Received March 18, 2020, accepted April 8, 2020, date of publication April 20, 2020, date of current version May 4, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2988631

A Storage Method for Remote Sensing Images Based on Google S2

XUJIN WANG[1], RUI WANG[1], WEN ZHANG[1], BEIBEI YANG[1], LINYI LI[1], FEI CHEN[2], AND LINGKUI MENG[1]

[1]School of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430072, China
[2]Wuhan Digital Engineering Institute, Wuhan 430070, China

Corresponding author: Lingkui Meng (lkmeng@whu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFC0405806, and in part by the Research on Automatic Extraction and Verification Technology of Typical Water Infrastructure based on GF-6 Satellite Data under Grant 03-Y20A04-9001-17/18-3.

ABSTRACT When using HBase to store tiles of remote sensing images, the spatial position of a tile is often used as the first part of the tile's rowkey so that tiles with high spatial correlations are stored close together to improve query efficiency. We refer to this storage method as the Geo-First model. However, Geo-First models have two problems: the load between nodes is unbalanced, and the accumulation of time-series remote sensing images has a negative impact on storage and query efficiency. Considering these two problems, we proposed a method for storing remote sensing images based on Google S2 and HBase. In our method, two strategies are adopted to eliminate these problems: the balanced placement strategy (BPS) and the periodic storage strategy (PSS). We evaluated our method by focusing on the effectiveness of BPS and PSS. The results show that our method achieves higher tile storage and query efficiency than three Geo-First models based on latitude and longitude, Geohash code, and Google S2 code. BPS effectively balances the load between nodes, while PSS alleviates the negative impact of the accumulation of time-series remote sensing images. Both BPS and PSS greatly improve tile storage and query efficiency.

INDEX TERMS HBase, remote sensing images, Google S2, load balancing, tile storage mode.

I. INTRODUCTION

With the development of ground observation technology, remote sensing images have become an important big data category [1] that play an important role in the fields of economic development, environmental protection and national defence construction [2]. The massive, heterogeneous, multiscale data of remote sensing images make storage difficult. The traditional storage methods used to store remote sensing images can be divided into three types: file system storage methods, relational database storage methods, and file-relational database hybrid storage methods. The above methods can effectively store and manage remote sensing images, but all have flaws such as low retrieval efficiency, poor concurrency, and high expansion costs [3], [4]. In recent years, with the development of Not Only SQL (NoSQL) databases, increasing numbers of researchers have favoured remote sensing image storage based on NoSQL [5]–[14]. Compared with traditional methods, NoSQL has performance advantages in terms of big data access, scalability, and concurrency. As a typical NoSQL database, HBase can efficiently store both structured and unstructured data [15], [16], and its column-family mechanism and storage characteristics make it highly suitable for storing remote sensing images.

As remote sensing technology has advanced, the resolution of remote sensing images has also increased, expanding the amount of data in a single image [17]. Thus, if a single image functions as the basic storage unit, the computational query and storage burden also greatly increase. Furthermore, considerable useless data will be transmitted during image queries because users are usually only interested in a small part of the image data; thus, computing resources and network resources will be wasted. Therefore, researchers often cut remote sensing images into tiles and store the tiles as the basic units in HBase [11], [14]–[28]. As the size of a single image increases, the number of tiles will increase. Thus, a reasonable index for tiles is necessary. A reasonable index structure can greatly improve the retrieval efficiency of the tiles, which is crucial for improving database performance. When using HBase to store tiles for remote sensing images,

The associate editor coordinating the review of this manuscript and approving it for publication was Yakoub Bazi.

the spatial correlations of tiles are a primary consideration for the design of indexes for tiles with high spatial correlation [23]–[28]. Using this approach, highly relevant tiles will be stored together because they have similar indexes, which improves the query efficiency of the tiles. Although this storage model can be used to effectively manage massive tiles, there are some problems: the node load in the clusters is unbalanced; consequently, as the number of stored time-series remote sensing images increases, the storage and query efficiency of the images is affected. These two problems often occur when storing remote sensing images with HBase; unfortunately, they are often overlooked when researchers design storage solutions. We elaborate on the causes of these problems in Section II.A.

A. RELATED WORK

When using HBase to store remote sensing images, researchers focus on designing similar rowkeys for highly spatially relevant tiles. On this basis, the rowkey for a tile often starts with a code that represents the geographic location of the tile, such as the Hilbert curve code, the Z-order curve code, or the line-filling curve code. For ease of presentation, we refer to this code as the geolocation code, and we refer to this storage model, which considers the geolocation code as the primary consideration, as the Geo-First model. Below, we briefly introduce the storage scheme of tiles designed by researchers based on the Geo-First model.

Huo [18] adopted the P2H code, which was obtained based on the Hilbert code, as the geolocation code, and constructed rowkeys based on the P2H code to efficiently query the tiles. Wang *et al.* [19] combined the GeoSOT code with metadata as the geolocation code to construct the tile's rowkey and achieved efficient storage of remote sensing images. Li *et al.* [20] and Zhong *et al.* [21] combined the Hilbert code with other attributes to construct tile rowkeys. Jing *et al.* [9], Jing and Tian [10] combined the tile grid ID, the Hilbert code, and other information to construct the tile's rowkey. Cheng [22] combined the latitude and longitude of the tile with other attributes to form the tile rowkey. When tiles are stored as pyramids, for each level of the pyramid, researchers also often use the Geo-First model to store tiles. For example, Li *et al.* [23] used latitude, longitude and the linear quadtree code as the geolocation code for constructing the tile's rowkey. Fu [24] and Cao *et al.* [25] used the Hilbert code to design the tile's rowkey. Fan [26] used the tile's geographic coordinate code as the geolocation code to form the tile's rowkey. Liu *et al.* [27] combined the tile's grouping and coordinates with other attribute information to form the tile's rowkey. Liu *et al.* [28] combined tile row and column number with other attributes to construct the tile's rowkey.

These studies constructed geolocation codes and stored the tiles in Geo-First models. Although these methods store spatially adjacent tiles together to improve image query efficiency, they all exhibit the same two problems described in Section II.A.

B. RESEARCH CONTRIBUTIONS

This paper proposes an efficient method for storing remote sensing images based on Google S2 and HBase. To address the two problems introduced in Section II. A, we design two strategies for our proposed method. The main contributions of this paper are summarized below.

1) We propose a remote sensing image storage method based on Google S2 and HBase. This method has higher storage and query efficiency than does the Geo-First model.
2) Our proposed method includes a balanced placement strategy (BPS) for tiles that addresses the unbalanced load between nodes.
3) Our proposed method includes a periodic storage strategy (PSS) for tiles that addresses the negative impact of the accumulation of time-series remote sensing images on data storage and query efficiency.

C. ORGANIZATION

The remainder of this paper is structured as follows: Section II discusses the study motivation and relevant background, explains the problems and describes Google S2. Section III introduces our proposed remote sensing storage method and the two included tile storage strategies. Section IV presents the evaluation of the proposed method and the two strategies; and Section V presents conclusions.

II. MOTIVATION AND BACKGROUND

A. PROBLEMS WITH THE GEO-FIRST MODEL

In this section, we introduce the generation mechanism of the two problems described in section I in more detail. Before discussing these problems, we first introduce HBase.

HBase's storage architecture consists of Zookeeper, Master, and RegionServer. Zookeeper is responsible for the overall management and scheduling of HBase. Master is responsible for the region distributions and for cluster load balancing. RegionServer is responsible for the storage and maintenance of regions and provides query and storage access to the data. The regions maintained in RegionServer are the logical units obtained by cutting the data table according to the rowkeys' range. A region consists of several rows of data sorted by the lexicographic order of the rowkeys. When the size of a region reaches a certain threshold, HBase will split it. The split threshold of a region changes dynamically, and as the number of regions increases, the split threshold of a region also gradually increases (e.g., 256 MB, 2 GB, 6.75 GB, and 10 GB). When regions split, the difference in the number of regions maintained by each RegionServer becomes larger, and HBase triggers a load-balancing mechanism when this number reaches a specified threshold. To perform load balancing, HBase redistributes the load pressure to all nodes by evenly distributing the number of regions maintained by each RegionServer to facilitate optimal cluster performance.

Next, we discuss the two problems of the Geo-First model.

