LARGE VECTOR SPATIAL DATA STORAGE AND QUERY PROCESSING USING CLICKHOUSE

Shuaijun Chen [1], Zhibao Wang [1,2 *], Lu Bai [3], Kunyi Liu [1], Juntao Gao [1], Man Zhao [4], Maurice D. Mulvenna [3]

[1] School of Computer and Information Technology, Northeast Petroleum University, Daqing 163318, China
[2] Bohai-Rim Energy Research Institute, Northeast Petroleum University, Qinhuangdao 066004, China
[3] School of Computing, Ulster University, Belfast BT15 1ED, UK
[4] School of Communication and Electronic Engineering, Qiqihaer University, Qiqihaer 161003, China

KEY WORDS: ClickHouse, vector spatial data, query processing, HBase, remote sensing.

ABSTRACT:

The exponential growth of geospatial data resulting from the development of earth observation technology has created significant challenges for traditional relational databases. While NoSQL databases based on distributed file systems can handle massive data storage, they often struggle to cope with real-time query. Column-storage databases, on other hand, are highly effective at both storage and query processing for large-scale datasets. In this paper, we propose a spatial version of ClickHouse that leverages R-Tree indexing to enable efficient storage and real-time analysis of massive remote sensing data. ClickHouse is a column-oriented, open-source database management system designed for handling large-scale datasets. By integrating R-Tree indexing, we have created a highly efficient system for storing and querying geospatial data. To evaluate the performance of our system, we compare it with HBase, a popular distributed, NoSQL database system. Our experimental results show that ClickHouse outperforms HBase in handling spatial data queries, with a response time approximately three times faster than HBase. We attribute this performance gain to the highly efficient R-Tree indexing used in ClickHouse, which allows for fast spatial data query.

1. INTRODUCTION

In recent years, the advancement of Earth observation technology has led to an increasing importance of remote sensing data in various fields such as urban construction, land resource survey, crop disease analysis, smart city, land use and environmental protection (Chi et al., 2016; Kucherov et al., 2017; Li et al., 2020; Shi et al., 2022; Song et al., 2020; Wang et al., 2022; Zhang et al., 2021; Zhu et al., 2021). The exponential increase in remote sensing data volume presents a considerable obstacle to the efficient organization and management of data. Furthermore, it hinders the capacity to analyse and respond promptly to spatial data queries and analyses. Moreover, as a result of the development of global positioning system and mobile intelligent terminals, high accuracy and low latency requirements have emerged for location services including precise positioning, regional query, route planning. In this era of massive spatial data, the key challenges in GIS development is to effectively organize, manage and store large-scale spatial data.

The use of relational databases for spatial data storage dates back to 1970 (Codd, 1970). Today they are widely used for spatial data storage, with popular databases such as Oracle Spatial and PostGIS for PostgreSQL featuring spatial data analysis capabilities. PostGIS, based on PostgreSQL for spatial expansion, has surpassed Oracle Spatial with query speeds that are 300% and 450% faster for spatial data querying and analysis, respectively (Shukla et al., 2016). PostgreSQL provides a simple development path to incorporate new space types, offering numerous features such as reliability, transaction integrity, support for SQL standards, pluggable type extension, and community-oriented development model, along with support for large GIS objects and R-Tree index as a general index structure. PostgreSQL mainly extends the geometry data type for the storage of spatial data, capable of effectively storing point, line, polygon, multipoint, multiline, and multipolygon. Despite its powerful spatial analysis function, traditional relational databases face significant performance bottlenecks. The efficient storage and access of massive data, high scalability, and availability of the database are the major gaps that a relational database cannot resolve. PostgreSQL struggles with decreasing query and processing efficiency as data volume increases. Additionally, deployment and installation of a single node result in limited data storage capacity and low security, and node failures can lead to data loss.

With the continuous growth of data volume and the arrival of the era of big data, data storage has begun to shift from centralized storage to distributed storage on multiple machines. There are three main types of data modes: (1) Tree-structured storage where data is organized in a folder containing file organization. (2) Block storage where data is grouped into blocks of the same size with each block having its own identifier, and (3) Object storage where data is stored in an object unit, and the metadata describing the data is stored in an independent database. In 2003, Google released its own distributed file storage system, Google File System (GFS), which uses a single master and multiple chunkservers to organize data. GFS divides files into chunks, each with a unique 64-bit identifier, and replicates each data block to multiple chunkservers to improve reliability (Ghemawat et al., 2003). Distributed data storage and replication mechanisms are capable of overcoming the limitations of traditional database security and storage scalability. NoSQL databases can be used on distributed clusters, providing strong scalability and storage capabilities, as well as low-latency query services (Wang et al.,

* Corresponding author

2017). In recent years, there has been extensive exploration of NoSQL database storage for spatial data.

