PAPER • **OPEN ACCESS**

A distributed storage method for real-time database System design

To cite this article: XianHui Li and Chun Zhou 2018 *IOP Conf. Ser.: Mater. Sci. Eng.* **452** 032010

View the article online for updates and enhancements.

IOP ebooks™

Bringing you innovative digital publishing with leading voices to create your essential collection of books in STEM research.

Start exploring the collection - download the first chapter of every title for free.

This content was downloaded from IP address 191.96.250.166 on 13/12/2018 at 17:12

A distributed storage method for real-time database System design

XianHui Li[1, 2, *], Chun Zhou[1, 2, a]

[1]Nari Group Corporation/State Grid Electric Power Research Institute, Nanjing 211106, China.
[2]China Realtime Database Co, Ltd., Nanjing 210012, China.

*Corresponding author: lixianhui@sgepri.sgcc.com.cn, [a]364073988@.qq.com

Abstract. With the rapid development of network technology, the amount of data generated by real-time system is increasing exponentially. Business applications are putting forward more and more high demand on the management and real-time requirement. As the traditional stand-alone RTDB technology has met the needs with difficulty, the distributed thinking is introduced into the RTDB field, mainly studied from the Data distribution mode, Data redundancy backup, Data consistency and other aspects, Propose a design of data distributed storage in real-time data system. This design can provide technical support for current RTDB scalability and reliability.

1. Introduction

Database technology was born in the late 20th century. Its theory and technology development is extremely rapid, and its application is becoming more and more widespread. An important branch of database technology is RTDB. RTDB is a database built by real-time data models. RTDB technology is a product of the combination of real-time system and database technology [1].

The specialization and productization of real-time databases in data communication, data storage, data retrieval, data access, data processing, and data display provide convenient and stable data support for the construction of analytical applications based on large-capacity real-time historical data. It enables the application system to make full use of valuable production real-time historical data from a higher and deeper level. With the rapid development of network technology, the amount of data generated by real-time systems has grown exponentially, and business applications have imposed increasingly higher requirements on the management of real-time data and the real-time nature of applications. The existing stand-alone RTDB technology has great limitations both in theory and in practical applications, and can no longer fully meet current needs.

At present, the use of RTDB in the industry still based on single-core centralized deployment [2]. The introduction of distributed computing and storage technology ideas [3] can well solve these bottlenecks encountered by stand-alone real-time databases. However, how to design a distributed RTDB to meet the transaction application processing timeliness and transaction throughput requirements is a difficult point. This paper designs a distributed RTDB management system by studying some key technologies of distributed RTDB (data distribution, data consistency, data redistribution), which can provide technical support for the development of RTDB applications.

Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

2. Overview

2.1. Overall Architecture Design

Distributed RTDB deployment architecture shown in Figure 1:

In the deployment architecture, the management nodes is the administrator of the entire distributed RTDB and mainly stores system metadata information, including key information such as the data distribution mode, the status of each node, and the consistency status of the active and standby nodes.

Scheduling nodes belong to the distributed access layer. Unified interface enables applications to access the distributed RTDB as a complete logical entity. In addition, the scheduling node also belongs to the distributed location layer, which is the distributer and collector of data. It is mainly responsible for data distribution, collection of query results, and task scheduling. When a distributed node queries and accesses data on multiple nodes, concurrent access processing enables parallel processing of multiple data requests on multiple storage nodes, thereby enabling efficient distributed data access.

Data nodes belong to the distributed storage layer. Each data node runs and manages a database instance. The data node is responsible for the actual storage of the entire database system data, receives data from the scheduling node, executes the decomposed query task, and the execution result is returned to the application program through the scheduling node. The number of data nodes is limited only by the hard conditions such as Ethernet bandwidth and the physical conditions of the equipment room. Each data node only stores data belonging to the corresponding partition and is logically equivalent. The active and standby data nodes implement data redundancy between nodes.

