

Technical Note

A Distributed Storage and Access Approach for Massive Remote Sensing Data in MongoDB

Shuang Wang ^{1,2}, Guoqing Li ^{1,*}, Xiaochuang Yao ¹ , Yi Zeng ³, Lushen Pang ⁴ and Lianchong Zhang ¹

¹ Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing 100094, China; wangshuang@radi.ac.cn (S.W.); yaoxc@radi.ac.cn (X.Y.); zhanglc@radi.ac.cn (L.Z.)

² School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing 100049, China

³ School of Information Science and Technology, Beijing Forestry University, Beijing 100083, China; zengyi@bjfu.edu.cn

⁴ School of Computer and Remote Sensing Information Technology, North China Institute of Aerospace Engineering, Langfang 065000, China; pangls@foxmail.com

* Correspondence: ligq@radi.ac.cn; Tel.: +86-10-8217-8062

Received: 16 September 2019; Accepted: 24 November 2019; Published: 27 November 2019



Abstract: With the rapid development of earth-observation technology, the amount of remote sensing data has increased exponentially, and traditional relational databases cannot satisfy the requirements of managing large-scale remote sensing data. To address this problem, this paper undertakes intensive research of the NoSQL (Not Only SQL) data management model, especially the MongoDB database, and proposes a new approach to managing large-scale remote sensing data. Firstly, based on the sharding technology of MongoDB, a distributed cluster architecture was designed and established for massive remote sensing data. Secondly, for the convenience in the unified management of remote sensing data, an archiving model was constructed, and remote sensing data, including structured metadata and unstructured image data, were stored in the above cluster separately, with the metadata stored in the form of a document, and image data stored with the GridFS mechanism. Finally, by designing different shard strategies and comparing MongoDB cluster with a typical relational database, several groups of experiments were conducted to verify the storage performance and access performance of the cluster. The experimental results show that the proposed method can overcome the deficiencies of traditional methods, as well as scale out the database, which is more suitable for managing massive remote sensing data and can provide technical support for the management of massive remote sensing data.

Keywords: remote sensing data; MongoDB; data management; sharding technology; GridFS mechanism

1. Introduction

As a spatial information carrier, remote sensing data play significant roles in many fields, such as environmental monitoring, land resources survey, and disaster assessment, with its characteristics of strong timeliness and large area coverage [1,2]. As earth-observation technologies (including remote sensing and satellite communication) and information technologies develop, a big data era has begun in the remote sensing field, where the amount of remote sensing data is massive and continues to increase exponentially [3,4]. According to statistics, the daily data amount produced by different satellite platforms is increasing at the terabyte level. Landsat8, for example, launched by the National Aeronautics and Space Administration in 2013, generates four hundred global images every day;

Sentinel, launched by the European Space Agency, produces thousands of data every day. Faced with massive remote sensing data, how to store and manage such with high efficiency is an issue that must be solved currently.

Remote sensing data consist of unstructured image data and structured descriptive information attached to the image, which is also called metadata; these two files are commonly saved in the same directory. Previous studies have taken advantage of different methods to store remote sensing data, and the most commonly used method is combining the file system with a database, with the image data stored in the storage device and the metadata stored in the table of database [5]. However, this mode can neither achieve true storage nor reflect the nature of image data, and when the storage path is changed, there are many items that need to be modified in the database, which can reduce the safety and reliability of data to some extent. The second approach that has received much attention is storing image data in a database directly, using the data type “BLOB” [6,7]. The above two methods are implemented based on a relational database management system. While as the amount of remote sensing data grows rapidly, the weaknesses of a relational database in the process of managing massive spatial data are becoming more and more obvious, including slow reading and writing speed and difficult horizontal expansion [8]. Especially the problem of low storage and access efficiency under the background of massive data, which can no longer meet the requirements for data management in a big data era. The last method that some applications adopt is the file system [9,10]. In this condition, the image data are organized through the file mode in high-performance storage devices. However, the process of data retrieval and data acquisition is inconvenient using this method, and in many cases, users have to develop specialized programs to realize these functions. Thus, Not Only SQL (NoSQL) databases, which are typically represented by MongoDB, whose unique data management mode can solve the problems that existed in the relational database, are receiving more attention [11,12]. In addition, the file storage mechanism GridFS within MongoDB can achieve the distributed storage of large binary files [13]. This paper proposes a method to manage remote sensing data based on MongoDB, with the metadata stored in the form of document and the image data stored with GridFS.

Many studies use NoSQL databases to manage the spatial datasets. Li et al. proposed a management strategy based on MongoDB for the frequent modification and complex spatial analysis of large-scale GIS (Geographical Information System) data, and designed experiments to verify the feasibility and effectiveness of their strategy [14]. Xiang et al. flattened a hierarchical R-tree structure into a tabular MongoDB collection to manage planar spatial data, and the results showed that the planar spatial data could be effectively managed by this method [15]. Wang and Hu proposed a cloudizing storage method for unstructured LiDAR point cloud data with the distributed file system GridFS based on MongoDB, and the results showed that the proposed method performed better than the local file system [16]. Current studies mostly concentrate on the management of typical GIS data, such as point, line and surface, and less on remote sensing data, while this paper focuses on constructing a distributed storage strategy for massive remote sensing data to improve the management and service level.

The primary focus of this study is to promote the storage efficiency and access efficiency of remote sensing data by establishing a distributed sharding cluster based on the MongoDB database. In our approach, we propose a distributed storage and access method for remote sensing data based on MongoDB. Firstly, we established a distributed cluster for remote sensing data using the sharding technology of MongoDB. Secondly, we constructed an archiving model to realize the unified management of remote sensing data referring to prevalent international standards and metadata structures. Furthermore, for remote sensing data, metadata and image data are stored separately, with the former stored as documents, and the latter stored with the GridFS mechanism, which is related by the data filename. Finally, with Landsat8 metadata and various satellite data used as experimental data sources, we conducted various experiments to verify the availability and effectiveness of the proposed method. The results show that the method proposed for remote sensing data has higher performance in data storage and access, and can effectively manage massive data. It can also provide guidance for data management strategies and meet the demand for massive data management.

2. Background

2.1. NoSQL and MongoDB

Although relational database management systems occupy an important position in the data management field, the large-scale data introduces a new challenge for data storage and management [17]. Bottlenecks exist in the process of managing massive data with traditional relational databases due to their ACID properties, and people have started to search for a more optimal solution. Consequently, NoSQL technology has emerged as a possible solution [18,19]. For the management of massive data in the distributed system, NoSQL databases usually focus more on the availability and partition tolerance and realize the eventual consistency, wherein it achieves the applications in many special areas [20]. With its feasible data model, high read and write performance, and powerful expandability under big data, NoSQL is more suitable for the storage and management of massive data.

In terms of its storage mode, NoSQL databases can be divided into four categories: key-value database, document-oriented database, column-family database, and graph-oriented database [21]. Presently, as a kind of document-oriented database, MongoDB is attracting much interest due to its free schema, support for complex queries, and powerful expandability. The fundamental data storage unit of MongoDB database is a document, while the BSON (Binary JavaScript Object Notation) format is used as the data storage structure, which is similar to the JSON (JavaScript Object Notation) format. Data is stored in the form of key-value in MongoDB. The document corresponds to the row in the table of the relational database, and many documents compose a collection that corresponds to the table of the relational database. Unlike the table, database users do not need to define the mode of the collection, while the structure of the table needs to be specified in advance in relational databases, and different types of documents can be stored in the same collection. Moreover, for large binary files, MongoDB provides a GridFS mechanism to store them.