This paper proposes a spatial data storage strategy based on ClickHouse database that utilizes the superior performance of the R-Tree index for spatial data retrieval in traditional relational databases. The R-Tree index is still used as the index of spatial data in ClickHouse. R-Tree is a balanced tree similar to B-Tree but it solves the problem of fast search high-dimensional space, unlike B-Tree, which solves the problem of fast search in one-dimensional space. R-Tree uses the minimum circumscribed rectangle as the boundary of the space geometry, and during the spatial retrieval, it filters out most of the areas that do not overlap with the search rectangle by gradually expanding the boundary (Guttman, 1984). Before storing vector data in ClickHouse, the proposed strategy build the data into an R-Tree, traverses the R-Tree in a hierarchical way during storage, and stores non-leaf nodes in the index table and leaf nodes in the detailed wide table. The data is partitioned based on the root node before storage, so that the non-search areas can be effectively isolated during the query, thereby speeding up the query process.

2. RELATED WORK

In recent years, there has been a surge of interest in using NoSQL databases for managing and storing large-scale datasets. Redis, Elasticsearch, MongoDB and HBase. Compared with these mainstream NoSQL databases, ClickHouse’s architecture is highly scalable, fault-tolerant, and distributed, making it an ideal choice for use cases that require high throughput, low latency, and real-time analytics. In this related work section, we provide an overview of the relevant literature that discusses the use of Redis, Elasticsearch, MongoDB, HBase, and ClickHouse, highlighting their advantages and disadvantages in handling Geospatial data.

2.1 Redis

Redis is an open source, memory-based data structure storage system that supports a variety of use cases, including serving as a database, cache, message broker, and streaming engine. It offers support for various data structures such as strings, hashes, lists, sets, sorted sets, bitmaps, hyperloglogs, and geospatial indexs and streams. Redis has built-in replication, Lua scripting, LRU eviction and different levels of persistence. Additionally, it provides high availability through Redis Sentinel and Cluster. When storing geographical locations, Redis uses corresponding key to store the longitude and latitude and the name of the data using Geo Set data structure, which is similar to the Sorted Set data structure. Its implementation leverages Geohash technology to encode the longitude and latitude bits in a staggered manner, and divides the geographical space into grid-shaped buckets with each grid having its own corresponding code (Makris et al., 2019). When querying a spatial range using the georadius method, it transforms the query of two-dimensional space into the comparison of strings of one-dimensional space, which makes it highly efficient for spatial range queries (Liu et al., 2014). (Hao Yu et al., 2012) proposed to use Redis to cache terabytes of geospatial data to cope with highly concurrent queries. However, Redis does not provide data security guarantees and improper design can lead to cache breakdown, cache penetration and cache avalanche.

2.2 Elasticsearch

Elasticsearch is a distributed, real-time search and data analysis engine that is highly scalable. It can perform full-text search, structured search, and analysis, making it a versatile tool for many use cases. Geographical location is represented in Elasticsearch using two different data types: geo_point and geo_shape. The former is used to represent geographical coordinate points with latitude and longitude, while the latter is used for complex geographical shapes using GeoJSON. Geo_points is useful for finding points within a certain range, calculating distances, and aggregating data display on the maps. On the other hand, geo_shape is used to filter the data and analyse whether two shapes intersect, contain or overlap in the geographical space. Elasticsearch uses GeoHash indexing for geographical location queries, ensuring high accuracy. One example of using Elasticsearch is for retrieving geographical location in digital corpus (Bartlett, 2019).

2.3 MongoDB

MongoDB database is a document-based database that organizes data in the form of BJSON. It offers high-performance data persistence, and an API query interface that allows for easy integration with external programs for CRUD operations. MongoDB is highly reliable and offers failover and redundancy mechanisms to prevent data loss as well as horizontal scalability through data partitioning across multiple machines. It supports multiple storage engines, including memory storage engine and WiredTiger storage engine. To efficiently retrieve geospatial coordinates, MongoDB employs two indexing methods: the 2D index for planar geometry and 2dsphere index for spherical geometry. GeoJSON format must be used for organizing geospatial data in MongoDB due to these indexing methods. MongoDB also provides the calculation of geospatial relationships such as geoWithin for inclusion, geoIntersects for intersection, and nearSphere for adjacency. Recent studies have explored different methods of storing remote sensing data in MongoDB. For instance, a remote sensing data management method was proposed based on MongoDB that stores metadata as documents, and image data in GridFS format (Wang et al., 2019) .