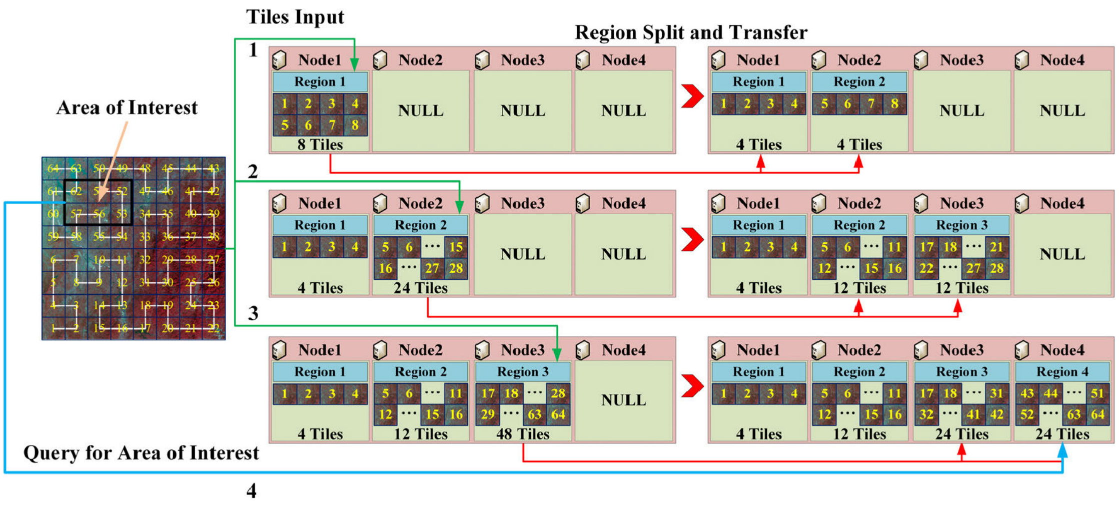


FIGURE 1. The storage process of tiles under the Geo-First Model and the split and transfer processes of the regions.

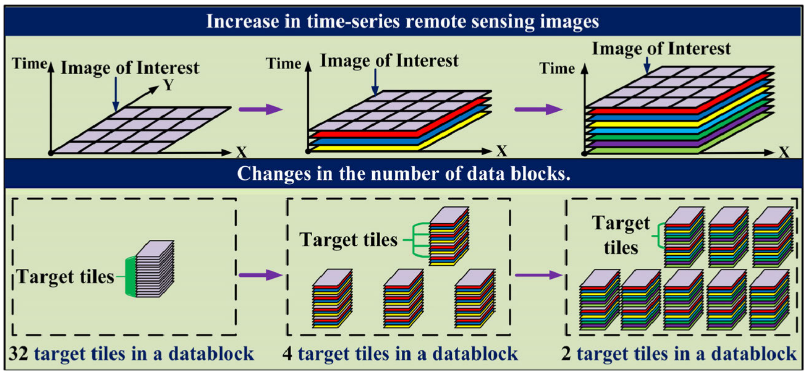

Increase in time-series remote sensing images
Time
Image of Interest
Y
X
Changes in the number of data blocks.
Target tiles
32 target tiles in a datablock
4 target tiles in a datablock
2 target tiles in a datablock

FIGURE 2. Under the Geo-First Model, the changes caused by the accumulation of time-series remote sensing images to the data blocks.

1) UNBALANCED LOAD AMONG NODES

From the previous introduction, a region's default split threshold increases dynamically. Assume that when the number of tiles stored in the region reaches 8, 24, and 48, the region reaches the splitting thresholds. Fig. 1 shows the storage process of the tiles storage process and the split and transfer of regions when storing remote sensing images with the Geo-First model.

As illustrated in Fig. 1, when we store the tiles according to the Geo-First Model, because the rowkey is sorted lexicographically, tiles with similar Hilbert codes are stored together (we use the Hilbert code to represent the geolocation code, which has excellent spatial proximity). As shown in Fig. 1, in the three stages of tile storage, a single node bears the entire load of storage requests in each stage, causing the hotspotting problem. As the number of tiles increases, HBase performs a series of split and transfer operations. Although each node maintains the same number of regions, the sizes of those regions are quite different. As shown in Fig. 1, after tile storage is completed, the four regions have different sizes, causing the data skew problem to arise. Hotspotting and data skewing inevitably cause load imbalance among nodes. If the black box in Fig. 1 is the user's area of interest, then stage 4 indicates that when the user queries data from that area, Node 4 bears all the query requests, and the load pressure is unbalanced among nodes. Therefore, when using the Geo-First Model to store tiles, the query and storage load between nodes will become severely unbalanced.

2) EFFECT OF TIME-SERIES REMOTE SENSING IMAGE ACCUMULATION ON DATA STORAGE AND QUERY EFFICIENCY

In a distributed database, some data are arranged in a continuous sequence to form data blocks, which are the basic units of database management. Data blocks can be used as the basic units of data organization and management to achieve efficient big data storage and management. The size of a data block in HDFS is 64 MB. If a data block can store 16 tiles, then under the Geo-First Model, the changes brought about by the accumulation of time-series remote sensing images in the data blocks are shown in Fig. 2.

As shown in Fig. 2, as the time-series remote sensing images accumulate, the number of data blocks increases. Assume that the image marked in Fig. 2 is a remote sensing image showing a certain area that the user wants to query. We know that when only one remote sensing image of the area is stored, the tiles are stored in a data block, and the database system needs to access and query only one data block. However, when four remote sensing images of the area are stored in the database, the database system needs to access and query four different data blocks to obtain the target tiles. Similarly, when there are eight images, eight data blocks need to be accessed and queried. Obviously, for the Geo-First model, accumulating time-series remote sensing images causes tiles with the same spatial location to be stored together and causes tiles belonging to the same image to become scattered and stored in additional data blocks, which increases the time consumed by a data query. Similarly, when storing images according to the lexicographic ordering rules of rowkeys, the tile storage system also needs to interact with multiple data blocks, which increases the time consumption of data storage.

B. GOOGLE S2

We use the global geographic grid Google S2 as a benchmark for generating tiles and designing the storage and organization mode based on the S2 code. Before describing our method, we explain why we adopt Google S2 to design tile storage solutions instead of the better-known geographic grid Geohash. We introduce Google S2 in detail, including its grid division method and principles, and emphasize the data structure of the S2 code because it is closely related to the proposed method.

Google S2 is a multidimensional spatial point indexing algorithm based on the Hilbert curve. Compared with Geohash [29], the Hilbert curve used by Google S2 has a better spatial clustering effect [30], and there are fewer spatial jumps. In addition, S2 has more grid division levels: it provides 30 levels of grid division. The size range of S2 cells is 0.74 cm^2 to 85,011,012.19 km^2, while the size range of cells provided by Geohash is 6.9 cm^2 to 25,000,000 km^2. In grid division, the granularity of the cell size in S2 is regular and can be applied for multiscale tile storage. Moreover, when the grid division accuracy is maximized, 12 bytes are required for storing the cell id of Geohash, while the cell code in S2 uses a UInt64 for storage, which requires less space.

To further explain the specific implementation algorithm of Google S2 and the data structure of cell id, we briefly introduce the Hilbert curve before introducing Google S2. As an excellent space-filling curve, the Hilbert curve maps two-dimensional spatial positions into one-dimensional digital codes to give cells unique codes while ensuring the spatial proximity of grid codes. The Hilbert curve is generated via stepwise recursion, and an nth-order Hilbert curve consists of four (n − 1)th-order Hilbert curves through the necessary rotations and connections. Fig. 3 shows the generation process for a fourth-order Hilbert curve.

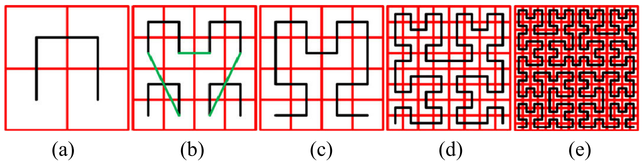


FIGURE 3. The filling process of the fourth-order Hilbert curve.

The first-order Hilbert curve fills a 2 × 2 plane, as shown in Fig. 3(a). For each cell where the inflection point of the first-order Hilbert curve is located, the quadtree is divided and filled with a first-order Hilbert curve. Then, the necessary rotations and connections are conducted to generate a second-order Hilbert curve, as shown in (b) and (c). Similarly, third- and fourth-order Hilbert curves can be generated, as shown in (d) and (e). Thus, through a step-by-step recursive generation method, a Hilbert curve of any order can be generated. Consequently, the plane can be filled and encoded at an arbitrary granularity. Google S2 uses Hilbert curves to fill and encode two-dimensional planes, thereby reducing the two-dimensional space into one-dimensional digital codes. Fig. 4 outlines the main ideas behind the implementation of Google S2.

First, the three-dimensional Earth is projected onto an external cube, and the deformation generated by the projection is corrected so that the spherical coordinates are mapped to the projected coordinates. Then, the cube is expanded into 6 two-dimensional planes, each of which is divided into $2^l \times 2^l$ cell sets according to the level of S2, and the projection coordinates are mapped onto the coordinate axis points. Finally, the Hilbert curve is used for filling and coding, and the geographic coordinates are mapped to the cell id. The S2 algorithm constructs a suitable data structure to characterize an arbitrary spatial position at a specified grid level. This data structure is the basis of our proposed method. Therefore, we introduce the data structure of the S2 code in Fig. 5.