2.2. Sharding Technology

Faced with the challenge of data processing in the big data era, the MongoDB database provides sharding technology as the solution to scale out [22]. Sharding technology is a database cluster system used to horizontally expand massive data, and the data is split and stored in different data nodes to handle greater data loads. Applying sharding technology can reduce the pressure on single machine performance caused by high data volume and high throughput applications, and improve random access performance under a large data volume. When storing massive data, one server might not be enough to store the data and provide acceptable read-write throughput. By establishing a cluster, the data can be divided and stored on multiple machines, so the database system can store and process more data to meet the processing needs of the large growth of data. Figure 1 presents the overall sharding cluster architecture of MongoDB.

The MongoDB cluster architecture mainly includes three parts: the shard server, the router server, and the config server. The actual data are stored in the shard server, which can be a replica set or a server. The router server is the entry that clients request the database and is used to address and locate the requests from these clients. All these requests need to be handled by mongos and forwarded to the corresponding shard server. The config server is used to store the configuration information of the router and the shard, which is set up at first and does not need significant space and resources.

When implementing sharding in the collection, one or more fields should be referred to as split data, which is also called the shard key. After the shard key is specified, the data are split into small data chunks, and different chunks are stored in corresponding sharding machines. There are three kinds of shard keys: ascending key, random key, and combined key. The values of the ascending key (such as object id and time) will grow steadily over time, and the latest insert documents will be assigned to the max chunk. The values of the random key have no specific rules to follow, such as name, MD5, and password, as they are absolutely random. As more data are inserted, all the data can be distributed evenly in different chunks. When there is no suitable shard key in the database,

a combined key can be considered. The selection of the shard key can affect the performance of the system, such as its scalability, so it is very important to choose an appropriate shard key [23].

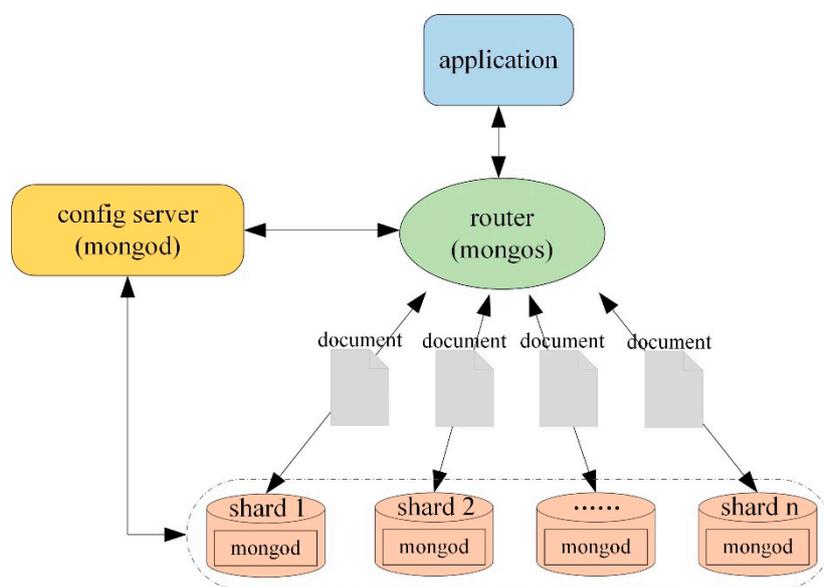


Figure 1. Overall architecture of the MongoDB sharding cluster.

2.3. GridFS Mechanism

MongoDB supports the storage of binary data through a lightweight file storage specification named GridFS. GridFS is a distributed file storage mechanism used to store large binary files, which splits the large file into many small file chunks, and each file chunk is stored as a document [24]. Under this mechanism, the large file is stored in two collections whose default names are fs.chunks and fs.files, with binary data stored in the fs.chunks collection and the descriptive information stored in the fs.files collection, which achieves the distributed storage of data.

When storing a binary file, if its size is greater than the pre-set chunksize value, the file will be divided into several chunks. Each file corresponds to a document in the fs.files collection, which corresponds to one or more documents in fs.chunks.

3. Storage and Access of Remote Sensing Data in MongoDB

3.1. The Distributed Cluster Architecture

MongoDB is an open-source NoSQL database management system with a feasible schema and powerful scalability, which can provide excellent storage and access capability, especially in the large data management field. Aiming at massive remote sensing data, including unstructured image data and structured metadata, a distributed storage method is proposed in this paper based on MongoDB. This research establishes the distributed storage and access architecture for remote sensing data, which is composed of several physical machines, and there is no duplication of data among the cluster.

To ensure the high availability and consistency of the remote sensing data, this research takes advantage of the strategy of “replica sets + sharding” to construct the cluster. That is to say, the shard is also a replica set and consists of a group of mongod processes, while mongod is a process that is mainly used to handle data requests and manage data storage in the MongoDB database. Moreover, as a special member of the replica set, when the backup nodes cannot connect to the primary node, the arbiter takes part in the election of the new primary node, which does not store data and occupies fewer resources. Figure 2 shows the distributed cluster architecture for remote sensing data.