2.4 HBase

HBase is a highly scalable NoSQL database that enables the storage of massive amounts of data in a distributed environment. Leveraging the Hadoop Distributed File System (HDFS), HBase uses Namespaces for data space division and Regions for data organization, storing data in a columnar format. Rather than defining specific columns when creating a table, HBase only requires a declaration of the Column Family, providing dynamic field specifications when writing data. This makes Hbase more suitable for scenarios with varying fields, when compared to traditional relational databases. Though HBase does not support geospatial data storage natively, its flexible architecture has led many researchers to explore the use of HBase for this purpose. For example, (Wang et al., 2017) proposed a method of storing vector data in HBase by dividing it into grids using space Z curve filling with the number of space filling curves used as RowKey for storage, and the geometric objects within each grid used as columns (Wang et al.,2017). Similarly, (Wang et al., 2019) proposed an efficient spatial big data storage and query method in HBase, which leverages Hilbert spatial curve filling to convert high-dimensional data into one-dimensional data, allowing for faster query processing.

2.5 ClickHouse

Clickhouse is an open-source column-oriented storage database (DBMS) that specializes in online analytical processing (OLAP) queries, allowing real-time generation of data analysis reports through SQL queries. ClickHouse has the following features: (1) ClickHouse supports various table-level storage engine, with over 20 engines in four categories, namely merge tree, log, interface and others enabling users to choose different storage engines according to specific requirements. (2) ClickHouse has a high throughput capacity and uses a structure similar to LSM tree to achieve sequential write performance, allowing it to write data in a sequential append manner during data import, and merge-sort multiple segments to optimize disk usage. ClickHouse’s write performance is superior, with an official open benchmark test showing it can achieve a write throughput of 50MB-200MB/s, equivalent to a write speed of 50W-200W pieces/s (based on an estimated 100 bytes per line). (3) ClickHouse divides the data into multiple partitions and further divides each partition into multiple index granularities, which can then be processed by multiple CPU cores for parallel data processing. This design allows a single query to utilize all the CPU of the entire machine, greatly reduces the query delay. Compared with other databases, ClickHouse has superior performance in various SQL queries, making it an excellent choice for low performance machines and outperforming MySQL which is the most popular open-source database at presently available (Wickramasekara et al., 2020).

3. METHODS

3.1 Storage method of spatial data in ClickHouse

ClickHouse lacks the inherent capability to store geospatial data, but this study builds a logic layer on top of ClickHouse to enable it to do so. This section will focus on the design of this extended logic layer with Figure 1 displaying the comprehensive structure of ClickHouse’s storage and retrieval of geospatial data.

