Research on Cassandra-Based Distributed Storage and Retrieval of Massive MUSER Data (Postprint)

Authors: Shi Yue, Wang Feng, Li Pengcheng, Liu Yingbo

Date: 2018-05-24T00:00:00+00:00

Abstract

The Mingantu Radio Spectral Heliograph generates massive amounts of observational data daily. Traditional relational databases face numerous issues when storing and managing this data, including high read/write latency, limited performance and capacity scalability, and weak availability. To address these problems, we conducted research on the application of NoSQL-based massive data storage and retrieval. First, we conducted a detailed analysis of the data characteristics, storage requirements, and challenges of the Mingantu Radio Spectral Heliograph. Then, we performed data modeling for the instrument, proposing a columnar non-relational data model, and simultaneously presented a synchronized storage method for metadata and data in NoSQL, thereby solving their consistency issues. On this basis, we implemented a massive astronomical data storage management system (MBDMS) based on Cassandra. Finally, experiments verified the efficiency, scalability, and feasibility of the system' s storage and retrieval. The results demonstrate that MBDMS can effectively meet data management needs and represents an effective solution to current data storage problems.

Full Text

Research on Distributed Storage and Retrieval of Massive MUSER Data Based on Cassandra

Shi Yue[1], Wang Feng[1,2], Li Pengcheng[1], Liu Yingbo[1]* [1]Key Laboratory of Computer Technology Application of Yunnan Province, Kunming University of Science and Technology, Kunming, Yunnan 650500, China [2]Yunnan Observatories, Chinese Academy of Sciences, Kunming, Yunnan 650216, China

Abstract

The Mingantu Ultrawide SpEctral Radioheliograph (MUSER) generates massive observational data daily. Traditional relational database management systems face significant challenges when storing and managing this data, including high read/write latency, limited performance and capacity scalability, and weak availability. This paper investigates the application of NoSQL technologies for MUSER data storage and retrieval. We first analyze the characteristics of MUSER data, its storage requirements, and the challenges that must be addressed. We then propose a column-oriented non-relational data model and present a method for synchronously storing metadata and data within a NoSQL framework to resolve consistency issues. Based on these foundations, we implement the MUSER Massive Astronomical Data Management System (MBDMS) using Cassandra. Experimental results validate the efficiency, scalability, and feasibility of the system' s data storage and retrieval capabilities. MBDMS effectively meets the data management requirements of MUSER and provides a viable solution to current MUSER data storage challenges.

Keywords: non-relational database; storage; retrieval; Cassandra

1. Research Status

Massive data storage management represents a critical challenge in astronomy. With the emergence of astronomical big data, research institutions worldwide have begun investigating big data storage solutions. Computer scientists at the University of Washington' s Department of Astronomy employ HDFS for distributed processing of massive astronomical image data, using MapReduce to decompose image datasets into smaller file sequences before system input, significantly improving processing efficiency while reducing total file count. Reference [3] proposes a novel data management technique called negative databases, which leverages complementary sets of observational data to obtain necessary information, enabling efficient management of MUSER data. Reference [4] explores the storage of FITS file header metadata using NoSQL and experimentally demonstrates its feasibility. To meet the high-performance retrieval and query requirements of massive astronomical data, reference [5] proposes a distributed search engine based on ElasticSearch, achieving efficient retrieval of massive FITS data. Reference [6] presents a distributed inverted index built on Cassandra to address scalability issues inherent in traditional relational databases, designing data models and query processing procedures that could theoretically be applied to astronomical data processing. While these approaches improve efficiency to some extent, they remain inadequate regarding high-efficiency data storage.

Reference [7] implements a high-performance NoSQL storage system, conducting in-depth research on data storage locations and query structures to ensure flexibility, portability, and stability. Reference [8] proposes distributed storage

and scaling solutions based on NoSQL to address the inability of traditional relational databases to meet massive data storage and access demands, suggesting the introduction of NoSQL as a mirror into database architecture systems to avoid resource waste and server overload to some degree. Experimental validation demonstrates that both approaches can be applied to astronomical data processing.