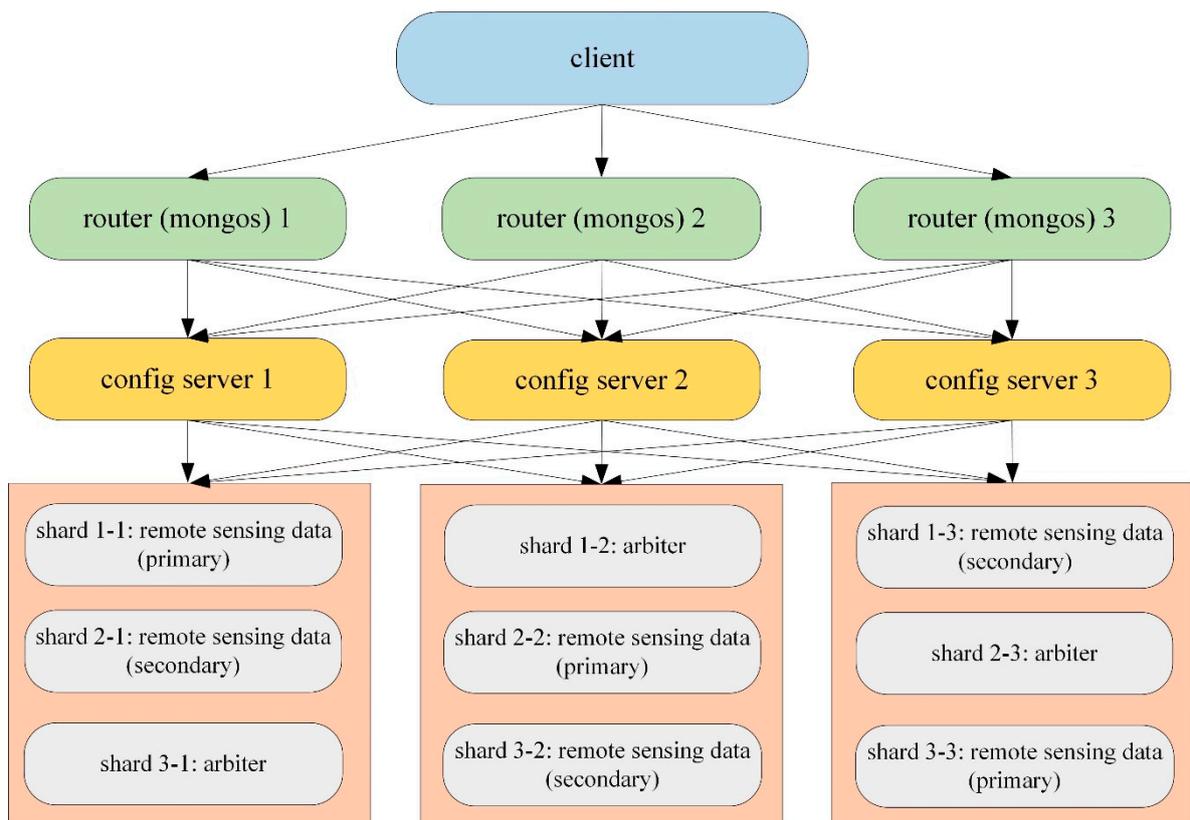


Figure 2. The distributed cluster architecture for remote sensing data.

3.2. Archiving Model for Metadata

Remote sensing metadata plays an essential role in correlative studies of earth-observation, and managing the metadata effectively can contribute to the application and sharing of remote sensing data [25,26]. In our research, remote sensing metadata is the descriptive information of the remote sensing image, which is generated to store attribute information. However, the contents of different metadata files are different, which brings difficulties to the unified management of remote sensing metadata. In the Landsat8 metadata file, for example, the “SENSOR_ID” field is the sensor identifier of the satellite, and the “SPACECRAFT_ID” field is the satellite identifier of the image, while in the ZY-3 metadata file, the corresponding fields are SensorID and SatelliteID. Therefore, constructing a unified archiving model for the management of remote sensing data is urgent.

Based on the investigation and survey of current metadata standards containing ISO (International Organization for Standardization) 19115 geographical information metadata standard and CSDGM (Content Standard for Digital Geospatial Metadata), as well as the metadata structures of multiple remote sensing data sources, the archiving model for remote sensing metadata is established, and its fields are determined: SatelliteID, SensorID, ReceiveDate, geographic coordinates, and so forth. These fields are shown in Table 1.

Table 1. Archiving model for remote sensing metadata.

No.	Field Name	Data Type	Description
1	ImageName	varchar	Image data name
2	SatelliteID	varchar	Satellite id
3	SensorID	varchar	Sensor id
4	ReceiveDate	varchar	Receive date
5	StartTime	varchar	Start time
6	StopTime	varchar	Stop time
7	ProductLevel	varchar	Product level
8	ProductFormat	varchar	Product format
9	Resolution	float	Image resolution
10	CloudPercent	float	Cloud value
11	ImageQuality	int	Image quality
12	CenterLatitude	float	Center latitude
13	CenterLongitude	float	Center longitude
14	TopLeftLatitude	float	Latitude in upper left corner
15	TopLeftLongitude	float	Longitude in upper left corner
16	TopRightLatitude	float	latitude in upper right corner
17	TopRightLongitude	float	Longitude in upper right corner
18	BottomRightLatitude	float	latitude in lower right corner
19	BottomRightLongitude	float	Longitude in lower right corner
20	BottomLeftLatitude	float	latitude in lower left corner
21	BottomLeftLongitude	float	Longitude in lower left corner
22	FileStorePath	varchar	Store path of image file
23	DataDownloadURL	varchar	Download link of data
24	DataOwner	varchar	Owner of the data
25	DataProvider	varchar	Provider of the data

The remote sensing metadata is shown in the form of document in the MongoDB database. When inserting data into MongoDB database, if the “_id” field does not exist in the document, MongoDB will generate the field automatically. Taking one Landsat8 metadata file as example, the standard storage in the MongoDB database is achieved through data processing. The storage mode of the remote sensing metadata is as follows:

```
{
  "_id": ObjectId("5cd393629e11f33380453591"),
  "ImageName": "LC80010762013365LGN01",
  "SatelliteID": "LANDSAT_8",
  "SensorID": "OLI_TIRS",
  "ReceiveDate": "2013/12/31",
  "StartTime": "2013:365:14:38:11.8686080",
  "StopTime": "2013:365:14:38:43.6386040",
  "ProductLevel": "L1",
  "ProductFormat": "GEOTIFF",
  "Resolution": 30,
  "CloudPercent": 4.5,
  "Image Quality": 9,
  "CenterLatitude": -23.11263,
  "CenterLongitude": -69.68594,
  "TopLeftLatitude": -22.06363,
  "TopLeftLongitude": -70.38825,
  "TopRightLatitude": -22.43484,
  "TopRightLongitude": -68.5718,
  "BottomRightLatitude": -23.78778,
```

```

    "BottomRightLongitude": -70.81204,
    "BottomLeftLatitude": -24.16399,
    "BottomLeftLongitude": -68.97206,
    "FileStorePath": "E:\entity data\landsat8",
    "DataDownloadURL": "https://earthexplorer.usgs.gov/browse-link/12864/LC80010762013365LGN01",
    "DataOwner": "USGS",
    "DataProvider": "USGS"
  }

```

In latter experiments, the "TopLeftLatitude" field is selected as shard key to compare the performance of the cluster. When the shard key is specified, remote sensing metadata can be divided and stored on different shards.

3.3. Storage and Access of Image Data Based on GridFS

When storing data in the form of document in the MongoDB database, the volume of each file must be less than sixteen megabytes. However, with the rapid development of earth-observation technologies, the data amount of remote sensing image has already reached hundreds of megabytes, or even bigger, which exceeds the limitation and can never satisfy the storage demand for remote sensing big data [27]. On account that the traditional document-objected method cannot be adopted for the storage of image data, this paper takes advantage of the GridFS file storage mechanism to manage large remote sensing image data.

Under this circumstance, the data are managed in two collections: rs.files and rs.chunks. The keys in the rs.chunks collection include _id, n, data, and files_id. "_id" stands for the unique identifier of the file chunk, "n" stands for the relative position in file of the chunk, "data" stands for the binary data in the chunk, and "files_id" is the same as "_id" in the rs.files collection. The keys in the rs.files collection include _id, length, chunksize, uploadDate, and MD5. "_id" stands for the unique identifier of the document in the rs.files, "length" stands for the number of the bytes, "chunksize" stands for the size of every chunk in bytes, "uploadDate" stands for the date and time that the file is uploaded to GridFS, and "MD5" is the check value of the file, which is calculated by the server.