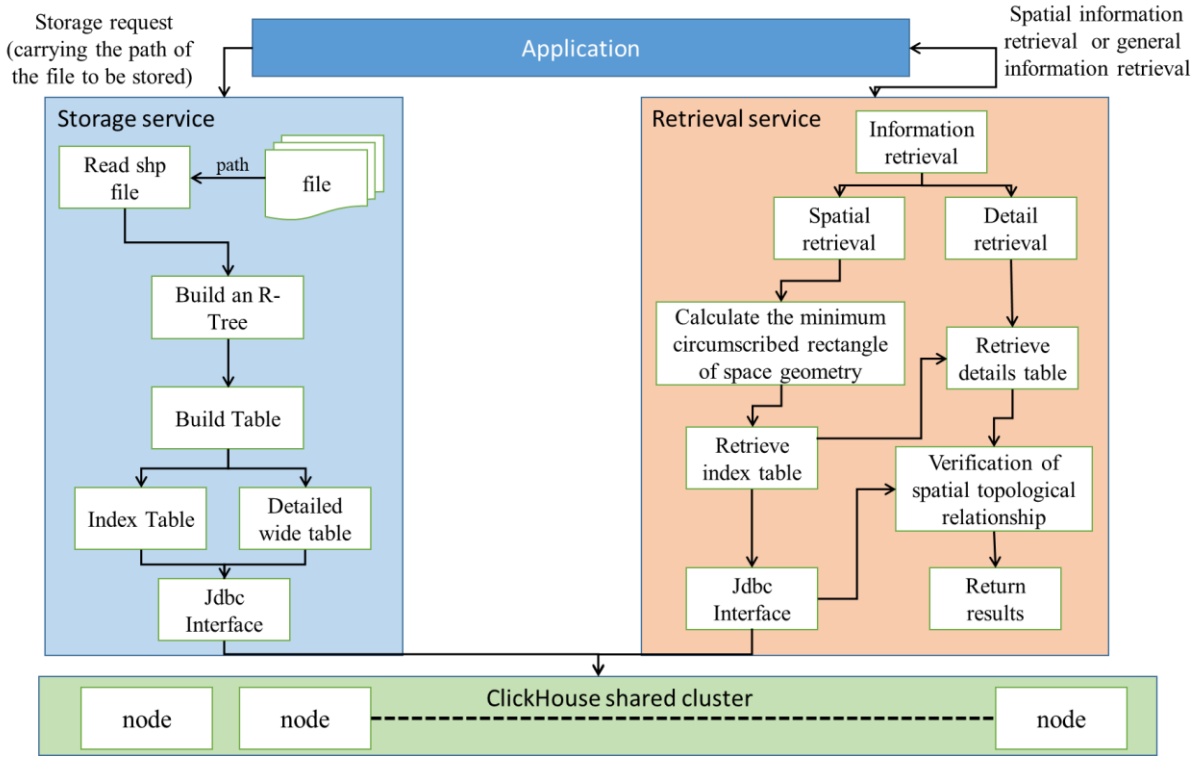

Figure 1. The overall architecture of ClickHouse for storing and retrieving geospatial data.

3.1.1 Storage service: The storage service in the ClickHouse shared cluster for spatial data performs the following operations:
S1: Read the corresponding shapefile file from the corresponding file system based on the path carried in the application request.
S2: Construct an R-Tree using the data from the shapefile file. R-Tree is preferred over the improved R*-Tree due to its proven ability of R-Tree to retrieve geospatial data has been fully proved in traditional relational databases. The construction time of R*-Tree is relatively long, and its query efficiency is not optimal when handling small data amounts.
S3: Generate a separate table for each shapefile since they contain unique geographic information. To accelerate geospatial query performance, two tables are created: an index table and a detailed wide table. The index table stores non-leaf node information, while the detailed wide table stores leaf node information in the R-Tree. Table 1 shows the structure of the index table, while Table 2 shows the structure of the detailed wide table.

id	partition_id	level	parent_id	min_x	max_x	min_y	max_y	is_leaf
971074336477675520	0	0	0	563069.6710008697	586413.1604943711	4483095.741016292	4494680.855637121	false

971074336477675536	0	1	9710743364776755520	563189.8072439035	564977.9116748271	4483095.741016292	4484222.747509402	true
971074336477675537	0	1	9710743364776755520	563069.6710008697	566930.1785916027	4483656.955635705	4484856.839555023	true
971074336477675538	0	1	9710743364776755520	563073.6778494965	568354.4889619449	4484207.59796222	4485157.414893977	true
971074336481869824	0	1	9710743364776755520	563147.7414593172	568336.8224599154	4484451.3144932935	4485683.854511977	true
971074336481869825	0	1	9710743364776755520	563875.0872299344	568345.09513854	4484857.7186132959	4485934.115992783	true
...	...	...	...	...	...	...	...	...

Table 1. Structure and partial data of index table

id	parent_id	min_x	max_x	min_y	max_y	...	the_geom
971074370996797440	971074336477675536	607240.6270169418	607393.3437808634	4508814.176007433	4508929.788199221	...	MULTIPOLYGON (((607393.3437808634 4508853.954510309, 607263.5223237597 4508814.176007433, 607240.6270169418 4508889.966283751, 607370.2550110875 4508929.788199221, 607393.3437808634 4508853.954510309)))
...	...	...	...	...	...	...	...

Table 2. Structure of detailed wide table and partial data.

ClickHouse’s ability to partition data allows for effective isolation of irrelevant data during queries, which is cleverly used when designing the index table. The table is built based on the root node with elements in the root node and the subtree below it stored in the corresponding partition as data of the same partition. The partition number is determined by the left-to-right position of the current element in the root node, while the partition number of the leaf node in the detailed wide table is the ID of the parent node, as shown in Figure 2. The level field records the level of the current node in the entire R-Tree. Since the index data is stored in the database, retrieval needs to be performed recursively with each retrieval taking the ID of the parent node as the filter condition is to isolate unrelated nodes in the same subtree. Fortunately, ClickHouse provides the Prewhere statement, which can filter the data in advance before the selection, and significantly improve efficiency, especially when the number of columns queried is significantly more than that filtered. The minimum bounding rectangle of the current node is defined by the min_x, max_ x, min_y and max_ y values. The is_leaf field identifies whether the current node is the last level of the index node and serves as the exit of the recursive query.

In the detailed wide table, the smallest granularity data of the current shapefile file is stored, with the ID generated by the snowflake algorithm used to identify and sort data. The index of ClickHouse itself is used to speed up retrieval efficiency when querying. The parent_id is used as a partition field, and the data of the same parent node is stored in a partition. The minimum limited rectangle can exclude the data that does not meet the requirements when querying. The_geom field records geometric data, while the remaining fields contain non-spatial attributes of the current data.