Reference [9] designs a MUSER data archiving and publishing system based on NoSQL technology, using the FastBit database for storage research and leveraging bitmap indexing advantages to substantially improve query efficiency. While related to our research, these studies focus primarily on data retrieval efficiency, whereas our work emphasizes data storage reliability, availability, and consistency.

2. Consistency Issues and Solutions

2.1 Causes of Consistency Problems

Metadata typically describes data attributes, such as image acquisition time and polarization mode for MUSER. Consequently, metadata becomes critical for supporting data retrieval, which constitutes an essential component of data management. The conventional storage approach separates metadata from data files, inherently creating consistency issues. If either metadata or data is lost, mismatches occur between them—a problem particularly prevalent given the enormous data volumes involved. Therefore, ensuring consistency between metadata and data is paramount. For subsequent analysis, we formally define the MUSER data consistency problem as follows:

Let D represent the data and M represent its metadata, where M comprises multiple field attributes. The consistency relationship between them is defined as consistent, while inconsistency is defined as inconsistent. Inconsistency can be further categorized into three types: (1) data exists but corresponding metadata cannot be found; (2) metadata exists but corresponding data cannot be found; and (3) both are missing. Processing methods for these scenarios are discussed in reference [10].

2.2 Traditional Storage Solutions

Two primary consistency solutions exist in traditional storage processes:

(1) **Simple consistency handling scheme**: This approach uses asynchronous non-blocking methods to store metadata and data files without consistency negotiation mechanisms, relying entirely on host reliability as the prerequisite for data consistency, as illustrated in [Figure 1: see original paper].

(2) **Two-phase commit protocol**: Reference [10] employs the mature two-phase commit (2PC) protocol for synchronization between data services,

assigning FITS file components to 2PC roles as shown in [Figure 2: see original paper]. The data acquisition server acts as the coordinator, while metadata and data serve as participants, with consistency confirmation achieved through the 2PC protocol among the three parties.

2.3 NoSQL-Based Consistency Solution

While synchronous storage of data and metadata is straightforward for small datasets with modest performance requirements, MUSER' s UVFITS files present unique challenges. Although many relational databases provide large object storage mechanisms—such as MySQL' s LongBlob, PostgreSQL' s Bytea, SQL Server' s Blob, and Oracle' s Blob/Clob—these approaches suffer from fundamental limitations imposed by ACID (Atomicity, Consistency, Isolation, Durability) properties. These limitations manifest as high write latency, poor horizontal scalability, and rigid data structures when confronting massive astronomical datasets.

To address these issues, we investigate a distributed massive data storage solution based on NoSQL. Through comprehensive research and evaluation, we select Cassandra [11] as the underlying storage platform. Compared with traditional relational databases, Cassandra offers rapid storage speed, high scalability, and flexible data structures. Furthermore, relative to other NoSQL databases, Cassandra provides several distinct advantages:

(1) **Decentralized architecture**: Single-point failures do not disrupt system operation.

(2) **Dynamic horizontal scaling**: New node addition does not affect ongoing processes.

(3) **Flexible storage schema**: Records can be dynamically augmented or modified during system runtime.

(4) **High-concurrency read/write capability**: The introduction of super-columns and column families reduces key matching operations and file data seek times, enabling high-speed data access.

Our new data model stores data files and metadata together, eliminating consistency issues while delivering substantial performance improvements over traditional approaches. compares data insertion performance between Cassandra and MySQL databases. Cassandra' s high-concurrency capabilities and decentralized architecture prove more advantageous for ensuring efficient and consistent data storage compared to MySQL' s shared-memory mechanism.