As shown in Fig. 5, the S2 code is a 64-bit binary string. According to our previous description, the projection cube is expanded into 6 planes. The first three bits (yellow blocks) represent the cube plane where the cell is located. The blue block is the tagged bit. The binary string (green blocks) from the fourth bit to the previous bit of the tagged bit is the Hilbert code of the cell. The Hilbert order is increased by one order; thus, 2 bits need to be added to indicate the position of the cell. Because the Hilbert order corresponds to the S2 level, when the level of S2 is 30, 60 bits are required to form the Hilbert code of the cell. The S2 cell code is fixedly stored by UInt64; thus, when the level of S2 is less than 30, the bits after the tagged bit are set to 0. In (b), 101 represents the plane where the cell is located, the next 8 bits represent the Hilbert code of the cell, the red number 1 represents the tagged bit, and the bits after the tagged bit are filled with 0s.

Using this data structure, the S2 algorithm can assign a unique code to any region on Earth according to the accuracy requirements. In addition, due to the superior performance of

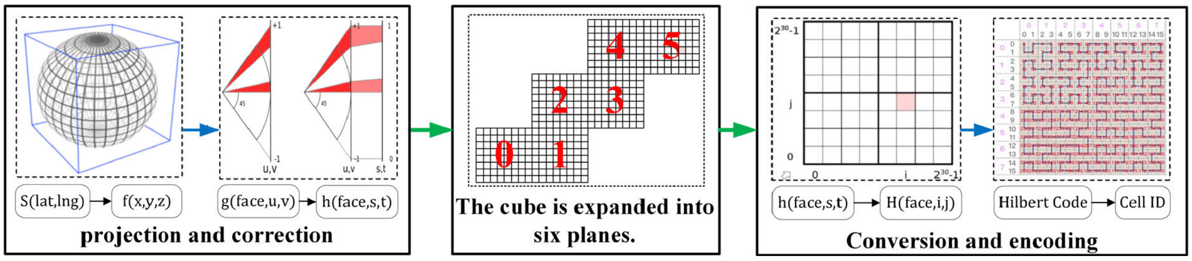


FIGURE 4. The main implementation process of the Google S2 algorithm.

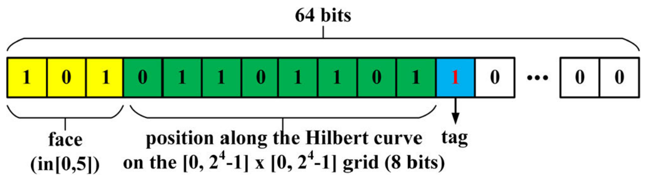

64 bits
1 0 1 0 1 1 0 1 1 0 1 1 0 ••• 0 0
face (in[0,5])
position along the Hilbert curve on the [0, 2^4-1] x [0, 2^4-1] grid (8 bits)
tag

FIGURE 5. Data structure of the cell code of s2.

TABLE 1. The cell sizes corresponding to different s2 levels.

S2 Level	Cell Area/km^2		
	Min	Max	Avg
9	194.643	408.1210	324.2913
10	48.6608	102.0303	81.0728
11	12.1652	25.5076	20.2682
12	3.0413	6.3769	5.0671
13	0.7603	1.5942	1.2667
14	0.1901	0.3986	0.3167
15	0.0475	0.0996	0.0792

the Hilbert curve, the S2 codes obtained by cells with high spatial correlations are close together; thus, this method can be applied for the storage and analysis of spatial data on a global scale.

III. METHODS

We propose a remote sensing image storage method that is more efficient than the Geo-First model. We introduce our method in Section III.A. In addition, our method includes specific strategies for the two problems described in Section II.A. Section III.B introduces the BPS for tiles, which addresses the problem of unbalanced load among nodes. Section III.C introduces the PSS for tiles, which alleviates the negative impact of the accumulation of time-series remote sensing images on tile storage and query efficiency.

A. STORING REMOTE SENSING IMAGES BASED ON HBASE AND GOOGLE S2

When using HBase to store remote sensing images, tiles must be acquired for storage and queries. Before obtaining the tiles, we need to determine the pixel size of the tiles. In our proposed method, we set tile size to 512 × 512 pixels, which is a commonly used size. In addition, we considered that many machine learning algorithms require the input data with a size of 512 × 512 pixels. After determining the tile size, we designed the method for obtaining tiles. Fig. 6 shows the tile acquisition process in our method.

First, we need to choose the appropriate S2 level according to the spatial resolution of the image. The cell sizes under different S2 levels are shown in Table 1. For example, when we store images with a spatial resolution of 8 metres, the ground area corresponding to a tile is approximately 16.7 km^2 ($8^2 * 0.512^2 \approx 16.7$). According to the data shown in Table 1, the S2 level must be 12 to ensure that the largest cell at this level can be covered by 512 × 512-pixel tiles. After determining the S2 level, we obtain the S2 cells based on the geospatial range of the image, as shown in step 1 of Fig. 6. Then, we segment the image based on the cells. We use the centres of the cells as the centres of the tiles and cut 512 × 512-pixel tiles from the original image. Based on the S2 geographic grid, this converts the basic storage unit for remote sensing images from image to tile, which greatly reduces the amount of useless data transmitted in subsequent data queries. A certain amount of redundancy exists in this cutting method; we plan to reconstruct lost tiles from redundant data in subsequent research. Next, the acquired tiles must be stored.

Before storing the tiles, the tile rowkey structure and the data table structure must be designed. To meet the query requirements and avoid the problems described in Section II.A, we designed the rowkey structure shown in Fig. 7.

The partition code is used to address the load imbalance among nodes: we explain this method in detail in Section III.B. The period code is used to address the accumulation of time-series remote sensing images on data storage and query efficiency: we explain this method in detail in Section III.C. The geolocation code is the S2 cell code. In this way, we can store tiles with high spatial correlation together to improve query efficiency. The attribute code is a combination of query conditions such as satellite, sensor, image production time, and cloud cover. The unique code is used to ensure the global uniqueness of the tile's rowkey. For example, when storing GF images, we choose the product number as the unique code.

A well-designed data table structure is also important. In our design, the tile data table structure contains the information necessary for tile queries and tile generation as well

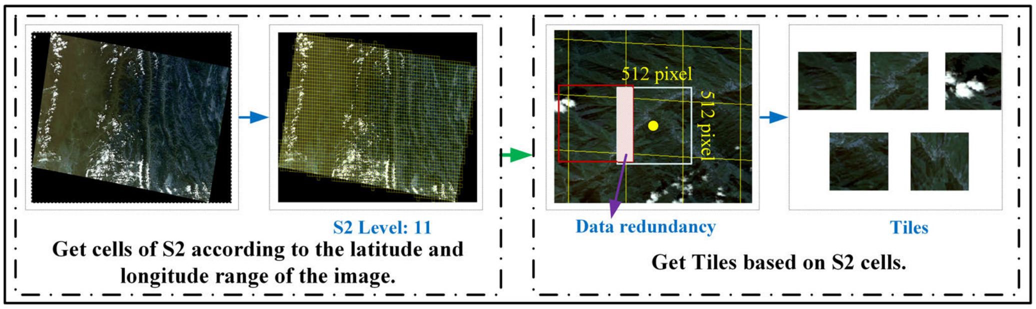

S2 Level: 11
Get cells of S2 according to the latitude and longitude range of the image.
512 pixel
512 pixel
Data redundancy
Tiles
Get Tiles based on S2 cells.

FIGURE 6. Tile acquisition based on S2 grid.

Partition code	Period code	Geoloaction code	Attribute code	Unique code

FIGURE 7. The rowkey structure of the tiles.

the information related to satellites, sensors, band data, affine transformation parameters, and projection coordinate systems. According to the storage mechanism of HBase, data of a column family will be stored together; thus, we use one column family to store this information, which decreases the target data query time. Therefore, we designed the data table structure shown in Fig. 8.

The storage model described in this section can quickly store and query remote sensing images. Most importantly, this method includes two strategies that help eliminate the two problems described in Section II.A. In Sections III.B and III.C, we describe both strategies in detail, as well as the process to obtain the partition code and period code shown in Fig. 7.

B. BALANCED PLACEMENT STRATEGY FOR TILES

To address the problem of load imbalance among nodes, we designed a BPS for tiles based on the characteristics of the S2 code.

As shown in Fig. 9, when there are 4 HBase nodes the image contains a total of 64 tiles; the red blocks indicate the tiles of interest to the user. As described in Section II.A, when we store the image in the Geo-First model, the tiles will be written to the nodes in the order of the Hilbert ID. After region splitting and transfer, the tiles will be distributed among the nodes as shown in Fig. 9, which shows a large gap in the number of tiles stored between nodes, and data skew occurs. In addition, the user's interest data are concentrated in one region, which forces Node 3 to address all queries because it maintains this region. Clearly, the load is seriously unbalanced, and the system cannot achieve optimal performance.