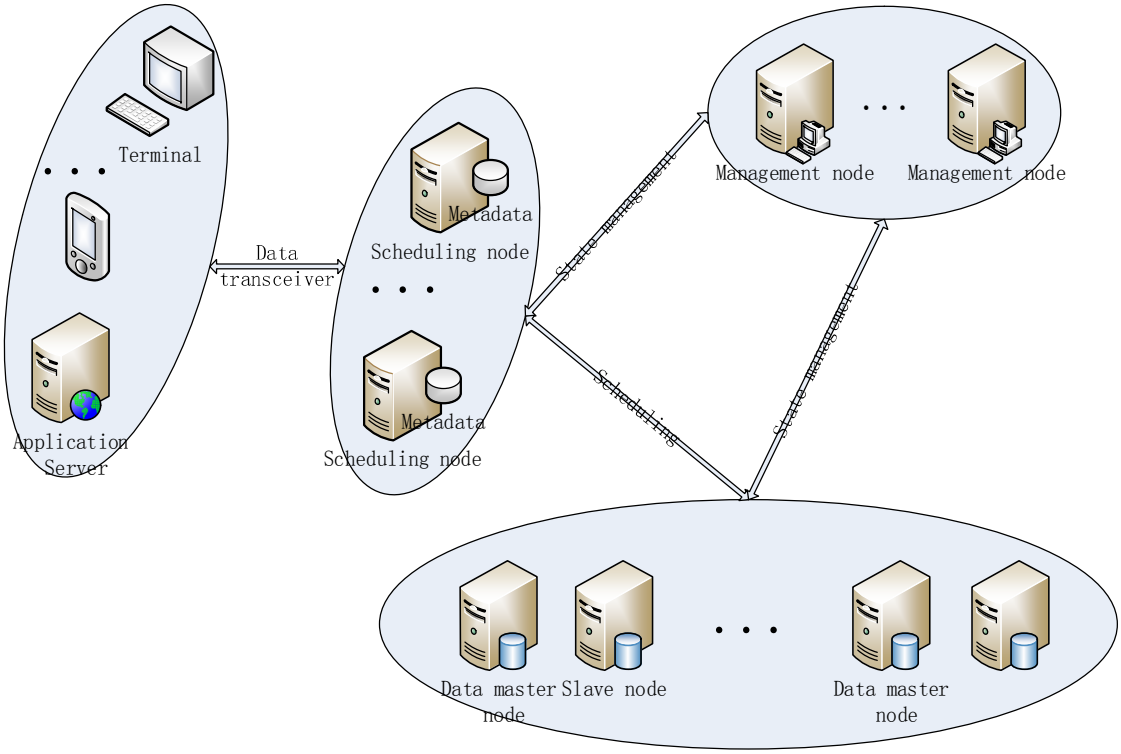

Figure 1. Distributed RTDB system deployment architecture.

2.2. Metadata table

The RTDB can be simply understood as consisting of two parts: tag point information database and value database [4], as shown in picture 2:

Tag point information

Tag point name1	Tag point Description1	Tag point ID1	Compression algorithm 1	…
Tag point name2	Tag point Description2	Tag point ID2	Compression algorithm 2	…
…				
Tag point nameN	Tag point DescriptionN	Tag point IDN	Compression algorithm X	…

Value database

Tag point name1	Timest amps	Value	Status	Timest amps	Value	Status	…
Tag point name2	Timest amps	Value	Status	Timest amps	Value	Status	…
…							
Tag point nameN	Timest amps	Value	Status	Timest amps	Value	Status	…

Figure 2. Label point table and data value table.

The tag point information base contains a table of basic information of the survey points, with the tag point tag (point_name) as a key, and a record containing basic configuration information of a tag point such as a tag point description, a compression algorithm. The user can query the basic information of the tag point from this tag point information base. The value database contains real-time value cache and historical data storage. Each record reflects the time stamp, value, and quality of real-time data generated by a tag point. Users can query real-time data values from the value database. Therefore, the two main dimensions of the RTDB are the tag points and the data time. If you need to distribute all the data of the RTDB to multiple nodes, you should start from these two dimensions.

Metadata storage: The tag-point table is stored as a metadata table on the scheduling node, and each scheduling node has a complete tag-point information. Multiple scheduling nodes are mutually backed up. The management node controls the status of the scheduling nodes and uses the synchronization thread to perform abnormal recovery.

2.3. Data flow

Data access data flow is shown in Figure 3. The data is sent from the application server or client to the scheduling node. The scheduling node sends the data to different master data nodes according to the distribution rules, And master data node forwards the data to backup node during the storage process.