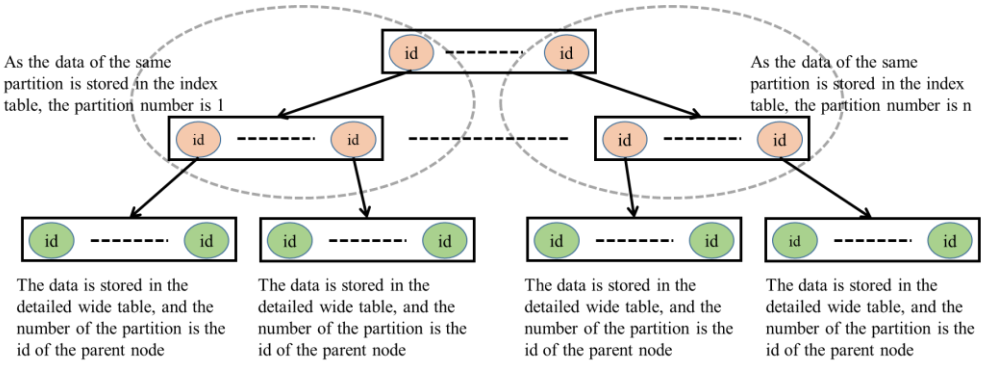

Figure 2. Use R-Tree to build the logic of index table and detail table.

3.1.2 Retrieval service: Data retrieval services can be categorized into geospatial retrieval and conventional retrieval. Geospatial retrieval involves providing spatial geometry information to retrieve data, while conventional retrieval requires general retrieval conditions such as city name, street name, etc. For conventional retrieval, index query is avoided, and the corresponding information is directly queried in the detailed wide table based on the conditions. For example, if the geometric value of a street named "Atlantic commons" is needed, the system will retrieve the relevant information from the detailed wide table and return it to the application. In contrast, geospatial retrieval requires calculating the minimum restricted rectangle of the incoming geometry first. The index table is then retrieved from the root node. When R-Tree is stored in the table, parent_id of the root node is set to 0. Therefore, when searching, the filter condition can be set to parent_id=0 to find the root node. Then, determine the node that intersects with the minimum bounding rectangle of the spatial geometry being queried in the root node. The system then recursively searches the lower level of R-Tree in the index table, taking the id of the parent node as the filter condition at each recursion. When the leaf node of the index is reached, the current node id is used as the filter condition to retrieve the corresponding data in the detailed wide table. During the search in the detailed wide table, the system still uses the intersection of the minimum restricted rectangle as the filter condition. However, the intersection of the minimum restricted rectangle in R-Tree does not necessarily mean that the real geometry is intersected (as shown in Figure 3). Therefore, further judgment is required when the corresponding data is obtained. Finally, when the geometric information is retrieved and the geometric information in the detailed wide table meets the topological requirements of the retrieval, the corresponding results are returned to the upper application.

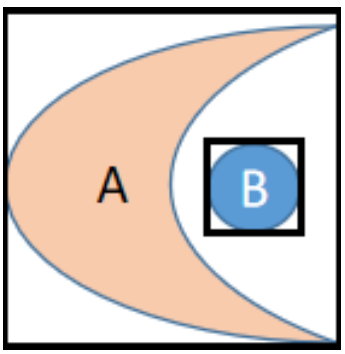

Figure 3. The minimum bounding rectangle of two polygons intersects, but the polygons do not intersect.

3.1.3 ClickHouse Sharded Cluster: ClickHouse provides two mechanisms for distributed data storage: replica and sharded cluster. The replica mechanism ensures the high availability of data by allowing the same data to be obtained from other servers if a ClickHouse node goes down. The write process of the replica mechanism is shown in Figure 4. However, it requires each server to accommodate the full amount of data, limiting horizontal expansion. To overcome this limitation, ClickHouse introduces sharding, which segments a complete piece of data and distributes different shards to different nodes. The Distributed Table Engine acts as a middleware, routing distributed data from multiple nodes with different shards through distributed logical tables. Unlike the replica mechanism, the sharded cluster mechanism allows for horizontal expansion of massive data storage. To ensure both horizontal scalability and high availability, ClickHouse combines sharding and replica mechanisms. In this paper, a sharded cluster architecture is adopted, where three machines are used as the partition cluster. Figure 5 illustrates the design of this architecture. By leveraging sharding and replica mechanisms, ClickHouse provides a powerful solution for distributed data storage and processing.

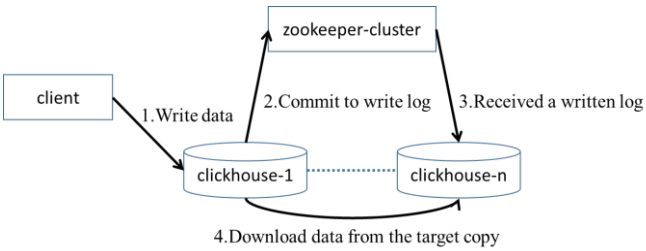

Figure4 . Copy writing process.