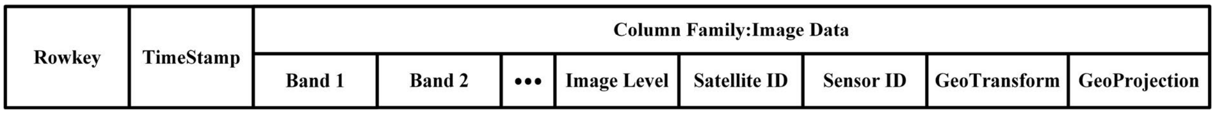
To solve this problem, we propose BPS. Our goal is to enable the tiles to be distributed in the form shown in Fig. 10. Unlike the Geo-First model, we subjectively control the node in which the tile is stored. We perform a modulo operation on the Hilbert ID of the tile and store the tile in the corresponding numbered node based on the result of the operation. For example, when there are n nodes, the nodes are sequentially numbered (0, 1, 2, 3, …, n − 1). If the result of the modulo operation of the Hilbert ID of a tile is equal to 1, the tile will be stored in Node 1. This storage method ensures that the storage loads on the nodes are similar. After storing the tiles, by observing the distribution pattern of the tiles in the cluster, we find that each node stores the same number of tiles. Consequently, the tiles covered by the area of interest are evenly stored across all the nodes, preventing data skew and hotspotting. In addition, tiles with high spatial correlations in each node are still stored in adjacent locations.

In this way, the load pressure is evenly distributed while still ensuring query efficiency. Below, we introduce the implementation method of BPS in detail.

In the HBase storage specification, data partitioning and sorting are controlled by the rowkey. The structure of the rowkey directly determines the data distribution mode in the cluster. As shown in Fig. 7, the partition code is designed to implement the tile distribution pattern shown in Fig. 10. The partition code is used to control the region in which the tile is stored. The ultimate goal is to eliminate the load imbalance among nodes. To store tiles in specified nodes, we need to partition the data table, and the number of partitions needs to be consistent with the number of nodes. We need to let each node maintain a region before tile storage to achieve an initial node load balance. If there are 4 nodes, we number the nodes in sequence (i.e., 0, 1, 2, 3). The rowkey ranges of these 4 regions will then be (0, 0|), (0|, 1|), (1|, 2|), and (2|, 3|), and they will be maintained by the corresponding nodes. Based on the lexicographic order of the data, the partition code will store the tiles in the nodes that maintain the corresponding partitions. For example, if the partition code is 0, then the rowkey formed by the partition code will fall into the (0, 0|) partition, and the tile will be stored in Node 0. Next, we introduce the method used to obtain the partition code.

As shown in Fig. 11, the S2 code is used as the initial code, and certain operations are performed on it to derive the Hilbert ID of the tile. The Hilbert ID acquisition process is divided into three steps.

First, the S2 code of the tile is obtained according to the latitude and longitude of the tile and the S2 level. Second, combined with the data structure of the S2 code we introduced in Section II.B, the Hilbert code representing the plane



Rowkey	TimeStamp	Column Family:Image Data							
		Band 1	Band 2	•••	Image Level	Satellite ID	Sensor ID	GeoTransform	GeoProjection

FIGURE 8. The structure of the tile data table.

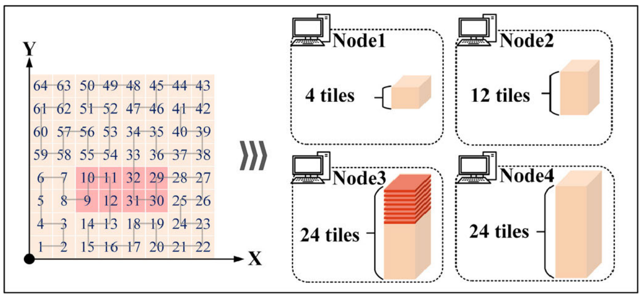


FIGURE 9. The distribution of tiles for the Geo-First Model.

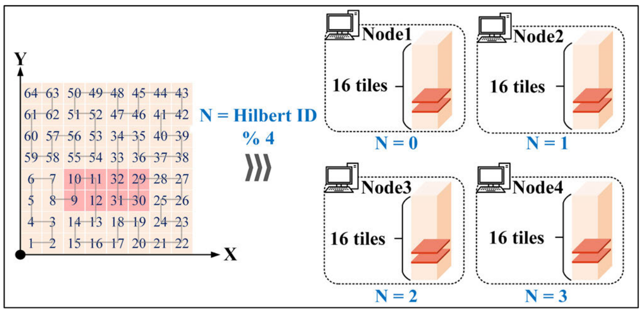


FIGURE 10. The distribution of tiles after using BPS.

position of the tile is obtained. Finally, we converted the Hilbert code to a long integer, which is convenient for the next operation. We record the obtained Hilbert ID as h and the number of partitions as n; then, the partition code p can be calculated by (1).

$$p = h \bmod n \tag{1}$$

TABLE 2. Number of occurrences of each partition code under different S2 levels.

S2 level	Number of tiles	Number of occurrences of each partition code				CV
		0	1	2	3	
6	8	2	2	2	2	0
7	22	6	6	5	5	0.0909
8	63	18	14	15	16	0.0939
9	217	59	52	52	54	0.0527
10	798	210	195	192	201	0.0344
11	3042	773	756	753	760	0.0100
12	11,884	3001	2960	2955	2968	0.0060
13	46,986	11,794	11,725	11,729	11,738	0.0023
14	186,837	46,806	46,656	46,674	46,701	0.0012

Using the above method, we can calculate a partition code for any tile. To prove that the numbers of tiles in the partitions are similar, we used the above method to count the number of occurrences of partition codes for all tiles in a GF1 image. There are 4 nodes in the cluster, and the results are shown in Table 2. The CV in Table 2 means the coefficient of variation. We counted the number of occurrences of each partition code and obtained the standard deviation σ and mean μ. The CV can be obtained by (2).

$$CV = \sigma / |\mu| \tag{2}$$

We divided the data table into 4 partitions (0, 0|), (0|, 1|), (1|, 2|), and (2|, 3|), which are maintained by the corresponding nodes. According to Table 2, at a given S2 level, the partition codes counts are highly similar; thus, the numbers of tiles in the partitions are also highly similar. Therefore, we can conclude that the numbers of tiles maintained by different nodes are also similar.

The BPS strategy maximizes the overall performance of the distributed database by storing the tiles uniformly, which avoids data skew and hotspotting and balances the load among nodes. After ensuring the balanced distribution of the tiles among the nodes, the tiles within the nodes are still stored together according to the spatial correlation, which improves query efficiency.

C. A PERIODIC STORAGE STRATEGY FOR TILES (PSS)

To address the accumulation of time-series remote sensing images, we designed a PSS for tiles.

As we described in Section 2) of Section II.A, when we store the tiles according to the Geo-First model, as time-series remote sensing images accumulate, tiles of the same image become scattered and stored in additional data blocks. When users need to store or query these images, the database must interact with many data blocks, which greatly reduces the query and storage performance of the data. Thus, we propose PSS to change the storage model of the tiles in each node and improve the data storage and query efficiency. PSS aggregates the tiles by time period to store tiles of the same image in fewer data blocks, reducing the time required to interact with the data blocks and improving the efficiency of data storage and queries. As shown in Fig. 12, the image we need to store and query contains four areas: Areas A–D. Storage Model 1 is the storage model of tiles without PSS, and Storage Model 2 is the storage model of tiles with PSS.

As shown in Fig. 12, we added time period control to Storage Model 2. We used 5 days as a time period and stored the tiles in the corresponding time period based on the image production time. We can select the year, month,

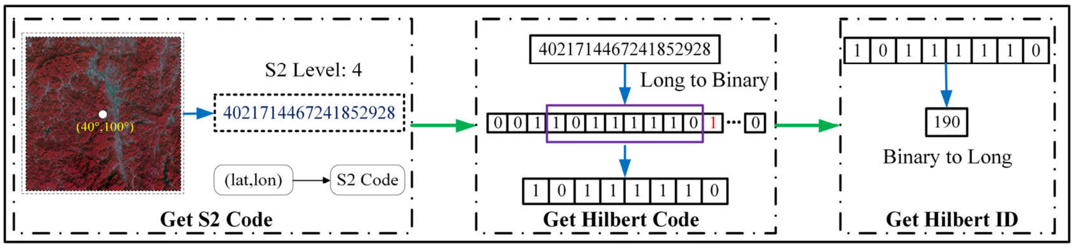


FIGURE 11. The acquisition of the Hilbert ID.