The data query data flow is as shown in Figure 4. The query request and query conditions are sent from the application server or client to the scheduling node. The scheduling node uses the tag points and time ranges involved in the query conditions. Through the distribution rules, filter the relevant data nodes, split and re-organize multiple sub-queries to allocate to multiple data nodes, multiple data nodes process queries in parallel, complete the results and return the results to the scheduling node. The scheduling node waits for all allocated children. After the query returns the result, it will do the aggregation process and feedback the result to the application server or client.

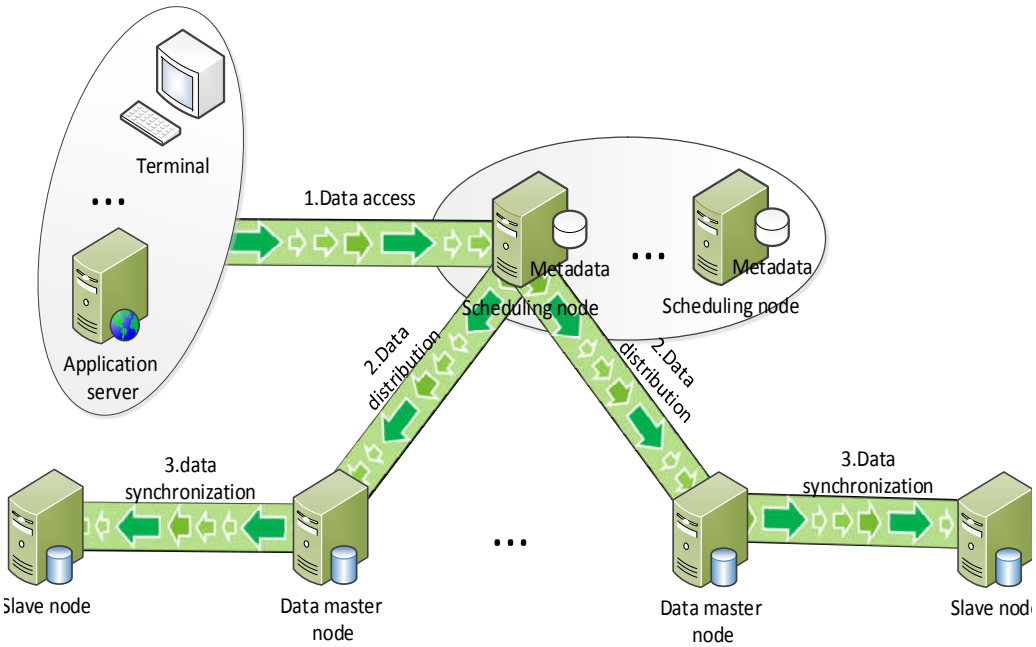

Figure 3. Data access data flow.