3. System Design and Implementation

3.1 MUSER Data Modeling

MUSER can operate in either cyclic or non-cyclic observation modes. Cyclic mode involves antenna observations cycling across radio frequency bands, while non-cyclic mode fixes observations within the same frequency range. MUSER supports two polarization modes (left-hand and right-hand circular) and four frequency bands (0.4–0.8 GHz, 0.8–1.2 GHz, 1.2–1.6 GHz, and 1.6–2.0 GHz). Each band contains 16 channels with 25 MHz bandwidth. During actual observations, one frame of data is generated every 3 ms, with each frame comprising a header and data totaling 0.1 MByte, yielding 19,200 frames per minute.

Our research focuses on storing observation time, filename, polarization mode, data type, frequency, and other information. Observation time serves as the primary key and is non-nullable. The filename field stores both the filename and storage path (non-nullable), while polarization and frequency represent frame header parameters. The result field contains image files converted from frame data, as shown in [Figure 3: see original paper] and [Figure 4: see original paper].

Cassandra can be viewed as a Key/Value data model constructed from a four-dimensional hash structure comprising Keyspace, Column Family, Key, and Column in a three-level nested configuration. [Figure 3: see original paper] and [Figure 4: see original paper] present two Cassandra database storage model designs: the first maps one Column Family (CF) to a single Column, while the second maps one CF to multiple Columns. The second design reduces CF-to-key matching operations and file seek times during read/write operations, thereby decreasing system overhead. Our data model adopts this second design pattern.

Cassandra' s data storage process involves committing action records to a log, writing data to an in-memory Memtable, and then batch-writing Memtable data to disk as SStable structures when system conditions are met.

3.2 MBDMS Implementation

[Figure 5: see original paper] illustrates the system architecture. The client accepts user requests, performs validation checks, and forwards requests to the server. Upon receiving requests, the server parses and processes them, returning data operation commands to the Data layer for execution. The server then determines operation success based on Data layer responses and returns information to the Client. The client currently requires three key functional modules: (1) a data processing module enabling users to query and delete database information with pagination; (2) an image conversion module allowing users to transform selected image data into viewable formats; and (3) an image retrieval module supporting feature-based similarity searches to identify the most similar images in the database.

We implement the massive astronomical data storage system using Python

and the Django framework, integrating interfaces for both relational and non-relational databases. [Figure 6: see original paper] displays the Web-based MUSER data retrieval interface. Given MUSER data' s massive, unstructured, and weakly consistent characteristics, client users fall into two categories: (1) metadata managers who focus on metadata uniformity and regularity to facilitate efficient storage, and (2) data users who retrieve frame data through stored metadata and require stronger consistency between metadata and frame data.

4. Performance Testing

To validate MBDMS performance, we conduct three test groups using its relational and non-relational database interfaces to connect MySQL and Cassandra databases. The experimental environment consists of four machines with E7200 Core 2 Duo 2.53GHz CPUs, 2GB RAM, and 7200 RPM SATA hard drives, running CentOS 64 (kernel version 2-504.el6.x86_64). Database versions are MySQL Cluster GPL 7.2.28 and Apache Cassandra 3.9, both using default configurations.

4.1 Query Statement Comparison

presents commonly used query statements for both databases. Retrieval experiments use time (primary key) as the query condition. By adopting unified storage of metadata and frame data, MUSER data consistency is ensured. Performance tests compare the new Cassandra-based storage strategy against the legacy MySQL approach, providing representative results. Query targets include file storage paths, data types, frequencies, polarization, and frame data. Both databases use identical query statements to ensure experimental accuracy.

4.2 Retrieval Performance and Scalability Testing

[Figure 7: see original paper] compares retrieval performance across different data volumes and node counts. [FIGURE:7(a)] shows query results with seven dimensions and three cluster nodes across varying database sizes. Scalability represents a fundamental requirement for MUSER big data storage—as data volume increases, performance and storage capacity must scale accordingly. Since data is typically served online, storage systems must support dynamic node addition and removal. Results demonstrate that Cassandra-based operations require less time than MySQL because Cassandra avoids consistency checking during data insertion, whereas MySQL' s shared-memory mechanism incurs significantly greater overhead for consistency guarantees.