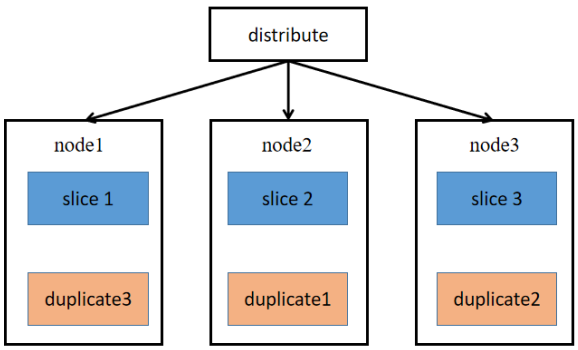

Figure5 . Design architecture of sharded cluster.

3.2 Storage method of spatial data in HBase

To better evaluate the advantages of ClickHouse in querying spatial geographic data, this paper includes a comparative test with Hbase. The experiment expands HBase using R-Tree index while maintaining the independent storage mode for index and detailed wide table to speed up the query. This is achieved by dividing the data into two tables, namely index table and detail table. Table 3 shows the design of the index table while Table 4 shows the design of the detailed wide table.

HBase stores data using a single partition and employs the "parentRowKey_SnowflakesAlgorithm" design rule for Row_Key to ensure its uniqueness. The parameters min_x, max_x, min_y, and max_y represent the smallest circumscribed rectangle, while is_ Leaf identifies whether the current node is a leaf node of the R-Tree index. The Start_Row and Stop_Row parameters specify the range of Row_Key of the child node. The primary purpose of these design features is to accelerate query performance using scan operations. To optimize query performance, HBase stores data for different column families separately. In the design of the detailed wide table, two column families are used: "ordinary" to store non-geometric information and "geometry" to store geometric information. This approach isolates geometric and non-geometric queries, enabling each to be optimized independently. By separating the data into two column families, HBase can accelerate both geometric and non-geometric queries, resulting in faster query performance and better scalability.

	Index_Info						
Row_Key	min_x	max_x	min_y	max_y	is_leaf	Start_Row	Stop_Row
0_97396928709184 7168	563069.671 0008697	586413.16 04943711	4483095.7 41016292	4494680.85 5637121	false	09739692 87091847 168_	09739692870 91847168|
0_97396928709184 7169	586214.691 145163	593074.05 78981118	4488435.7 70831544	4500914.98 72505395	false	09739692 87091847 169_	09739692870 91847169|
0_97396928709184 7170	567299.814 0316624	586522.29 45189208	4492040.8 63678092	4498558.07 9844725	false	09739692 87091847 170_	09739692870 91847170|
0_97396928709184 7171	591493.622 9326547	598798.51 87040224	4489965.5 61497574	4506856.94 2924007	false	09739692 87091847 171_	09739692870 91847171|
...	...	...	...	...	...	...	...

Table 3. Design of index table in HBase.

	ordinary	geometry				
rowkey	...	min_x	max_x	min_y	max_y	the_geom
09739692870918471689 73969287091847204_0	...	573351.878 0943268	573878.107 750165	4487638.4 6718892	4488463.03 9496107	MULTIPOLYGON (((572938.2665684235 4486928.674520253, 572933.6947068771 4486936.54728656, 572929.7565177245 4486944.75539434, 572926.4764236218 4486953.247941074, 572923.8747660567 4486961.97226028, 572921.9676791976)))
...	...	...	...	...	...	...

Table 4. Design of Detail Table in HBase.

4. EXPERIMENT AND RESULTS

This section is based on real world data and aims to provide a comprehensive evaluation of the experimental systems. To achieve this, section 4.1 presents the experimental environment used in the study, including details about the hardware and software configurations. Section 4.2 provides a detailed description of the experimental dataset. Finally, in section 4.3, the experimental results are presented and compared to provide insights into the strengths and weaknesses of the systems under test. By using real-world data and a rigorous experimental methodology, this study aims to provide valuable insights into the performance and scalability of the experimental systems, which can be used to inform decisions about their use in real-world applications.

4.1 Experiment setting

The experiments were performed on a virtual machine environment with a 4GB RAM and 2.80GHz Intel(R) Core(TM) i7-7700HQ CPU. ClickHouse was deployed in a single-node storage configuration, while HBase was set up as a three-node cluster with identical hardware and software configurations. Table 5 provides detailed specifications for each of the nodes in the HBase cluster.