Taking one of the Sentinel1 image data as an example, the storage structure of the image data in the rs.files collection is as follows.

```

{
  "_id": ObjectId("5cacc5303af3542ea4a36ca9"),
  "chunkSize": 261120,
  "uploadDate": ISODate("2019-04-09T16:15:55.577Z"),
  "length": 915158806,
  "md5": "97eb15fc211c2645268a5b16b21e10f6",
  "filename": "S1A.zip"
}

```

The storage structure of image data of Sentinel1 in rs.chunks collection is as follows.

```

{
  "_id": ObjectId("5cacc5303af3542ea4a36caa"),
  "files_id": ObjectId("5cacc5303af3542ea4a36ca9"),
  "n": 0,
  "data": { "$binary": "...", "$type": "00" }
}

```

Figure 3 shows the procedures for applying the GridFS mechanism of MongoDB database to store image data.

1. Retrieve the image data waiting to be stored according to the specified image name. If the data with the same name exists, then finish the operation; if not, start to store the data with the GridFS mechanism.
2. Store the data in two collections: rs.files and rs.chunks. The "rs.files" collection stores the metadata of each image, while "rs.chunks" collection stores the binary data of each image.
3. The data in the "rs.files" collection usually does not need to be split because its data volume is small, while "files_id" and "n" are selected as a combined shard key to divide the data into different shard nodes.
4. When accessing the image data, the data is retrieved in the "rs.files" collection with the specified query terms, and then the value of "_id" is obtained. Owing to the equal relationship between "_id" in rs.files and "files_id" in rs.chunks, "files_id" is also determined accordingly. Then the image data can be read sequentially through the value of "n".

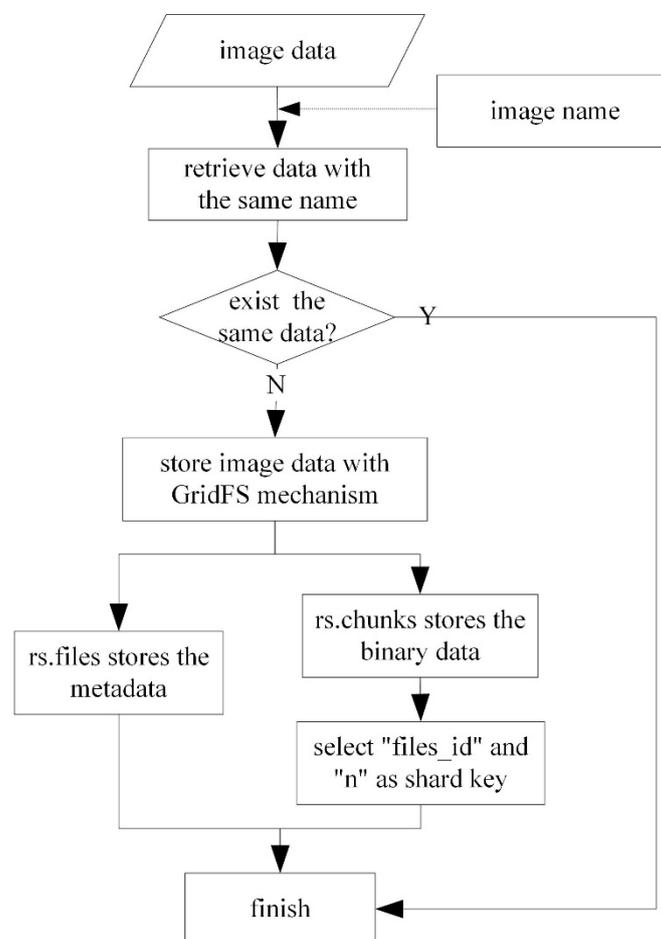


Figure 3. The flow diagram of storing image data with GridFS.

With the GridFS file storage mechanism, the image data is split to small data chunks and stored in the corresponding nodes. Figure 4 shows the storage mechanism of remote sensing image data.

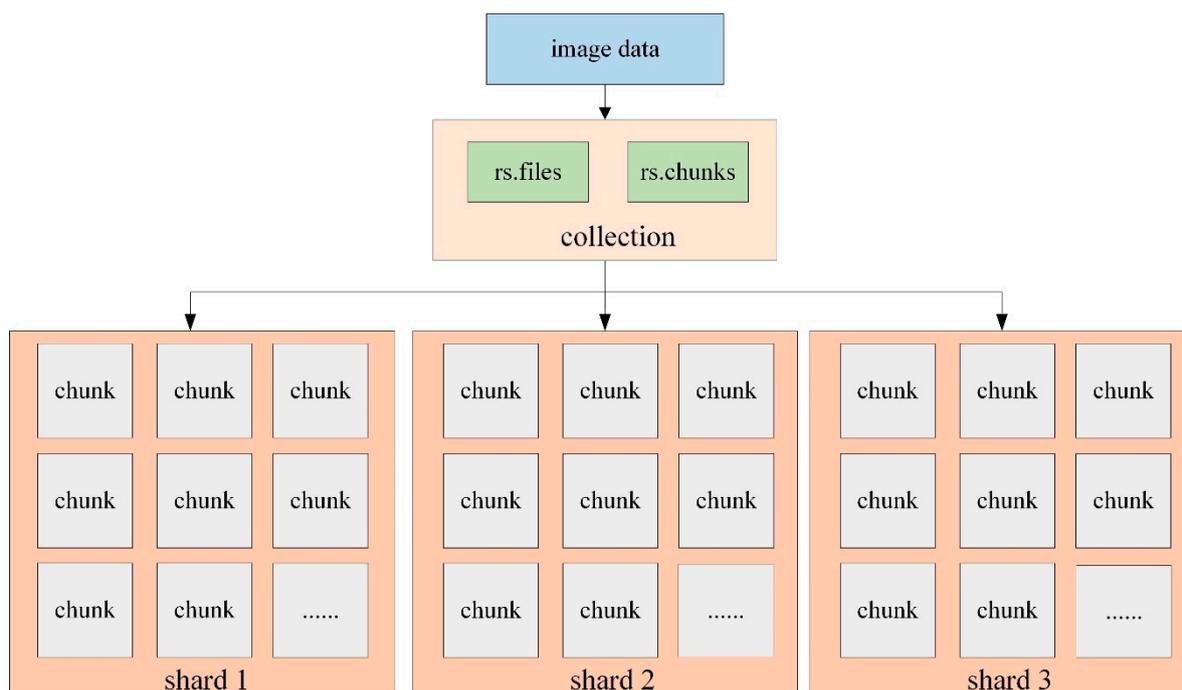


Figure 4. The storage mechanism of remote sensing image data.

4. Experimental Design

The key to the MongoDB distributed cluster is choosing an appropriate shard key. Therefore, to verify the practicability and availability of the proposed method for remote sensing data, several groups of comparison experiments are designed, including remote sensing metadata storage and access under different shard key strategies, image data storage, and access with the GridFS mechanism of MongoDB and PostgreSQL.

4.1. Experimental Data

The experimental data include remote sensing metadata and image data. In this research, the Metadata Extraction Tool, which was developed using Java programming language, is used to extract the necessary fields and values of the metadata file in the Landsat8 data package and transform them into corresponding fields in the archiving model. There are about 100 million global metadata records from 2013 to 2017, which can be obtained from the USGS website "<https://earthexplorer.usgs.gov/>". We downloaded the image data from different data centers, and these data include Landsat data, FY data, and Sentinel data. To simplify the expression, the time information and other information after it in the data filename of each data product are replaced with the symbol "*". Table 2 presents detailed information of each data product.