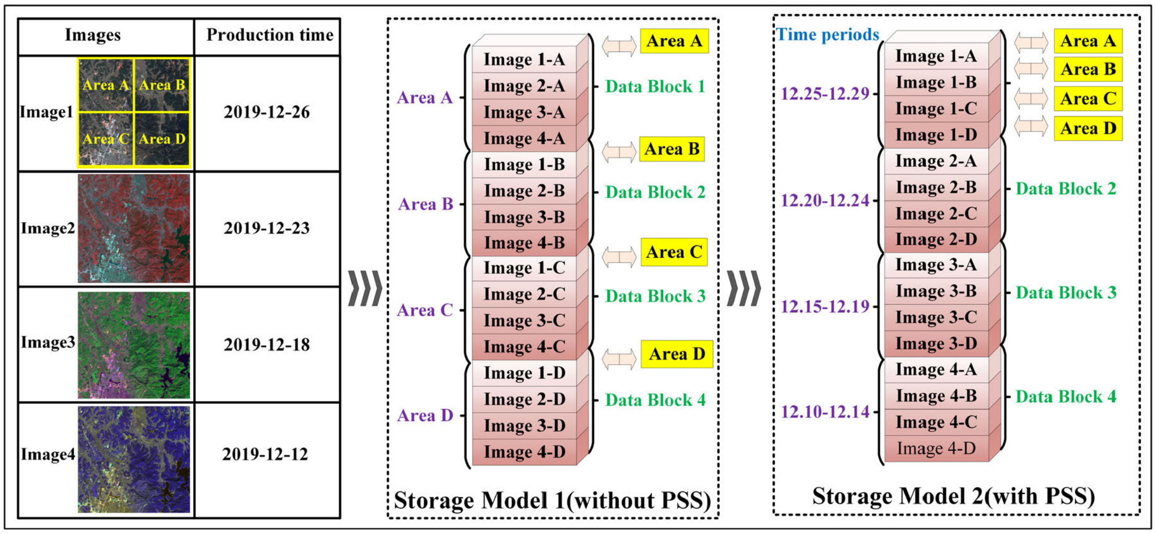


FIGURE 12. Tile storage modes with and without using PSS.

tenth, week, etc., as a time period to meet specific storage requirements. Under Storage Model 1, when we need to store or query an image, we need to interact with 4 data blocks, while Storage Model 2 needs to interact with only 1 data block. In this way, tiles of the same image are stored closer together and distributed over fewer data blocks, improving the query and storage efficiency of the target data. Note that for the tiles in each defined time period, we still store the tiles according to their S2 codes, which ensures their spatial correlation and improves query efficiency. Based on Fig. 12, we designed the period code shown in Fig. 7 to control the storage model for the tiles in a single node. Next, we introduce the implementation of the period code in detail.

First, we choose a certain time of practical significance as the initial time (St) for data management. Because the first satellite was launched on October 4, 1957 [30], we set St to 00:00:00 on October 4, 1957. We refer to a certain time range as a fixed period, called the basic time unit C. For the storage and management of remote sensing images, we recommend using a year, month, or week as the basic time unit. Finally, the period code T of the tiles of the image is the difference between the image acquisition time P_t and S_t divided by C. The calculation method for the period code T is shown in (3).

$$T = \lfloor (P_t - S_t)/C \rfloor \tag{3}$$

We can choose a suitable period for summarizing the data based on the collection density of the time-series remote sensing images. For example, when data collection is frequent, 3 days is used as a basic time unit, while for sparse image collection, 1 year is used as a basic time unit. PSS makes it possible to reduce the number of data blocks that need to be traversed and processed, thereby improving the efficiency of tile query and storage.

IV. EXPERIMENTAL STUDY

In this section, we first introduce the experimental environment and experimental data. Then, we introduce the experiment conducted to compare our method with three common Geo-First models in terms of tile storage and query efficiency. Finally, we introduce the experiments conducted to evaluate BPS and PSS.

A. EXPERIMENTAL ENVIRONMENT AND DATA

This experiment uses an Inspur P8000 server infrastructure with 4 virtualized nodes, 3 of which serve as data nodes to provide HDFS services. The hardware and software parameters of the experiment are shown in Table 3.

We selected a time series of GF1 images of the Erhai area as the data source; there are 12 images produced from January to

TABLE 3. The hardware and software parameters of the experiments.

Items	Parameters
CPU	Intel(R) Xeon(R) CPU E5-2630 L v4 @ 1.80 GHz (40 Cores)
RAM	65,536 MB
Disk	LSI INSPUR SCSI Disk Device * 2
Number of Nodes	4
Node Configuration	4 Cores, 4 GB Memory, 800 GB Disk
Node Operating System	CentOS 7.5.1804
Zookeeper Version	3.4.10
Hadoop Version	2.7.7
HBase Version	1.2.12

December. We use the 11-level S2 grid as the benchmark to cut the image. The tile size is 512 × 512 pixels, resulting in 35,260 tiles. The total data size is approximately 70 GB.

B. EXPERIMENTS

To evaluate our method, we implemented experiment 1. Then, to evaluate BPS and PSS, we implemented experiments 2 and 3. For each experiment, we introduce the experimental process and analyse the results.

1) METHOD EVALUATION EXPERIMENT

a: EXPERIMENTAL METHOD

As introduced in the related work discussion, three space-filling curve codes, i.e., the line-filling curve, Z-order curve, and Hilbert curve, are often used to obtain geolocation codes, as shown in Fig. 13.

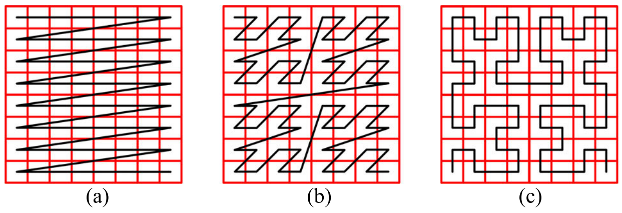


FIGURE 13. Common space-filling curves: (a) line-filling curve, (b) Z-order curve, and (c) Hilbert curve.

We construct three Geo-First models based on the three curves. For the three Geo-First models, the rowkey structure is designed as shown in Fig. 14.

Geolocation code	Attribute code	Unique code

FIGURE 14. The rowkey structure of the three Geo-First models.

The three codes are explained in Section III.A, and that information is not repeated here. For the model based on the line-filling curve, we choose latitude and longitude as the geolocation code. For the model based on the Z-order curve, we choose the Geohash code as the geolocation code. Genhash is a practical application of the Z-order curve that is widely used to index spatiotemporal data [31]–[35]. For the model based on the Hilbert curve, we choose the Google S2 code as the geolocation code.

In this experiment, we compare our method with the three Geo-First models in terms of storage and query efficiency for different numbers of tiles. We use query time and storage time as metrics for the comparison. In addition, to compare the stability of query efficiency of the four models in different query areas, we select 5 regions with aspect ratios of 16:1, 4:1, 1:1, 1:4, and 1:16 as experimental areas. The number of tiles covering each region is 5,000. We use query time as a metric for comparison.

b: EXPERIMENTAL RESULT ANALYSIS

Analysis of Tile Storage and Query Efficiency:

The storage time and query time of the tiles in the four storage models are shown in Fig. 15(a) and (b).

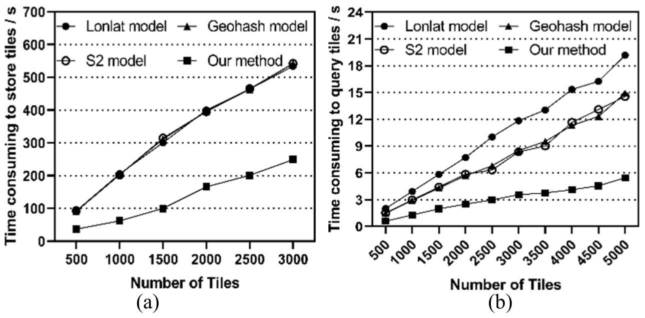


FIGURE 15. The (a) storage and (b) query consumption of tiles in the four storage models.

In terms of data storage efficiency, the three Geo-First models perform similarly, as shown in Fig. 15(a), because the tiles in the three models are stored sequentially in the nodes according to the lexicographic order of the geolocation code. The steps for storing and transferring the tiles in the cluster are almost identical, as reflected in Fig. 1; thus, the storage efficiencies of the three models are also similar. In Fig 15(b), in terms of data query efficiency, the Lonlat model performs the worst, while the Geohash and Google S2 models have similar query efficiencies. The spatial correlation performance of the Z-order curve and the Hilbert curve are similar, but both are better than the line-filling curve; thus, the Geohash and Google S2 models based on these two curves are similar but better than the Lonlat model. Our method has both high query and storage efficiency, as Fig. 15 shows. In terms of data storage, the storage efficiency of our method is approximately 2.6 times better than that of the three Geo-First models. In terms of the data query, the query efficiency of our method is approximately 3.3 times better than that of the Lonlat model and approximately 2.5 times higher than that of the Geohash model and the S2 model. Unlike the three Geo-First models, our method uses BPS and PSS. As introduced in Sections III.B and III.C, these strategies improve the storage and query efficiency of tiles, and the experimental results verified this hypothesis.