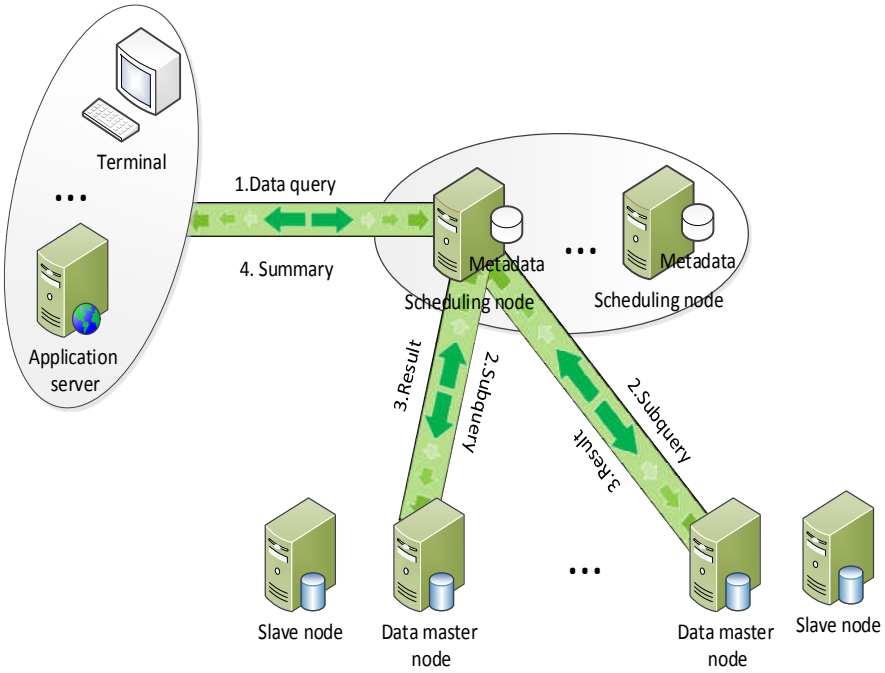

Figure 4. Data query data flow.

This involves the allocation of primary and secondary nodes. Each sub-query can only be sent to a data master or backup node, unless the query exception will continue to be sent from the scheduling node to another node to do the query (at the same time to report the node status to the management node).The scheduling node can select the distribution node by determining the busyness of the active and standby nodes, and the data node periodically reports the current active state and the busyness level to the management node. The scheduling node also periodically synchronizes all data node

statuses from the management node. The busyness metric [5] can be conveniently measured from the average CPU usage, average network usage, current disk usage, and current memory usage.

3. Data Distribution Design

3.1. Distribution rules

When designing distribution rules, it is necessary to take into account the relevance of the tag point data (if it is possible to place part of the query operations directly within the data node, it is obviously more important than to concentrate the calculation data after transferring the data to the scheduling node). , but also to consider the efficiency of parallel processing of distributed architecture.

We can distribute a batch of label points and data of a time period in the same data node. It can be considered by a custom HASH function. The hash function is similar to the following format:

slice_ id= (w1× (hash_str (point_name) /b1)+w2× (day_ time (time)/b2))% Hash_ Bucket

Where slice_ id is the last function to determine the allocated fragment number, it obtains a set of Hash_ Bucket from the function calculation result.Hash_str (point_name) and day_time (time) are a quantized function of the label point name and time period, respectively. Taking into account the relevance of data within the label point or time area, design b1 and b2 control the degree of data dispersion within the label point or within an adjacent time period.W1 and w2 are coefficients, or they are called unnormalized weights. For example, if w2 is set to 0 in extreme cases, the data is completely distributed according to the label points. The data at all times of each label point is stored in the same data fragment or its backup fragment. Similarly, if w1 is set to 0, it means that the data is distributed according to the time period. The data of all the label points in each time period is stored in the same data fragment or its backup fragment.

Considering the actual operation of the system, the nodes may have adjustment changes. If the Hash_ Bucket is set as the initial number of nodes and each slice_ id corresponds to a node, the node changes will cause the Hash_ Bucket to change, and then the whole data will be redistributed and all the data of the system will be involved. The file, because it needs to use the new hash function to calculate the corresponding data node for each data value, this method can only improve the redistribution efficiency by increasing the parallelism of the hash operation and data transmission.

Learning from the consistent HASH algorithm [6] can effectively solve this problem. The management node stores an array (fragment mapping table) of size Hash_ Bucket. Each element stores the data node where the index of the corresponding element is located. The mapping relationship can be manually assigned or automatically assigned by mode of node_ num (number of data nodes).

Figure 5 shows how the data is distributed to different nodes according to the distribution rules. The distribution rules include custom HASH functions and fragmentation mapping tables. Label points and data can be distributed to the HASH function and fragmentation mapping table. Different data nodes, distribution rules need to be determined before the label point is written, and once the distribution rules are determined, they will not change until the data node changes need to be redistributed, and only the fragmentation mapping table can be changed.