Table 2. Detail information of each image data.

Data Filename	Satellite	Data Source	Data Amount
LT05_L1GS_123046_*	Landsat5	https://earthexplorer.usgs.gov/	131 MB
LC08_L1GT_123046_*	Landsat8	https://earthexplorer.usgs.gov/	926 MB
FY3A_MERSI_GBAL_L1_*	FY3A	http://satellite.nsmc.org.cn/portalsite/default.aspx	285 MB
FY3B_MERSI_GBAL_L1_*	FY3B	http://satellite.nsmc.org.cn/portalsite/default.aspx	328 MB
FY3C_MERSI_GBAL_L1_*	FY3C	http://satellite.nsmc.org.cn/portalsite/default.aspx	444 MB
S1A_IW_GRDH_1SDV_*	Sentinel1	https://scihub.copernicus.eu/dhus/#/home	872 MB
S2A_MSIL1C_*	Sentinel2	https://scihub.copernicus.eu/dhus/#/home	522 MB
S3A_OL_1_EFR___2016*	Sentinel3	https://scihub.copernicus.eu/dhus/#/home	711 MB
S3A_OL_1_EFR___2017*	Sentinel3	https://scihub.copernicus.eu/dhus/#/home	618 MB

4.2. Experimental Environment

The experimental environment is built on a cluster of three physical computers, and the configuration of each machine is the same: ubuntu-16.04.6 operating system, 16 GB RAM, a 500 GB hard disk, and a 3.20 GHz core CPU. For our experiments, we used MongoDB 4.0.8 and PostgreSQL 11.3 for comparison, which are deployed on the nodes. The configuration of each shard node in the MongoDB cluster is shown in Table 3.

Table 3. Configuration information table of each node in MongoDB cluster.

Node	IP Address	Port
Ubuntu01	10.3.102.199	Shard 1-1: 27001 Shard 2-1: 27002 Shard 3-1: 27003 Mongos 1: 20000 Config 1: 21000
Ubuntu02	10.3.102.204	Shard 1-2: 27001 Shard 2-2: 27002 Shard 3-2: 27003 Mongos 2: 20000 Config 2: 21000
Ubuntu03	10.3.102.205	Shard 1-3: 27001 Shard 2-3: 27002 Shard 3-3: 27003 Mongos 3: 20000 Config 3: 21000

The PostgreSQL database is deployed on each node with port number 5432 in the cluster based on the master-slave structure. The node whose IP address is 10.3.102.199 is configured as the master, while the other two nodes are configured as slave 1 and slave 2, and the slave is a replication of the master.

4.3. Experimental Principle

For the metadata storage experiment, this paper computes the amount of inserted metadata per second, i.e., the insert speed. While for the other experiments, this paper computes the execution time difference of different commands by running programs. The calculation formulas are as follows.

$$T_{avg} = \frac{\sum_{i=1}^n (T_{endn} - T_{startn})}{n} \quad (1)$$

$$S_{insert} = \frac{Vol_{data}}{T_{avg}} \quad (2)$$

Formula (1) is applied to calculate the average execution time, while Formula (2) is used to calculate the average insert speed. In Formula (1), T_{end} represents the end time for executing commands, T_{start} represents the start time for executing commands, n represents the execution time for the same experiment, and T_{avg} represents the average execution time. In order to reduce the experimental errors, the same experiment was carried out five times, so n equals five in this experiment, and the value of T_{avg} is calculated by averaging five figures. In Formula (2), Vol_{data} represents the data volume, and S_{insert} represents the insert speed of data.

5. Results and Analysis

In this section, we carry out the experiments with reference to the experimental design in Section 4. The content of this section contains three aspects. The first two are the experimental results of the storage and access performance comparison of the remote sensing data, and the third is the analysis of the results.

5.1. Metadata

After the values of the fields in the metadata archiving model are obtained, they are first stored in the MongoDB database to facilitate other experiments. In order to study the influence on cluster performance under different shard key strategies, we choose different shard key strategies. Considering that the query terms with remote sensing data mainly concentrated on a spatial range including latitude and longitude information or an imaging time range in practical application, the “TopLeftLatitude” field is chosen as a shard key because it carries spatial information. In the meantime, the hashed values of “_id” are calculated, where the shard key named “_id_hashed” is established. Finally, we use “_id_hashed” and “TopLeftLatitude” as shard keys, and both the storage experiments and access experiments are conducted under these two circumstances.

5.1.1. Storage

The storage experiments are conducted by inserting different amounts of metadata into the MongoDB database. The storage mode of the remote sensing metadata is described in detail in Section 3.2. In order to make the experimental results more intuitive and reliable, we have chosen the insert speed under different shard key strategies for comparison in this experiment. The experimental result is shown in Figure 5.

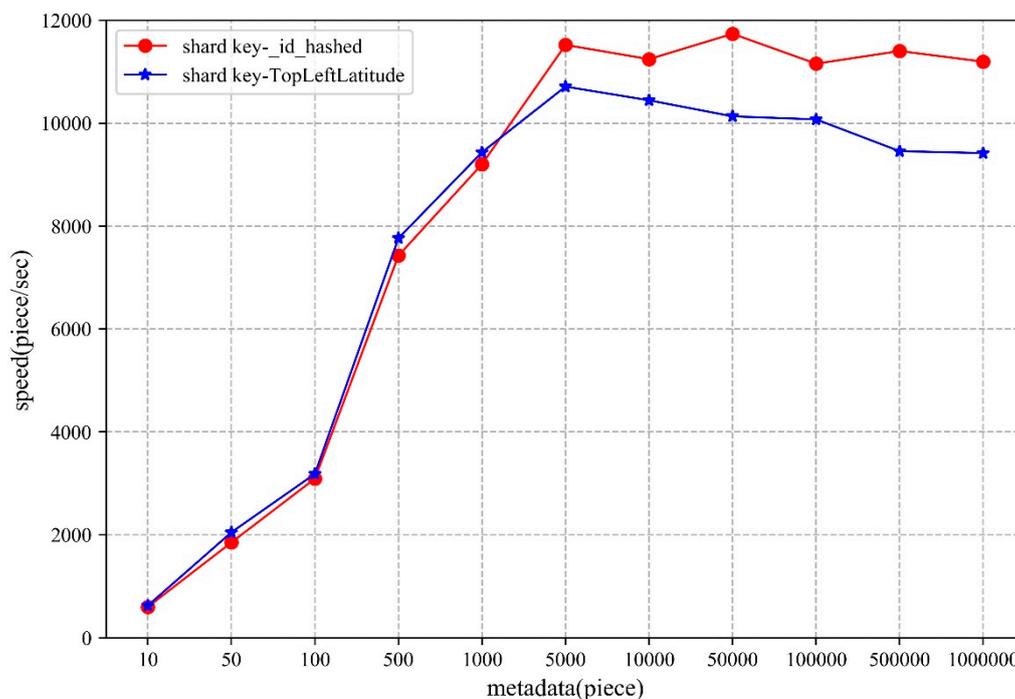


Figure 5. The comparison of the metadata storage time with different shard keys.