[FIGURE:7(b)] compares query performance across different cluster node counts with five million records and seven dimensions. Results indicate that more nodes yield better cluster performance. Experiments confirm that MBDMS performance is superior when using the Cassandra non-relational database interface.

4.3 Query Dimension Comparison

[Figure 8: see original paper] compares single-dimension and multi-dimension (seven dimensions) query performance between the two databases. Results show that increased query dimensions raise system overhead for both systems, but MySQL' s time consumption grows at a substantially higher rate than Cassandra's. This difference arises because MySQL's shared-memory storage approach differs from Cassandra' s index tree node distribution, creating greater index pressure when multiple processes query simultaneously and causing system overhead to increase dramatically.

4.4 Index Space Comparison

[Figure 9: see original paper] analyzes storage space and index space usage for both databases storing identical datasets. Minimizing disk overhead is essential for efficient massive data storage systems. Tests reveal that non-relational databases occupy significantly less space than relational databases for MBDMS, demonstrating clear advantages for massive data storage.

These experiments show that for smaller datasets (below five million records), MySQL performance generally meets storage and retrieval requirements with query times around 7.5 seconds. However, for larger datasets (exceeding ten million records), MySQL query times surpass 17 seconds, rendering it unsuitable for subsequent data processing. Across single-dimension, multi-dimension, and insertion experiments, MBDMS demonstrates clear performance advantages when interfacing with Cassandra.

5. Conclusion

This paper implements high-speed synchronous storage of MUSER frame header and frame data using Cassandra, resolving consistency issues arising from separate storage in relational database environments. Cassandra' s scalability provides adaptability for massive astronomical datasets. MBDMS integrates effectively with Cassandra to meet common astronomical data management requirements. Future work will further optimize MBDMS storage and retrieval performance and provide interfaces for other mainstream NoSQL databases.

References

[1] Mei Ying, Liu Donghao, Wang Feng, et al. A study of the FITS-IDI Format for the Chinese Spectral Radio Heliograph[J]. Astronomical Research & Technology—Publications of National Astronomical Observatories of China, 2014, 11(4): 388-395.

[2] Mei Ying. Research on Key Technologies of MUSER Massive Data Preprocessing[D]. Kunming: Kunming University of Science and Technology, 2015.

[3] Shi Congming, Wang Feng, Deng Hui, et al. High performance negative database for massive data management system of the Mingantu Spectral Radioheliograph[J]. Publications of the Astronomical Society of the Pacific, 2017, 129(978): 1-12.

[4] Liu Yingbo, Wang Feng, Ji Kaifan, et al. Research on metadata storage and querying of FITS file based on NoSQL[J]. Application Research of Computers, 2015, 32(2): 461-465.

[5] Chen Yajie, Wang Feng, Deng Hui, et al. The application of ElasticSearch in the massive astronomical data retrieval[J]. Acta Astronomica Sinica, 2016, 57(2): 241-251.

[6] Tang Liyang, Ni Zhiwei, Li Ying. Scalable Distributed Inverted Index Built on Cassandra[J]. Computer Science, 2011, 38(6): 187-190.

[7] Hou Pengpeng. Design and Implementation of a High-Performance NoSQL Storage System[D]. Beijing: University of Chinese Academy of Sciences, 2013.

[8] Pan Hongzhi. Research and Implementation of High-Performance NoSQL Storage System[D]. Jilin: Jilin University, 2014.

[9] Lin Maoqiang. Design and Implementation of MUSER Data Archiving and Publishing System Based on NoSQL Technology[D]. Kunming: Kunming University of Science and Technology, 2015.

[10] Liang Bo, Chen Tengda, Yu Konglin, et al. Research on the consistency of FITS data in the process of distributed real-time storage[J]. Astronomical Research & Technology, 2016, 13(4): 489-497.

[11] Hu Chaoye. Analysis and Application Research of High-Concurrency Read-Write System Based on Cassandra Database Cluster[D]. Shanghai: Shanghai Jiao Tong University, 2013.

Note: Figure translations are in progress. See original paper for figures.