Analysis of the Stability of Query Efficiency:

In query areas with different aspect ratios, the query efficiencies of the four storage models are shown in Fig. 16.

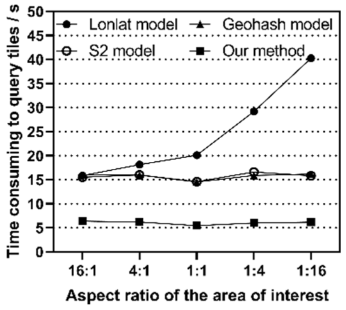


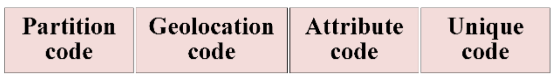
FIGURE 16. The query time of the four storage models in the query area with different aspect ratios.

As shown in Fig. 16, the stability of the query efficiency of the Lonlat model based on the line-filling curve is the worst, and it is closely related to the aspect ratio of the query area. The other two Geo-First models and our proposed method are barely affected by the aspect ratio of the query area. This result occurs because the Lonlat model is implemented based on the line-filling curve, which has poor spatial proximity. Therefore, the query efficiency is greatly affected by the aspect ratio of the query area. The other models are based on the Z-order curve or the Hilbert curve, which produce good spatial neighbours. Therefore, the query efficiency of the data is not easily affected by the aspect ratio of the query area. Our method has the highest query efficiency because it uses both BPS and PSS, which promote tile query efficiency.

2) BPS EVALUATION EXPERIMENT

a: EXPERIMENTAL METHOD

This experiment evaluates the performance differences between storage models with and without BPS in terms of tile storage and query efficiency, node load, data skew, and HBase's ability to resist pressure. The rowkey structure of the former is shown in Fig. 17 and that of the latter is shown in Fig. 14.



Partition code	Geolocation code	Attribute code	Unique code

FIGURE 17. The rowkey structure of the storage model with BPS.

We evaluated the performance of the two storage models at different scales and used query time and storage time as the metrics for comparison. We reflect the load of a node by counting the number of requests received by that node. Data skew is described based on the size of the storefiles stored by the nodes. We define HBase's resistance to pressure as the maximum number of concurrent requests that HBase can withstand, and we calculate the standard deviation R_{SD} of the number of requests among nodes.

b: ANALYSIS OF THE EXPERIMENTAL RESULTS

Tile Storage and Query Efficiency Analysis:

The storage and query times of the tiles in the two storage models are shown in Fig. 18.

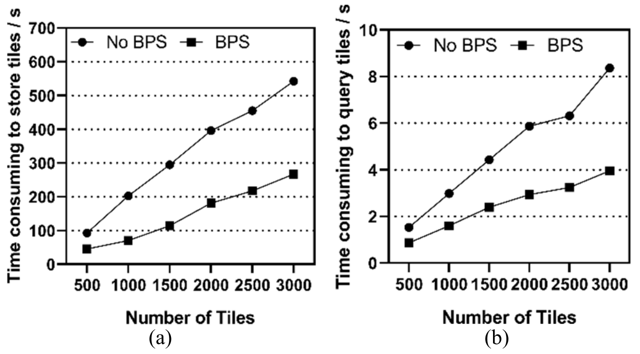


FIGURE 18. The (a) storage and (b) query times of tiles in the two storage models.

As shown in Fig. 18(a) and (b), in terms of data query and storage efficiency, the storage model using the BPS strategy has higher efficiency. On average, after applying BPS, the storage efficiency of the data increases by approximately 1.2 times, while the query efficiency approximately doubles. As described in Section III.B, the BPS strategy changed the distribution characteristics of the tiles and evenly distributed the data storage and query pressure on each node, making full use of the storage and computing resources of the cluster. Therefore, the data storage and query efficiency is improved.

Analysis of Node Load:

The storage load pressure experienced by a node directly determines the future data query load pressure. We counted the number of requests received by each node with stored tiles and calculated the standard deviation and range of the number of node requests on this basis, as shown in Fig. 19.

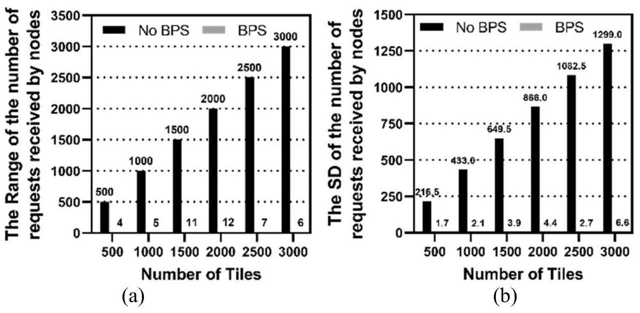


FIGURE 19. Load differences between nodes in the two storage models: (a) range of the number of requests; (b) standard deviation of the number of requests.

From Fig. 19(a) and (b), after applying BPS, the standard deviation and range of the number of requests among nodes are smaller and more stable. Thus, BPS is effective and helps to ensure that the load among nodes is balanced. Thus, under the BPS strategy, the numbers of requests accepted by the nodes in the cluster are similar, indicating that the load pressure on the nodes is largely balanced, and hotspotting does not occur during data storage and querying. BPS transfers the cluster load pressure from only a few nodes to all nodes, which greatly improves the data query and storage efficiency, as confirmed by Fig. 18.

Data Skew Analysis:

Storefiles are the basic form of data storage in HBase. The data skew among nodes is reflected by the differences in the sizes of the storefiles stored by the nodes. Therefore, we use the standard deviation and range of the sizes of the storefiles stored by the nodes to represent the data skew. The results are visualized in Fig. 20.

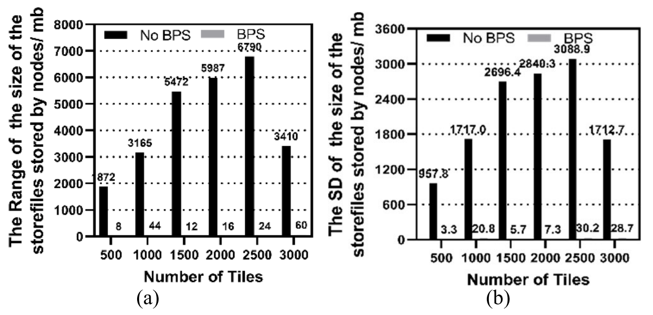


FIGURE 20. **The gaps between the sizes of storefiles stored by nodes in the two storage models: (a) size range of the storefiles stored by nodes in the two storage models; (b) standard deviation of storefile sizes stored by nodes in the two storage models.**

From Fig. 20(a) and (b), after using BPS, the range and standard deviation of the size of the storefiles stored in each node are smaller and more stable. The experimental results show that BPS can effectively avoid large differences in the amount of data stored among nodes and solve the problem of data skew. This improvement is possible because BPS subjectively controls the node in which the tile is stored to ensure that the number of tiles stored by each node is basically the same. In terms of impact, after using BPS, the amount of data stored by each node is also basically the same, which alleviates load imbalance and improves data query and storage efficiency. These conclusions are verified in Figs. 18 and 19.

Analysis of HBase's Ability to Resist Pressure:

The maximum number of concurrent requests (Rmc) that a database can receive is an important indicator of the database's ability to resist pressure. We use various methods to query the tiles and design different numbers of query tiles to determine the changes in the anti-stress performance of the cluster before and after using BPS. We calculate the standard deviation of the number of requests received by the nodes (Rsd) and the state of the nodes. Finally, we found the maximum number of concurrent requests that the two storage models can withstand, as shown in Table 4. Fig. 21 is based on Table 4.