As can be seen from the above figure, storage performance is affected by the number of metadata records. When choosing “_id_hashed” and “TopLeftLatitude” as shard keys, the two curves exhibit a similar changing trend, where the insert speed levels off after going through a process of growth. For the shard key named “_id_hashed”, the data are stored randomly and evenly, which guarantees the load balancing among the shard nodes in the MongoDB cluster. While for “TopLeftLatitude” whose values range from -90 to 90 , the new inserted data can be routed to various chunks, and when viewing the data distribution on the shard nodes at this time, we find that the image data is distributed evenly among the nodes. However, strictly speaking, the data storage with “TopLeftLatitude” does not achieve an absolute random and even distribution compared with the “_id_hashed” shard key, which yields better storage performance with the latter.

In addition, when the number of metadata records inserted are less than 5000, the insert speed of the two sharding strategies keeps rising because the metadata are not partitioned at that moment, and there are sufficient resources that can be utilized in the MongoDB cluster.

5.1.2. Access

The access experiments are conducted by retrieving the metadata in a certain range, with longitude values ranging from seventy-five degrees to one-hundred-and-thirty degrees and latitude values ranging from twenty degrees to fifty degrees, which is a rectangular area and covers many provinces of China. The access performance of the remote sensing metadata under various volumes is tested by executing commands in the program. The query command is:

```
"db.meta.find({$and:[ {"TopLeftLatitude": {$lt: 50}}, {"TopLeftLongitude":{$gt: 75}}, {"TopRightLatitude": {$lt: 50}}, {"TopRightLongitude": {$lt: 130}}, {"BottomRightLatitude":{$gt: 20}}, {"BottomRightLongitude": {$gt: 75}}, {"BottomLeftLatitude":{$gt: 20}}, {"BottomLeftLongitude": {$lt: 130}} ]})"
```

This paper chooses the execution time difference for comparison in this experiment. The experimental result under three shard key strategies is shown in Figure 6.

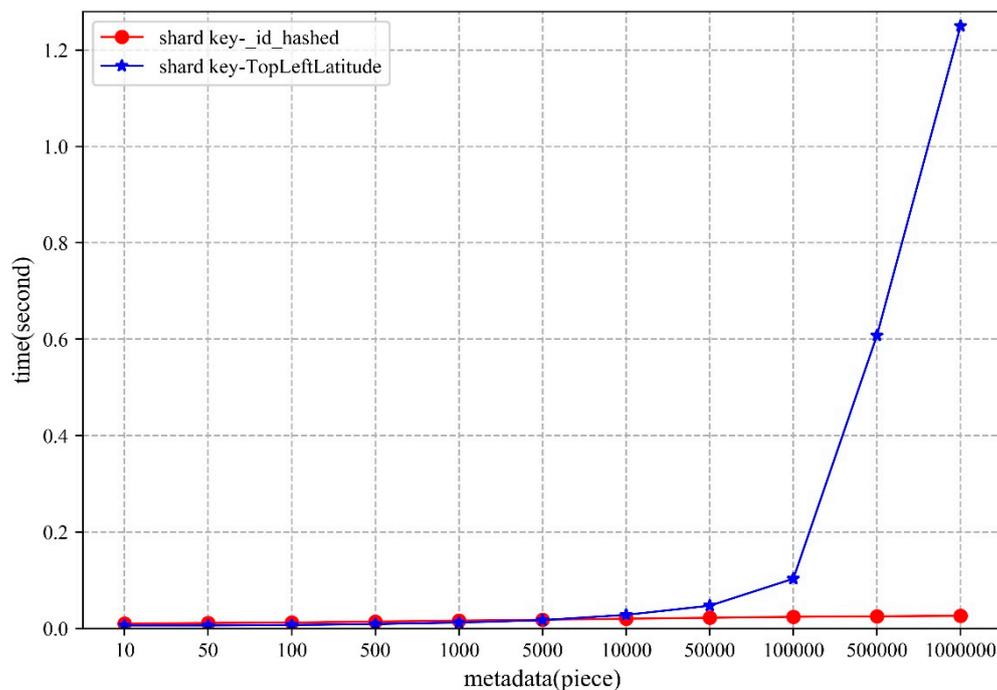


Figure 6. The comparison of the metadata access time with different shard keys.

As Figure 6 shows, the changing trend of the two curves is different. When choosing “_id_hashed” as the shard key, the access time tends to remain steady as the volume of metadata increases, which means the access performance is slightly influenced by the metadata volume in this situation.

However, when the “TopLeftLatitude” field is chosen as the shard key, the access time increases along with the volume of metadata, because the metadata is distributed in different shard nodes according to the value of the “TopLeftLatitude” field, and the data is retrieved among these nodes. In particular, when the metadata volume is 1,000,000, the access time that the “TopLeftLatitude” shard key consumes is forty-eight times longer than that of “_id_hashed”, which indicates that as for the “TopLeftLatitude” shard key, access performance is more affected by the metadata volume than with the latter.

5.2. Image Data

To verify the storage and access performance of the MongoDB database for large binary files (namely, the remote sensing image data in this experiment), the relational database PostgreSQL is selected for comparison. PostgreSQL is recognized as the most powerful open source object-relational database management system; it supports abundant data types and provides rich interfaces. In this experiment, “files_id” and “n” are selected as the combined shard key to reduce the load pressure of a single shard in the MongoDB cluster, while PostgreSQL uses “_id” as the index, as there are only two fields, and the other is applied to store the binary data.

5.2.1. Storage

With different amounts of remote sensing image data inserted into MongoDB and PostgreSQL, the average storage time with the two databases can be computed through multiple experiments with Formula (1), which is chosen for the comparison in this experiment. The experimental result is shown in Figure 7.

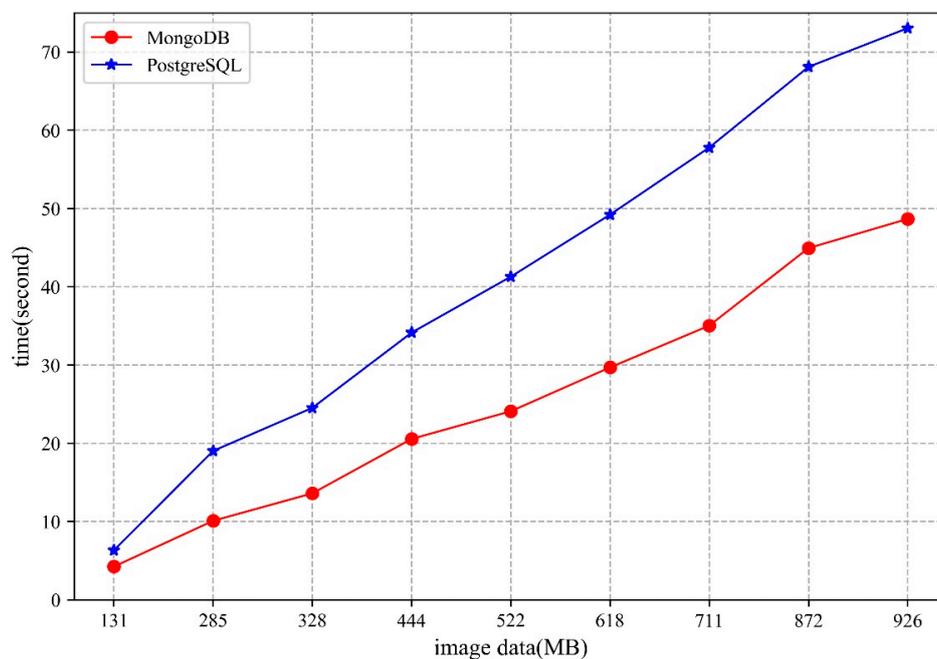


Figure 7. The comparison of the image data storage time with different databases.