CPU type	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Processor number	3
RAM	4G
ClickHouse Version	21.7.3.14
Hadoop version	3.1.3
Zookeeper version	3.5.7
HBase version	1.3.1

Table 5. Node configuration for HBase

4.2 Dataset

To evaluate the performance and scalability of the systems under test, a range of real-world datasets were used in the experiments. Specifically, the datasets included community data, census data for New York (“New York Census Data,” n.d.), building data, and land data for Papua New Guinea in 2018 (“Papua New Guinea Data,” n.d.). The size and characteristics of each dataset are summarized in Table 6, which provides details such as the number of records, file size, and data format. By using real-world datasets, this study aims to provide insights into the systems’ ability to handle complex and diverse data types, as well as their performance and scalability under realistic conditions.

Name	Description	Amount of data
nyc_neighborhoods	New York Community Data	129
gis_osm_landuse_a_free_1	Land Data in Papua New Guinea	34700
nyc_census_blocks	New York 2018 Census Data	38700

gis_osm_buildings_a_free_1	Building Data in Papua New Guinea	369254

Table 6. Experimental dataset

4.3 Experiment results

To compare the efficiency of spatial queries between ClickHouse and HBase for different data volumes and query rates, we randomly selected 1% of the data from the tables for cross-checking during the experiments. The results, as shown in Figure 6 indicate that ClickHouse outperforms HBase in terms of query speed, particularly for large amounts of data. When the data volume in a single table data was 129 records, HBase's response time for a data query reached the second level, while ClickHouse returned the query results in milliseconds. When the data volume in a single table increased to 10,000 records, HBase's response time was about three times longer than ClickHouse's. Furthermore, with the increase in the volume of data, HBase's response time increased significantly. These findings demonstrate that ClickHouse is more efficient than HBase is spatial data query, especially for larger datasets.

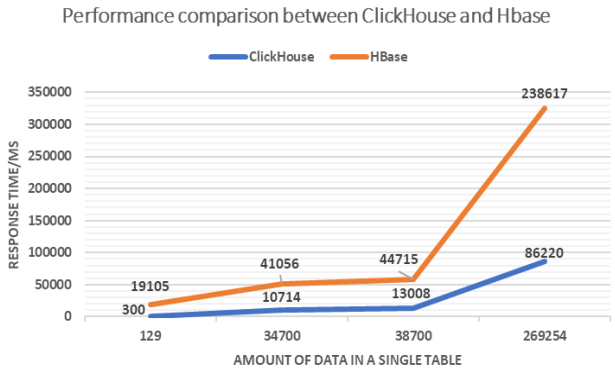

Figure 6. The minimum bounding rectangle of two polygons intersects, but the polygons do not intersect.

5. CONCLUSIONS

This paper proposes a storage model for spatial data using ClickHouse and R-Tree as an index for spatial queries. To evaluate its performance, we compare the geospatial data queries with the HBase database using the same indexes. The results of the experiments demonstrate that ClickHouse outperforms HBase in geospatial data queries, even when using the same indexes.

REFERENCES

Bartlett, R., 2019. Local geographic information storing and querying using Elasticsearch, in: Proceedings of the 13th Workshop on Geographic Information Retrieval. Presented at the GIR'19: 13th Workshop on Geographic Information Retrieval, ACM, Lyon France, pp. 1–4. https://doi.org/10.1145/3371140.3371144

Chi, M., Plaza, A., Benediktsson, J.A., Sun, Z., Shen, J., Zhu, Y., 2016. Big Data for Remote Sensing: Challenges and Opportunities. Proc. IEEE 104, 2207–2219. https://doi.org/10.1109/JPROC.2016.2598228

Codd, E.F., 1970. A relational model of data for large shared data banks. Commun. ACM 13, 377–387. https://doi.org/10.1145/362384.362685

Ghemawat, S., Gobioff, H., Leung, S.-T., 2003. The Google file system, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. Presented at the SOSP03: ACM Symposium on Operating Systems Principles, ACM, Bolton Landing NY USA, pp. 29–43. https://doi.org/10.1145/945445.945450

Guttman, A., 1984. R-trees: a dynamic index structure for spatial searching. ACM SIGMOD Rec. 14, 47–57. https://doi.org/10.1145/971697.602266

Hao Yu, Yuehu Liu, Chuan Tian, Liang Liu, Mingchao Liu, Yong Gao, 2012. A cache framework for geographical feature store, in: 2012 20th International Conference on Geoinformatics. Presented at the 2012 20th International Conference on Geoinformatics, IEEE, Hong Kong, China, pp. 1–4. https://doi.org/10.1109/Geoinformatics.2012.6270288