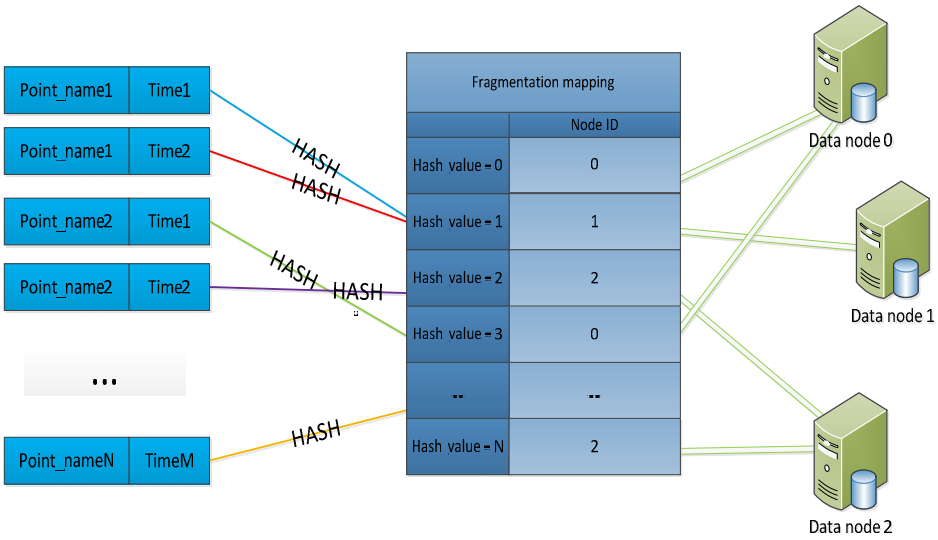

Figure 5. Data distribution rules.

If the distribution rule is not set in advance before the entire distributed RTDB system starts, the fragmentation mapping table is automatically generated according to the number of data nodes after startup. Before accessing tag points and data, HASH function related parameters still need to be set through the management client. The distribution rules are directly transferred to the management node for storage. The scheduling node needs to obtain the distribution rule from the management node before distributing the label point and data. After the scheduling node acquires the distribution rule for the first time, it will be stored in memory. It is not necessary to reacquire the allocation rule every time the data is allocated. If the data node changes during system expansion or shrinking, and the distribution rule needs to be modified, the scheduling node needs to be restarted. Initialize and get the distribution rules.

3.2. Data Redistribution Design

As the total amount of data changes, it may be necessary to consider the expansion or reduction of data nodes. The change of data nodes must consider the data equalization and data redistribution.

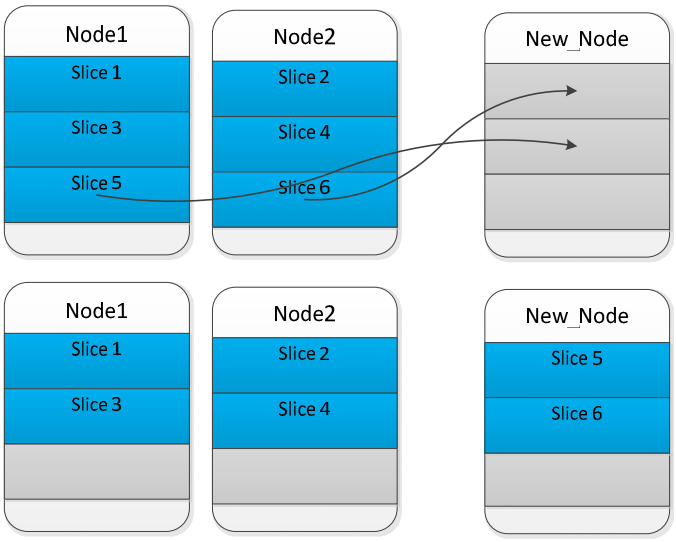

Figure 6. Data fragment movement during data node expansion.

The previously designed real-time library system uses the fragmentation mapping table and the way the hash distribution is distributed in two layers. The redistribution strategy can be designed so that the hash function does not change and only changes the fragmentation mapping table. For example, a new mapping table is automatically generated according to the new number of nodes, and the scheduling node compares the difference between the old and new fragmentation mapping tables to set the corresponding fragment movement path of each data node and allocates it to each data node. Fragmented directory for data migration. In this way, the sharding is the minimum moving unit, no data decompression and hash operations are needed, and the CPU operation, disk IO, and network IO overhead are greatly reduced.

A more efficient approach is to divide the data evenly across the entire database system. Figure 6 is a process of capacity expansion. Part of the data nodes whose total amount of data is greater than the average amount of data is fragmented to less than the average amount of data. Move so that the total amount of data that needs to be moved in the data redistribution is minimized.

