Dynamic Index Structure for Spatial Searching[C]. SIGMOD Conference 1984: 47-57

GeoJSON, 2008, “The GeoJSON Format Specification”, http://geojson.org/geojson-spec.html (16 Jan. 2008)

Han, M., Feng, K., 2015, GIS application based on Cloud Storage for Atmosphere Enviroment Monitoring [C], IC3ME 2015, 972-976

MongoDB, 2016, “Document Growth”, https://docs.mongodb.org/manual/core/data-model-operations/#data-model-document-growth (3 Apr. 2016).

Redis Labs, 2016, “Introduction to Redis”, Israel, http://redis.io/topics/introduction (25 Mar. 2016).

Saltenis, S., Jensen, C., 2000, Indexing the positions of Continuously Moving objects [C] SIGMOD 2000

Santos P, Moro M, Davis A, 2015, Comparative Performance Evaluation of Relational and NoSQL Databases for Spatial and Mobile Applications [J]. Database and Expert Systems Applications

ORACLE, 2014, ORACLE Spatial and Graph Developer’s Guide [M].

Liu, Q., Mao, S.J., Li, M., etc. 2015, Release and Storage of Mine Gas Monitoring Data Based on Sensor Web [C]. Geoinformatics 2014.

Wikipedia, 2016, “Neo4j”, https://en.wikipedia.org/wiki/Neo4j (4 Mar. 2016).

Wikipedia, 2016, “Apache HBase”, https://en.wikipedia.org/wiki/Apache_HBase (16 Mar. 2016).

Zhang, X.M., Wei, S., Liu, L.M., 2014, an Implementation Approach to Store GIS Spatial Data on NOSQL Database [C]. Geoinformatics 2014