As can be observed from Figure 7, the time required to store the image data of both MongoDB and PostgreSQL increases along with the increasing volume of the remote sensing image data, while the storage performance of the MongoDB database is relatively more stable. In addition, when storing the same image data file, PostgreSQL consumes more time than MongoDB. The time difference between the two databases is obvious, especially when the inserted image data amount is large. For example, when inserting the data “S3A_OL_1_EFR____2016*”, the time that PostgreSQL takes is one point six times longer than that of MongoDB. To sum up, MongoDB performs better than PostgreSQL in storing large remote sensing image data.

5.2.2. Access

After all the remote sensing image data are inserted into MongoDB and PostgreSQL, the access experiments can be conducted. The average access time with two databases can be computed through multiple experiments with Formula (1), which is chosen for comparison in this experiment. The experimental results are displayed in Figure 8.

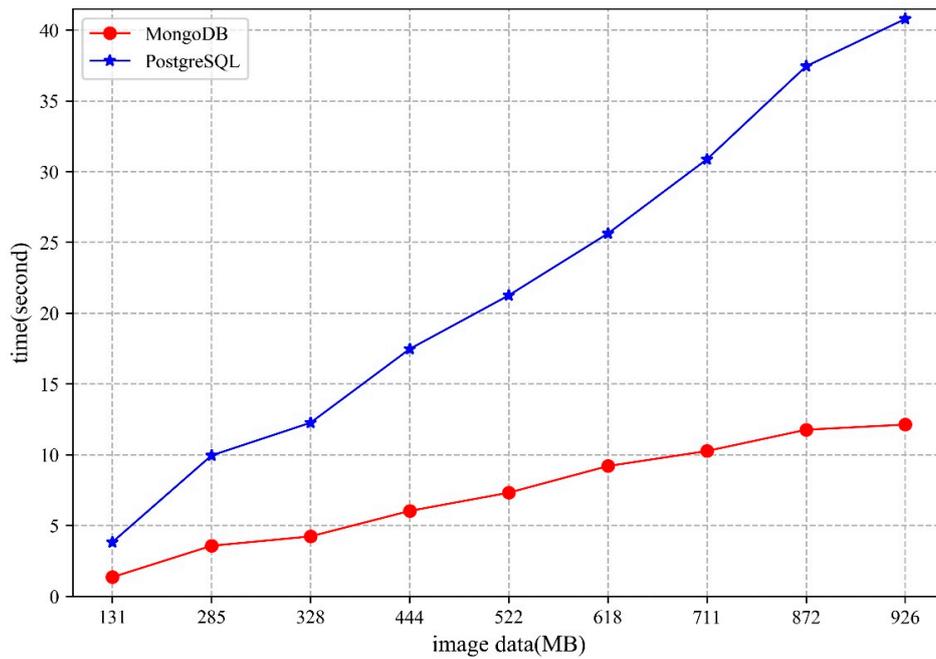


Figure 8. The comparison of the image data access time with different databases.

Figure 8 shows that, as the remote sensing image data amount grows, there is a growing trend in the time required to access data in both MongoDB and PostgreSQL, while the growth of the PostgreSQL database is relatively faster. In addition, when accessing the same image data file, PostgreSQL consumes more time than MongoDB. For instance, when accessing the data “LT05_L1GS_123046_*”, the time that PostgreSQL takes is two point eight times longer than that of MongoDB. Thus, MongoDB performs better than PostgreSQL in accessing large remote sensing image data.

Compared with the method proposed by Wang and Hu [16], which managed the LiDAR point cloud data in the MongoDB database with two shard key strategies including “files_id” and “n”, we used the image data in the same file size as the former to conduct the comparison experiments. Here, we label the method proposed in this paper as “Method 1” and the other as “Method 2”. The experimental results are presented in Figure 9.

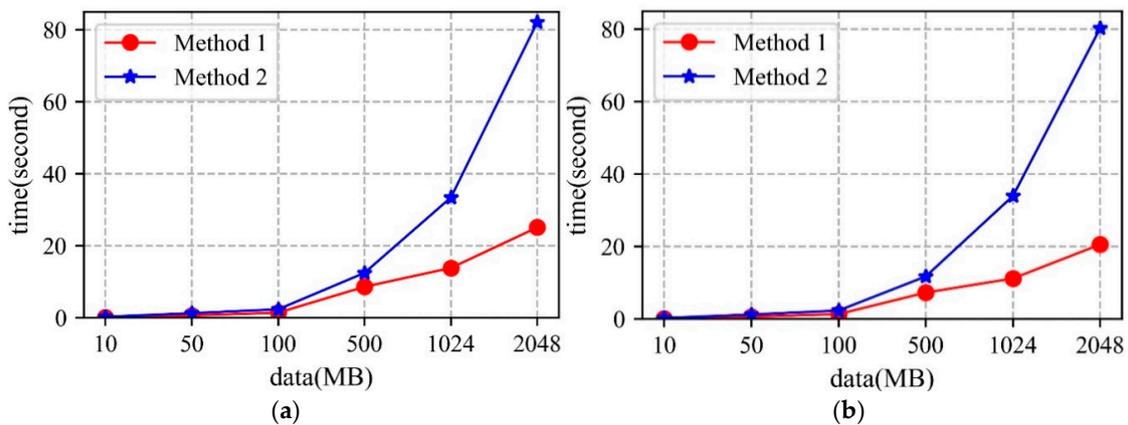


Figure 9. The comparison of the image data access time with different shard keys between the two methods. (a) “files_id”; (b) “n”.

As can be observed from Figure 9, no matter which shard key is applied, Method 1 consumes less time in accessing data than the other method. For instance, compared with Method 2, the data access time with “files_id” decreases by 60.9%, 52.9%, 39.7%, 31%, 58.6%, and 69.4%, respectively,

under different data sizes, which indicates that the performance of Method 1 outweighs that of Method 2.

5.3. Analysis

Through the above experiments, the following conclusions can be drawn by analyzing the experimental results:

- The MongoDB database uses the WiredTiger storage engine, which stores the data as disk files. When there are sufficient memory resources to cope with the storage and access requests in the cluster, higher performance can be obtained. Moreover, the network bandwidth will also influence the storage and access performance of the MongoDB cluster.
- From the perspective of shard key strategy, the paper chooses two different shard keys to conduct the experiments, though neither of them can guarantee optimal performance in both storage and access. Therefore, when designing the shard key strategy, practicality should be considered.

6. Conclusions

Faced with the deficiencies of slow reading and writing speed, difficult horizontal expansion and low query efficiency in the process of managing massive remote sensing data with traditional relational database management systems, this paper proposes a distributed storage and access method for massive remote sensing data using the MongoDB sharding cluster architecture, with structured metadata stored in the form of document and unstructured image data stored with the GridFS mechanism. The result shows that the proposed method can overcome the weak points of traditional methods, scale out the database, and is more suitable to manage massive remote sensing data. For future work, we plan to study the influence of the number of data nodes on the performance of the distributed system.