Jiajun Liu, Haoran Li, Yong Gao, Hao Yu, Dan Jiang, 2014. A geohash-based index for spatial data management in distributed memory, in: 2014 22nd International Conference on Geoinformatics. Presented at the 2014 22nd International Conference on Geoinformatics, IEEE, Kaohsiung, Taiwan, pp. 1–4. https://doi.org/10.1109/GEOINFORMATICS.2014.6950819

Kucherov, B., Pribyl, O., Artyushenko, V., 2017. Increasing efficiency of getting results of satellite remote sensing for smart cities, in: 2017 Smart City Symposium Prague (SCSP). Presented at the 2017 Smart City Symposium Prague (SCSP), IEEE, Prague, Czech Republic, pp. 1–6. https://doi.org/10.1109/SCSP.2017.7973854

Li, J., Pei, Y., Zhao, S., Xiao, R., Sang, X., Zhang, C., 2020. A Review of Remote Sensing for Environmental Monitoring in China. Remote Sens. 12, 1130. https://doi.org/10.3390/rs12071130

Makris, A., Tserpes, K., Anagnostopoulos, D., Nikolaidou, M., de Macedo, J.A.F., 2019. Database system comparison based on spatiotemporal functionality, in: Proceedings of the 23rd International Database Applications & Engineering Symposium on - IDEAS '19. Presented at the the 23rd International Database Applications & Engineering Symposium, ACM Press, Athens, Greece, pp. 1–7. https://doi.org/10.1145/3331076.3331101

New York Census Data [WWW Document], n.d. URL http://postgis.net/workshops/zh_Hans/postgis-intro/about_data.html

Papua New Guinea Data [WWW Document], n.d. URL http://download.geofabrik.de/australia-oceania/

Shi, K., Bai, L., Wang, Z., Tong, X., Mulvenna, M.D., Bond, R.R., 2022. Photovoltaic Installations Change Detection from Remote Sensing Images Using Deep Learning, in: IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium. Presented at the IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, IEEE, Kuala Lumpur, Malaysia, pp. 3231–3234. https://doi.org/10.1109/IGARSS46834.2022.9883738

Shukla, D., Shivnani, C., Shah, D., 2016. Comparing oracle spatial and postgres PostGIS. IJCSC 7, 95–100.

Song, G., Wang, Z., Bai, L., Zhang, J., Chen, L., 2020. Detection of oil wells based on faster R-CNN in optical satellite remote sensing images, in: Notarnicola, C., Bovenga, F., Bruzzone, L., Bovolo, F., Benediktsson, J.A., Santi, E., Pierdicca, N. (Eds.), Image and Signal Processing for Remote Sensing XXVI. Presented at the Image and Signal Processing for Remote Sensing XXVI, SPIE, Online Only, United Kingdom, p. 17. https://doi.org/10.1117/12.2572996

Wang, H., Zhu, Y., Wang, J., Han, H., Niu, J., Chen, X., 2022. Modeling of spatial pattern and influencing factors of cultivated land quality in Henan Province based on spatial big data. PLOS ONE 17, e0265613. https://doi.org/10.1371/journal.pone.0265613

Wang, S., Li, G., Yao, X., Zeng, Y., Pang, L., Zhang, L., 2019. A Distributed Storage and Access Approach for Massive Remote Sensing Data in MongoDB. ISPRS Int. J. Geo-Inf. 8, 533. https://doi.org/10.3390/ijgi8120533

Wang, Y., Li, C., Li, M., Liu, Z., 2017. HBase storage schemas for massive spatial vector data. Clust. Comput. 20, 3657–3666. https://doi.org/10.1007/s10586-017-1253-1

Wickramasekara, A., Liyanage, M.P.P., Kumarasinghe, U., 2020. A comparative study between the capabilities of MySQL and ClickHouse in low-performance Linux environment, in: 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer). Presented at the 2020 20th International Conference on Advances in ICT for Emerging Regions (ICTer), IEEE, Colombo, Sri Lanka, pp. 276–277. https://doi.org/10.1109/ICTer51097.2020.9325483

Zhang, J., Wang, Z., Bai, L., Song, G., Tao, J., Chen, L., 2021. Deforestation Detection Based on U-Net and LSTM in Optical Satellite Remote Sensing Images, in: 2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS. Presented at the IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium, IEEE, Brussels, Belgium, pp. 3753–3756. https://doi.org/10.1109/IGARSS47720.2021.9554689

Zhu, M., Wang, Z., Bai, L., Zhang, J., Tao, J., Chen, L., 2021. Detection of industrial storage tanks at the city-level from optical satellite remote sensing images, in: Bruzzone, L., Bovolo, F., Benediktsson, J.A. (Eds.), Image and Signal Processing for Remote Sensing XXVII. Presented at the Image and Signal Processing for Remote Sensing XXVII, SPIE, Online Only, Spain, p. 33. https://doi.org/10.1117/12.2600008