According to Table 4, in the concurrent query scenario, the storage model using BPS achieves superior query performance. The number of requests received by the nodes

TABLE 4. **Concurrent query performance and anti-stress performance in the two storage models.**

Number of tiles	NO BPS R_{SD}	NO BPS Node status	BPS R_{SD}	BPS Node status
10	4.33	ONLINE	0.50	ONLINE
20	8.66	ONLINE	0	ONLINE
30	12.99	ONLINE	0.50	ONLINE
40	17.32	ONLINE	0	ONLINE
50	21.65	ONLINE	0.50	ONLINE
60	25.98	ONLINE	0	ONLINE
70	NULL	ALL DEAD	0.50	ONLINE
80	NULL	ALL DEAD	0	ONLINE
90	NULL	ALL DEAD	0.50	ONLINE
100	NULL	ALL DEAD	0	ONLINE
150	NULL	ALL DEAD	0.50	ONLINE
200	NULL	ALL DEAD	0.70	ONLINE
250	NULL	ALL DEAD	0.50	ONLINE
300	NULL	ALL DEAD	NULL	ALL DEAD

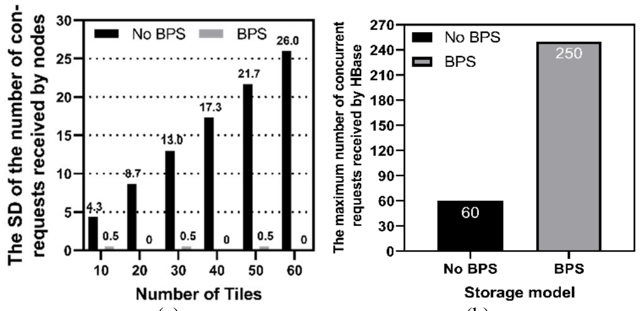


FIGURE 21. **Query performance and anti-stress performance in the two storage models: (a) standard deviation of the number of requests; (b) maximum concurrent requests that HBase can receive.**

remains the same after using BPS, which is in sharp contrast to the number of requests before using BPS, as shown in Fig. 21(a). The maximum number of concurrent requests (Rmc), that HBase can withstand before and after using BPS are 60 and 250, respectively. Without BPS, when the Rmc exceeds 60, nodes will successively die, and the entire cluster will eventually become paralysed. When using BPS, the Rmc reveals that the cluster can withstand approximately 4 times the number of requests as a cluster without BPS—which occurs because each cluster has 4 nodes. BPS evenly distributes the cluster load across each node, which improves the cluster resistance. Therefore, under the same hardware configuration, BPS effectively improves cluster query performance and increases its pressure resistance.

3) PSS EVALUATION EXPERIMENT

a: EXPERIMENTAL METHOD

This experiment evaluates the performance differences in the storage models with and without PSS from two aspects: tile storage and query efficiency, and image storage and query efficiency. The former's rowkey structure is shown in Fig. 22, and the latter's rowkey structure is shown in Fig. 14.

Period code	Geoloaction code	Attribute code	Unique code

FIGURE 22. **The rowkey structure of the storage model with PPS.**

We evaluated the storage and query performance of the two storage models in terms of the amount of data at different scales. We use the storage and query times for comparison. In addition, to evaluate the difference in efficiency of the two storage models in practical applications, we compared the storage and query times for a GF1 remote sensing image.

b: EXPERIMENTAL RESULT ANALYSIS

Analysis of the Tile Storage and Query Efficiency:

Fig. 23(a) and (b) shows the time consumption required for storage and queries.

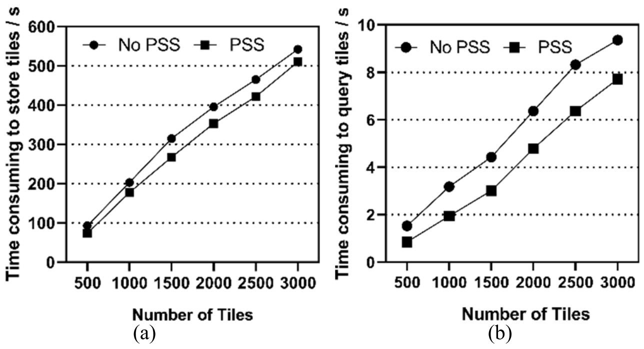


FIGURE 23. **The (a) storage and (b) query consumption of tiles in the two storage models.**

According to Fig. 23(a) and (b), in terms of data query and storage efficiency, the storage model using the PSS strategy has higher efficiency. On average, after applying PSS, the data storage efficiency increases by approximately 15 percent, and the query efficiency increases by approximately 45 percent. As described in Section III.C, PSS causes changes in the distribution patterns of the tiles within a node. Storing and querying target tiles require interactions with fewer data blocks, increasing the storage and query efficiency of the tiles.

Analysis of the Storage and Query Efficiency of an Image: Fig. 24 shows the store and query time consumption of a GF1 remote sensing image using the two storage models. The storage times for the image in the two storage models are shown in Fig. 24(a), and Fig. 24 (b) shows the query times for the image in the two storage models. Notably, in the traditional image storage method, an image query needs to obtain only a record in the database. To store remote sensing images as tiles, we need to traverse all the tiles in the image when querying and downloading the image. Therefore, the query time of an image in this experiment is the total time taken to traverse all the tiles in the image.

According to Fig. 24(a), under the scenario where the entire GF1 image needs to be stored, the storage model using PSS has higher storage efficiency because, after using PSS,

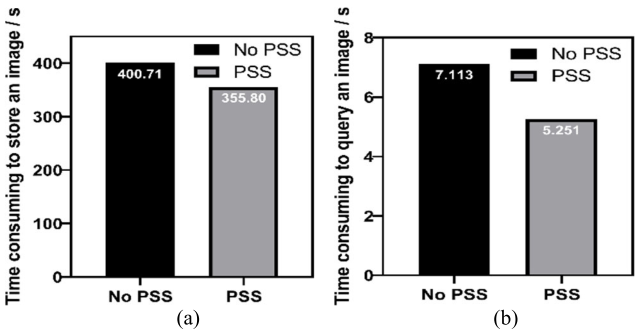


FIGURE 24. **The (a) storage and (b) query consumption of a GF1 image in the two storage models.**

the number of data blocks that must be interacted with to store the GF1 image is greatly reduced, which improves the image storage efficiency, as hypothesized in Section III.C. Compared with the storage model without PSS, the storage time of the image can be reduced by approximately 45 s. Similarly, according to Fig. 24(b), when querying all the tiles in the GF1 image, the storage model using PSS has higher query efficiency. Compared with the storage model without PSS, the image query time is reduced by approximately 2 s. These experimental results show that PSS effectively improves the storage and query efficiency both for tiles and for an entire image.

V. CONCLUSION

We propose a method for storing remote sensing images based on HBase and Google S2. Unlike the Geo-First model, our method considers both load imbalance among nodes and the negative impact of accumulating time-series remote sensing images on tile storage and query efficiency. To solve these two problems, we designed two strategies: BPS and PSS.

First, we evaluated the differences in the tile storage and query efficiency between our method and three common Geo-First models. Then, we evaluated the specific effects of BPS and PSS. The results show that our method has the highest storage and query efficiency compared with the other tested storage models. In terms of storage efficiency, our approach requires approximately 2.6 times less space than do the three Geo-First models, and its query efficiency is approximately 3.3 greater that of the Lonlat model and 2.5 times that of the Geohash and S2 models. Specifically, the results show that BPS is effective at balancing the load among nodes, eliminates data skew and hotspotting, and improves the system's ability to resist stress by approximately 4 times. Overall, BPS increases the data storage and query efficiency by approximately 1.2 and 1 times, respectively. PSS optimizes the storage model of the tiles in each node, which increases the storage efficiency of the data by approximately 15 percent and improves the query efficiency by approximately 45 percent. The storage and query efficiency for entire images is also improved. Our approach reduces the query time for a GF1 image by approximately 2 s and the storage time by approximately 45 s.

In future work, we plan to apply the proposed remote sensing image storage method to other distributed databases. Furthermore, our future research will focus on lost tile reconstruction and mining fine-grained remote sensing data.

ACKNOWLEDGMENT

(Xujin Wang and Rui Wang contributed equally to this work). The authors would like to thank the anonymous reviewers for their valuable comments.

REFERENCES

[1] L. I. Deren, L. P. Zhang, and G. Xia, ''Automatic analysis and mining of remote sensing big data,'' *Acta Geodaetica Cartographica Sinica*, vol. 43, no. 12, pp. 1211–1216, Dec. 2014.

[2] X. Lü, C. Cheng, J. Gong, and L. Guan, ''Review of data storage and management technologies for massive remote sensing data,'' *Sci. China Technol. Sci.*, vol. 54, no. 12, pp. 3220–3232, Dec. 2011.

[3] J. F. Qing, ''Research on efficient management of massive remote sensing image data,'' M.S. thesis, College Resour. Environ. Sci., Hunan Norma Univ., Changsha, China, 2019.

[4] Y. K. Zheng, ''The research and implementation of remote sensing data storage and management system based on Hadoop,'' M.S. thesis, College Softw. Eng., Southeast Univ., Nanjing, China, 2016.

[5] Y. Hajjaji and I. R. Farah, ''Performance investigation of selected NoSQL databases for massive remote sensing image data storage,'' in *Proc. 4th Int. Conf. Adv. Technol. Signal Image Process.*, Sousse, Tunisia, Mar. 2018, pp. 1–6.

[6] Z. Xiao and Y. Liu, ''Remote sensing image database based on NOSQL database,'' in *Proc. 19th Int. Conf. Geoinform.*, Shanghai, China, Jun. 2011, pp. 1–5.

[7] C. Li and W. Yang, ''The distributed storage strategy research of remote sensing image based on Mongo DB,'' in *Proc. 3rd Int. Workshop Earth Observ. Remote Sens. Appl.*, Changsha, China, Jun. 2014, pp. 101–104.