4. Stand-alone RTDB storage design

The RTDB storage design follows the existing single-machine RTDB design as much as possible. That is, designing a single database instance which can be performed as an independent entity or as a data node in a distributed architecture. So a single data node needs to include All modules of RTDB: network transceiver module, cache module, data processing module, storage module (including index). The data storage module is mainly considered here. The data cache module and the data processing module are all built on this module. The network transceiver module is only related to the designed communication protocol and is relatively independent of the data logic.

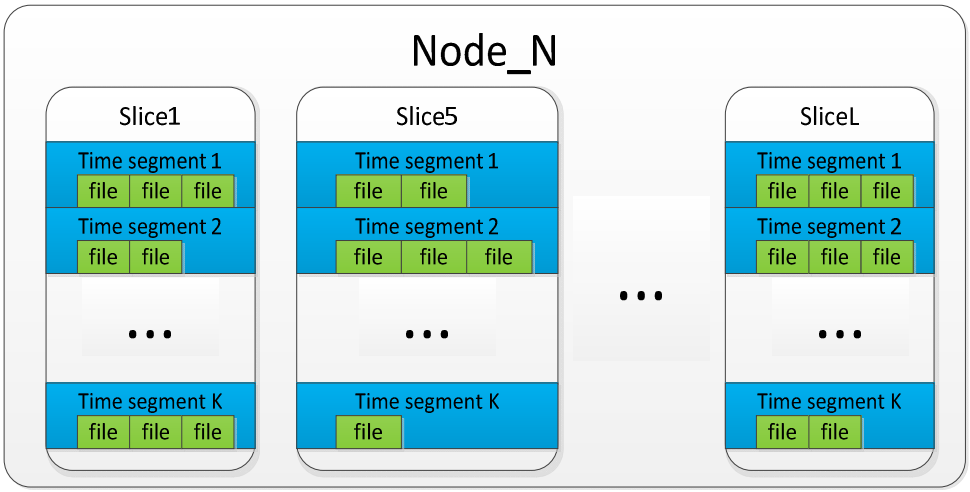

Figure 7. Single data node basic storage structure.

Data storage is managed according to fragmentation. As shown in FIG. 7, each slice_id corresponds to a file directory. If running as a separate database instance, the instance contains all fragmentation numbers. If it is a node in a distributed architecture, the instance contains a partial fragmentation in the fragmentation mapping table. Each fragment file directory stores data blocks according to the time difference data file group, and the file group and the time period correspond one by one. Each file group and fragment directory stores a version number. Each time you write and modify the updated version number, the query does not update the version number [7].

From the point of view of the entire distributed system, the data node is the most basic storage and calculation unit in the system. From the perspective of data integrity, the internal data file group of the data node is the smallest granular self-descriptive storage unit, and is also the most Basic self-recovery and synchronization unit.

5. Data redundancy backup and consistency processing

One of the difficulties in distributed databases is how to deal with the consistency of stored data, including metadata and data synchronization and exception handling.

5.1. Management node data consistency

The management node generally needs to configure multiple nodes to ensure high availability. It mainly stores key configuration such as distribution rules and small granular information such as the status of each active and standby node. It is suitable for implementation based on similar architectures such as corosync or zookeeper, ensuring the information is in an orderly and consistent manner. For this type of scenario, there are currently a wide range of application practices, for example [8].

5.2. Metadata consistency of scheduling nodes

Besides synchronize the node states to the memory from the management node. the scheduling node mainly stores the metadata information of tag points. The addition, deletion, and modification of label points are executed in a transactional manner, and the atomicity and strong consistency of a single operation need to be guaranteed. That is, each scheduling node operates needs to be ensure successful synchronization to all other normal scheduling nodes. Otherwise, we need to either set abnormal status to the node that fails to synchronize so that we can do a synchronization recover later, or perform the undo operation on the successfully executed node. In addition, subsequent operations must wait for all synchronous operations of the pre-operation to complete.

The metadata information of scheduling node is divided into blocks, and all operations have strong consistency, so the normal node should be exactly the same for each block and block sequence. The management node regularly detect abnormal scheduling nodes and do synchronous recover operations.