7. Patents

We have submitted an application for an invention patent resulting from the work reported in this paper to the National Intellectual Property Administration, PRC, and now this patent is open to the public. The patent name is “A distributed storage method for large-scale remote sensing data based on MongoDB”, and the patent application number is 201910585556.4.

Author Contributions: Shuang Wang proposed the main idea and wrote this paper. Shuang Wang, Guoqing Li, and Xiaochuang Yao conceived and designed the paper. Xiaochuang Yao and Lushen Pang provided key guidance on the implementation of the distributed cluster architecture. Yi Zeng and Lianchong Zhang collected the data and participated in the experiment. All authors read and approved this paper.

Funding: This research was funded by the Strategic Priority Research Program of the Chinese Academy of Sciences, grant number XDA19020103.

Acknowledgments: We would like to thank the anonymous reviewers and editors for commenting on this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ma, Y.; Wu, H.; Wang, L.; Huang, B.; Ranjan, R.; Zomaya, A.; Jie, W. Remote sensing big data computing: Challenges and opportunities. *Future Gener. Comput. Syst.* **2015**, *51*, 47–60. [[CrossRef](#)]
2. Guo, H.; Wang, L.; Chen, F.; Liang, D. Scientific big data and digital earth. *Chin. Sci. Bull.* **2014**, *59*, 5066–5073. [[CrossRef](#)]
3. He, G.; Wang, L.; Ma, Y.; Zhang, Z.; Wang, G.; Peng, Y.; Long, T.; Zhang, X. Processing of earth observation big data: Challenges and countermeasures. *Chin. Sci. Bull.* **2015**, *60*, 470–478. [[CrossRef](#)]
4. Reichman, O.J.; Jones, M.B.; Schildhauer, M.P. Challenges and opportunities of open data in ecology. *Science* **2011**, *331*, 703–705. [[CrossRef](#)] [[PubMed](#)]
5. Li, G.; Huang, Z. Data infrastructure for remote sensing big data: Integration, management and on-demand service. *J. Comput. Res. Dev.* **2017**, *54*, 267–283. [[CrossRef](#)]

6. Wang, H.; Tang, X.; Li, Q. Research and implementation of the massive remote sensing image storage and management technology. *Sci. Surv. Mapp.* **2008**, *133*, 156–157. [[CrossRef](#)]
7. Pendleton, C. The world according to Bing. *IEEE Comput. Graph. Appl.* **2010**, *30*, 15–17. [[CrossRef](#)] [[PubMed](#)]
8. Qin, X.; Wang, H.; Li, F.; Li, C.; Chen, H.; Zhou, X.; Du, X.; Wang, S. New landscape of data management technologies. *J. Softw.* **2013**, *24*, 175–197. [[CrossRef](#)]
9. Ramapriyan, H.; Pfister, R.; Weinstein, B. An overview of the EOS data distribution systems. In *Land Remote Sensing and Global Environmental Change*; Ramachandran, B., Justice, C.O., Abrams, M.J., Eds.; Springer: New York, NY, USA, 2011; pp. 183–202.
10. Sun, J.; Gao, J.; Shi, S.; Wang, H.; Ai, B. Application of distributed spatial database in massive satellite images management. *Bull. Surv. Mapp.* **2017**, *5*, 56–61. [[CrossRef](#)]
11. Sadalage, P.J.; Fowler, M. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*; Pearson Education: New Jersey, NJ, USA, 2013.
12. Pokorny, J. NoSQL databases: A step to database scalability in web environment. *Int. J. Web Inf. Syst.* **2013**, *9*, 69–82. [[CrossRef](#)]
13. Gu, Y.; Wang, X.; Shen, S.; Wang, J.; Kim, J.-U. Analysis of data storage mechanism in NoSQL database MongoDB. In Proceedings of the 2015 IEEE International Conference on Consumer Electronics-Taiwan, Taipei, Taiwan, 6–8 June 2015; pp. 70–71.
14. Li, S.; Yang, H.; Huang, Y.; Zhou, Q. Geo-spatial big data storage based on NoSQL database. *Geomat. Inf. Sci. Wuhan Univ.* **2017**, *42*, 163–169. [[CrossRef](#)]
15. Xiang, L.; Huang, J.; Shao, X.; Wang, D. A mongodb-based management of planar spatial data with a flattened R-tree. *ISPRS Int. J. Geo-Inf.* **2016**, *5*, 119. [[CrossRef](#)]
16. Wang, W.; Hu, Q. The method of cloudizing storing unstructured LiDAR point cloud data by MongoDB. In Proceedings of the 2014 22nd International Conference on Geoinformatics, Kaohsiung, Taiwan, 25–27 June 2014; pp. 1–5.
17. Meng, X.; Ci, X. Big data management: Concepts, techniques and challenges. *J. Comput. Res. Dev.* **2013**, *50*, 146–169.
18. Corbellini, A.; Mateos, C.; Zunino, A.; Godoy, D.; Schiaffino, S. Persisting big-data: The NoSQL landscape. *Inf. Syst.* **2017**, *63*, 1–23. [[CrossRef](#)]
19. Shen, D.; Yu, G.; Wang, X.; Nie, T.; Kou, Y. Survey on NoSQL for management of big data. *J. Softw.* **2013**, *8*, 1786–1803. [[CrossRef](#)]
20. Han, J.; Haihong, E.; Le, G.; Du, J. Survey on NoSQL database. In Proceedings of the 2011 6th International Conference on Pervasive Computing and Applications, Port Elizabeth, South Africa, 26–28 October 2011; pp. 363–366.
21. Chodorow, K. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*, 2nd ed.; O'Reilly Media, Inc.: California, CA, USA, 2013.
22. Sharma, S.; Shandilya, R.; Patnaik, S.; Mahapatra, A. Leading NoSQL models for handling big data: A brief review. *Int. J. Bus. Inf. Syst.* **2016**, *22*, 1–25. [[CrossRef](#)]
23. Liu, Y.; Wang, Y.; Jin, Y. Research on the improvement of MongoDB Auto-Sharding in cloud environment. In Proceedings of the 2012 7th International Conference on Computer Science & Education (ICCSE), Melbourne, VIC, Australia, 14–17 July 2012; pp. 851–854.
24. GridFS—MongoDB Manual. Available online: <https://docs.mongodb.com/manual/core/gridfs/#gridfs> (accessed on 10 June 2019).
25. Espinoza-Molina, D.; Datcu, M. Earth-observation image retrieval based on content, semantics, and metadata. *IEEE Trans. Geosci. Remote Sens.* **2013**, *51*, 5145–5159. [[CrossRef](#)]
26. Li, G.; Zhang, H.; Zhang, L.; Wang, Y.; Tian, C. Development and trend of Earth observation data sharing. *J. Remote Sens.* **2016**, *20*, 979–990. [[CrossRef](#)]
27. Huang, K.; Li, G.; Wang, J. Rapid retrieval strategy for massive remote sensing metadata based on GeoHash coding. *Remote Sens. Lett.* **2018**, *9*, 1070–1078. [[CrossRef](#)]