[8] X. R. Wang, Q. G. Yang, and F. Chen, ''Storage model design and implementation of high resolution and hyperspectral remote sensing image based on NoSQL,'' *Earth Sci.*, vol. 40, no. 8, pp. 1420–1426, Aug. 2015.

[9] J. Weipeng, T. Dongxue, C. Guangsheng, and L. Yiyuan, ''Research on improved method of storage and query of large-scale remote sensing images,'' *J. Database Manage.*, vol. 29, no. 3, pp. 1–16, Jul. 2018.

[10] W. Jing and D. Tian, ''An improved distributed storage and query for remote sensing data,'' *Procedia Comput. Sci.*, vol. 129, pp. 238–247, Jan. 2018.

[11] Z. Zhou and Z. Huo, ''Information intelligent management system based on Hadoop,'' *Wireless Pers. Commun.*, vol. 102, no. 4, pp. 3803–3812, Oct. 2018, doi: 10.1007/s11277-018-5411-4.

[12] L. Qiu, Q. Zhu, Z. Du, M. Wang, and Y. Fan, ''An on-demand retrieval method based on hybrid NoSQL for multi-layer image tiles in disaster reduction visualization,'' *ISPRS Int. J. Geo-Inf.*, vol. 6, no. 1, p. 8, 2017.

[13] H. Wang, X. Tang, S. Shi, and F. Ye, ''Research on the construction of data management system of massive satellite images,'' in *Proc. Int. Workshop Big Geospatial Data Data Sci. (BGDDS)*, Wuhan, China, Sep. 2018, pp. 1–4.

[14] R. Rajak, D. Raveendran, M. C. Bh, and S. S. Medasani, ''High resolution satellite image processing using Hadoop framework,'' in *Proc. IEEE Int. Conf. Cloud Comput. Emerg. Markets*, Nov. 2015, pp. 16–21.

[15] J. R. Chen and J. J. Le, ''Reviewing the big data solution based on Hadoop ecosystem,'' *Comput. Eng. Sci.*, vol. 35, no. 10, pp. 25–35, Oct. 2013.

[16] *HBase*. Accessed: 2010. [Online]. Available: http://hbase.apache.org/

[17] L. I. Deren, ''Towards the development of remote sensing and GIS in the 21st century,'' *Geomatics Inf. Sci. Wuhan Univ.*, vol. 28, no. 2, pp. 127–131, Feb. 2003.

[18] S. M. Huo, ''Research on the key techniques of massive image data management based on Hadoop,'' M.S. thesis, Nat. Univ. Defense Technol., Changsha, China, 2010.

[19] L. Wang, C. Cheng, S. Wu, F. Wu, and W. Teng, ''Massive remote sensing image data management based on HBase and GeoSOT,'' in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Milan, Italy, Jul. 2015, pp. 4558–4561.

[20] Q. Li, Y. Lu, X. Gong, and J. Zhang, ''Optimizational method of HBase multi-dimensional data query based on Hilbert space-filling curve,'' in *Proc. 9th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.*, Guangdong, China, Nov. 2014, pp. 469–474.

[21] Y. Zhong, J. Han, and J. Fang, ''An adaptive hierarchical caching scheme for remotely sensed database,'' in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Melbourne, VIC, Australia, Jul. 2013, pp. 604–607.

[22] Y. Q. Cheng, ''Research on Remote sensing data storage technology based on distributed system,'' M.S. thesis, School Inf. Comput. Eng., Northeast Forestry Univ., Harbin, China, 2018.

[23] W. Q. Li, J. C. Xie, J. X. Li, and L. Li, ''Storage system on the remote sensing tile data based on big data technology in suddenwater pollution incident,'' *Comput. Syst. Appl.*, vol. 25, no. 2, pp. 31–37, Feb. 2016.

[24] K. Fu, ''Study on the optimization of large-scale unmanned aerial vehicle remote sensing data processing based on cloud computing,'' M.S. thesis, School Mech., Elect. Inf. Eng., Shandong Univ., Tsinan, China, 2018.

[25] M. G. Cao, X. D. Gao, and Y. Q. Chen, ''HBase-based big data storage of remote sensing tile data for forest fire protection,'' *J. Northeast Forestry Univ.*, vol. 46, no. 02, pp. 35–39, Feb. 2018.

[26] J. Y. Fan, ''The key techniques of cloud GIS based on Hadoop,'' Ph.D. dissertation, College Geomatics, Inf. Eng. Univ., Zhengzhou, China, 2013.

[27] X. Liu, X. Pandeng, Z. Guobin, and L. Xue, ''Parallel and distributed retrieval of remote sensing image using HBase and MapReduce,'' *Geography Geo-Inf. Sci.*, vol. 30, no. 5, pp. 26–28, May 2014.

[28] Y. Liu, B. Chen, W. He, and Y. Fang, ''Massive image data management using HBase and MapReduce,'' in *Proc. 21st Int. Conf. Geoinform.*, Kaifeng, China, Jun. 2013, pp. 1–5.

[29] *Geohash*. Accessed: 2014. [Online]. Available: http: //en.wikipedia.org/wiki/Geohash/

[30] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, ''Analysis of the clustering properties of the Hilbert space-filling curve,'' *IEEE Trans. Knowl. Data Eng.*, vol. 13, no. 1, pp. 124–141, Jan. 2001.

[31] K. Huang, G. Li, and J. Wang, ''Rapid retrieval strategy for massive remote sensing metadata based on GeoHash coding,'' *Remote Sens. Lett.*, vol. 10, no. 2, pp. 111–119, Feb. 2019.

[32] J.-Y. Jung, G. Heo, and K. Oh, ''Urban zone discovery from smart card-based transit logs,'' *IEICE Trans. Inf. Syst.*, vol. E100.D, no. 10, pp. 2465–2469, Oct. 2017.

[33] N. Guo, W. Xiong, Y. Wu, L. Chen, and N. Jing, ''A geographic meshing and coding method based on adaptive Hilbert-geohash,'' *IEEE Access*, vol. 7, pp. 39815–39825, 2019.

[34] J. Zhang, C. Yang, Q. Yang, Y. Lin, and Y. Zhang, ''HGeoHashBase: An optimized storage model of spatial objects for location-based services,'' *Frontiers Comput. Sci.*, vol. 14, no. 1, pp. 208–218, Feb. 2020.

[35] Y. Zhou, S. De, W. Wang, K. Moessner, and M. Palaniswami, ''Spatial indexing for data searching in mobile sensing environments,'' *Sensors*, vol. 17, no. 6, p. 1427, 2017.

XUJIN WANG received the bachelor's degree from Northeastern University, Shenyang, China, in 2018. He is currently pursuing the master's degree with the School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, China.

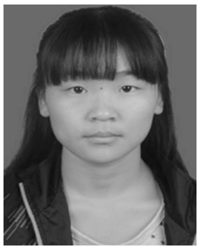
His research interests include remote sensing big data management, high-performance geography, and applications of deep learning in remote sensing.

RUI WANG is currently pursuing the Ph.D. degree in remote sensing and information engineering with Wuhan University, Wuhan, China.

His research interests include the use of big data and remote sensing technology. He has participated in the construction of the Water Responsive Response Remote Sensing Intelligent Platform. He was a recipient of first prize of the Dayu Water Conservancy Science and Technology Award (No. DYJ20150307-G15).

WEN ZHANG received the Ph.D. degree from Wuhan University, Wuhan, China, in 2009.

She is currently a Lecturer with the School of Remote Sensing and Information Engineering, Wuhan University. She has authored more than 10 peer-reviewed articles. Her research interests include network GIS and spatial data analysis.

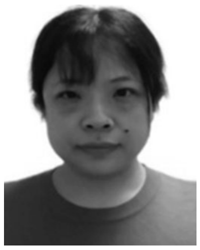


BEIBEI YANG received the master's degree from Wuhan University, Wuhan, China, in 2019, where she is currently pursuing the Ph.D. degree with the School of Remote Sensing and Information Engineering.

Her research interests include remote sensing data management methods, and space-time geographic data mining and analysis.

LINYI LI received the Ph.D. degree in photogrammetry and remote sensing from Wuhan University, Wuhan, China, in 2008.

From January 2014 to January 2015, he was a Visiting Scholar with CSIRO Land and Water, Australia. He is currently an Associate Professor with Wuhan University. His research interests include remote sensing image analysis and applications and computational intelligence algorithms.



FEI CHEN received the master's degree from the Wuhan Digital Engineering Research Institute, Wuhan, China, in 2006.

She is currently with the Wuhan Digital Engineering Research Institute. Her research interests include software modeling, and software service flow arrangement design, command control and simulation.

LINGKUI MENG was a Visiting Professor with the joint Centre of Cambridge-Cranfield for High Performance Computing, Cranfield University, Cranfield, U.K., from 2006 to 2007. He is currently a Professor with the School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, China. His research interests include remote sensing applications in hydrology, cloud computing, and big data analysis.