5.3. Data redundancy backup and consistency

Mirror mechanism is used to ensure the high availability of the cluster. The design needs to satisfy real-time and fault-tolerance. For the consistency of the master and mirror data nodes, the strongest consistency is the simplest approach. However, CAP theory [9] tells us that consensus, availability, and partitioning cannot be satisfy both, and weak consistency has its scope of application, especially in scenarios where high real-time requirements and low requirements for timely consistency are required.

The status of the master and mirror data nodes is monitored by management node. The scheduling node obtains the status of the active and standby data nodes through the management node, which can assign write nodes and query nodes. At the same time, the status of the data nodes during real-time task processing is also fed back to the management node in time.We have mentioned The data flow of writing and query earlier. when writing, Data is directly written to master node by the scheduling node. The automatic transfer of data from master node to the mirror node does not guarantee the strong consistency, which only ensures Final consistency. When querying, the mirror node and the master node shares the query pressure. The scheduling node dispatches the query task according to the load.

5.4. Data exception handling

If an exception occurs in the main library node, the standby node is assigned to be upgraded to the master node by the management node. After the abnormal node recovers, it will continue to operate as a backup node. The management node starts the consistency detection when an abnormal data node is detected. The two-layer version number detection and file level synchronization can minimize the impact of the recovery operation on real-time processing performance, and the recovery efficiency is high. The specific detection and recovery mode is: By analyzing the inconsistencies between the version numbers of the primary and secondary data nodes and the version number of the file group within the fragment, determine the indexes and data file blocks that need to be synchronized and synchronize the data. That is, the replacement operation is directly performed on the file groups of different version numbers, and the data flow is transmitted through the underlying TCP.

In order not to affect the real-time performance of the entire system, the online synchronization mode is used for the exception recovery. That is, the data does not affect the normal reading and writing during the synchronization process. The following strategy is adopted: The modification records generated by the master node in the data synchronization process are recorded in the master data node in Log mode. After the synchronization is completed, the Log and the update data are parsed. The data modification in the parsing process is still added to the end of the Log until all Log parsing is completed. The management node sets the status of the active and standby data nodes to Normal.

6. Conclusion

Since the performance, reliability, scalability and other requirements of RTDB are getting higher and higher in the industrial field, this paper proposes a data distributed storage design in real-time data systems by combining distributed technologies and RTDB technologies. At the same time, a set of solutions for data redundancy, data redistribution and data consistency are given. Provide technical support for breaking the bottleneck of current RTDB technology.

Acknowledgments

This paper is partially funded by science and technology projects (massively distributed parallel processing database system development and application)

References

[1] LIU Yunsheng. Real-time database system [M], Beijing:China Science Publishing & Media Ltd, 2012: 16.
[2] CUI Le,SHI Junchang,ZHANG Xiaohua,WANG Rongxi. The application of RTDB in industry [J] Industrial Instrumentation & Automation, 2015 , 4 :73-75.
[3] Jun YANG,Guo-hua ZHAO,Ke-jia WANG,Jun-de SONG. A modern service-oriented distributed storage solution[J]The journal of China Universities of Posts and Telecommunications,2009 , 16 (1) :120-126.
[4] JIAN Siting Real-time database and relational database design features [J] Programmable controller and factory automation, 2012,2: 28.
[5] LI Chunhui,XIE Yongbin. Optimization Simulation of Resource Load Balancing Scheduling in Cloud Computing Environment [J] Computer Simulation, 2017,12: 420-425.
[6] D. Darger, E. Lehman, T. Leighton, M. Levine, D. Lewin and R. Panigrahy. Consistent Hashing and Random Trees:Distributed Caching Protocols for Relieving Hot Spots On the World Wide Web. ACM Symposium on Theory of Computing, 1997. 1997: 654-663.
[7] ZHANG Weiyan ,WU Zhijie,XIA Tao. EJB Design Pattern of Version Number to Protect Database Consistency[J] Journal of Terahertz Science and Electronic Information Technology, 2004,2: 136-139,146.
[8] NI Chao. From Paxos to Zookeeper: Principles and Practices of Distributed Consistency [M], Beijing:Publishing House of Electronics Industry,2015: 59.
[9] E Brewer.cap twelve years later: how the rules have changed [J] Computer, 2012 , 45 (2) :23-29.