MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

V.N. Karazin Kharkiv National University

School of Mathematics and Computer Science

Department of Theoretical and Applied Informatics

Master's Thesis

Survey of Data Replication Protocols in Distributed Systems

Author:

Final year Master's Program student,

specialty - Computer Sciences and Information Technologies,

educational program: "Informatics"

Liu Mingxuan

Supervisor: Kyrylo Rukkas

Reviewer: Kyryl Korobchynskyi

Adviser: Oleksandr Barskyi

Kharkiv, 2024

Аннотація

На тлі поступового дозрівання централізованих систем баз даних розподілені системи з'явилися і продовжують розвиватися. У свою чергу, технології реплікації та синхронізації даних стали пріоритетами в галузі досліджень цих систем. Сюжетні системи реплікації баз даних мають дві архітектурні форми: архітектуру на рівні системи бази даних та архітектуру проміжного програмного забезпечення, незалежного від окремих систем баз даних. Перша має відносно вищу ефективність реплікації, в той час як друга має відносно більшу універсальність. Більшість зарубіжних виробників баз даних реалізовують системи реплікації баз даних у своїх продуктах. У нашій країні, зазвичай використовується архітектура проміжного програмного забезпечення, яка має відносно нижчу ефективність. Тому необхідно проводити дослідження функції реплікації на рівні системи бази даних. Основні завдання цієї роботи включають наступне: описати контекст дослідження та стан досліджень на внутрішньому та міжнародному ринках систем реплікації баз даних, проаналізувати основні технології, пов'язані з цими системами, після чого встановити передумови для реплікації транзакцій та асинхронної реплікації; виконати аналіз потреб та визначити загальну архітектуру системи, провести порівняння та аналіз технологій, пов'язаних з двома основними компонентами сервера реплікації, та розробити відповідні методи отримання та передачі даних; конкретно розробити та реалізувати ці два основні компоненти, наприклад, вдосконалити контрольні

таблиці, використовувати модель передачі з проміжними агентами, розв'язувати конфлікти даних за допомогою маркування часу, реалізувати управління маршрутизацією та розробити процедури аварійного оброблення; нарешті, зважаючи на оцінку сильної консистентності, кінцеву консистентність, доступність системи, споживання пропускної здатності мережі, споживання ресурсів зберігання, споживання обчислювальних ресурсів, продуктивність операцій читання та запису, запропонувати методи та ідеї оцінки продуктивності протоколів реплікації даних для розподілених систем, що забезпечують цінну інформації для досліджень та практики, пов'язаних з реплікацією даних в розподілених системах. Ключові слова: Розподілена, Реплікація баз даних, Продуктивність.

summary

As centralized database systems mature, distributed systems emerge and continue to develop, and data

replication and synchronization technology becomes a key research area. Database replication systems have two forms: database system-level architecture and independent middleware architecture. The former has relatively high replication efficiency, while the latter has relatively stronger versatility. Most foreign database vendors have implemented database replication systems in their products, while domestic vendors tend to prefer middleware architectures with slightly lower efficiency. Therefore, it is necessary to study the database system-level replication function. The main work of this paper covers: introducing the research background and the current status of database replication systems at home and abroad, and establishing transactional replication and asynchronous replication as the premise after analyzing the main related technologies; carrying out demand analysis and determining the overall architecture, comparing and analyzing the technologies involved in the two core components of the replication server, and designing corresponding data acquisition and transmission modes; carrying out specific design and implementation of the two core components, such as improving the control table, adopting the intermediate agent transmission model, relying on timestamps to resolve data conflicts, implementing routing management, and designing emergency handling procedures; finally, giving

methods and ideas for the performance evaluation of distributed system data replication protocols from many aspects such as strong consistency evaluation, eventual consistency evaluation, system availability, network bandwidth consumption, storage resource consumption, computing resource consumption, read operation performance, and write operation performance, which provides a valuable reference for the related research and practice of distributed system data replication.

Keywords: distributed, database replication, performance

Chapter 1introduction

1.1 Research background and significance

1.1.1 Research Background

With the rapid advancement of information technology, the amount of data has shown an explosive growth trend, and the requirements for data processing and storage are also increasing. Distributed systems have emerged, which can achieve efficient processing and storage of large-scale data by relying on the computing resources and storage capabilities of multiple nodes, meeting the diverse needs of modern enterprises and Internet applications.

In a distributed system, data is usually stored in multiple nodes to enhance the availability, reliability, and performance of the system. However, this also raises the problem of data consistency. Due to factors such as network latency, failures, and concurrent operations between nodes, data may be inconsistent on different nodes. For example, in an e-commerce system, multiple users may operate on the inventory data of the same product at the same time. If there is no effective data replication protocol to ensure data consistency, it may lead to serious problems such as overselling or inventory data errors.

Traditional centralized database systems are relatively easy to maintain data consistency because data is stored on a single node or a few tightly coupled nodes. However, the architectural complexity of distributed systems makes data replication protocols a key technology. Different application scenarios have different requirements for data replication. For example, financial trading systems may focus more on strong consistency and data integrity, while social media applications may allow eventual consistency to a certain extent in exchange for higher system response speed and availability.

1.1.2 Research significance

Ensure data consistency and integrity: Data replication protocols ensure data consistency on multiple nodes through specific mechanisms, such as master-slave replication, multi-master replication, or consensus-based replication. In applications such as distributed databases and file systems, this is crucial to maintaining data accuracy and reliability, and can avoid business errors and data loss risks caused by data inconsistency.

Improve system availability and reliability: When some nodes fail, the data replication protocol can ensure that the data on other normal nodes is still available, so that

the system can continue to provide services. For example, in a distributed storage service in a cloud computing environment, even if a storage node fails, through data replication, the user's data can still be obtained from other replica nodes, which significantly improves the overall reliability and fault tolerance of the system.

Optimize system performance and scalability: A reasonable replication protocol can dynamically adjust the data replication strategy according to the system load and node performance to achieve efficient data distribution and access. For example, by replicating hot data to nodes close to users, data access latency can be reduced and the system response speed can be improved. At the same time, as the scale of the distributed system expands, the data replication protocol can adapt to the addition of new nodes and the exit of old nodes, ensuring that system performance is not affected and achieving good scalability.

Promote the widespread development of distributed system applications: In the current digital age, many fields such as big data analysis, the Internet of Things, and blockchain rely on distributed system technology. Reliable data replication protocols have laid a solid foundation for applications in these fields, enabling them

to operate stably in complex distributed environments, promoting the innovation of related technologies and the expansion of application scenarios, and bringing new opportunities and changes to social and economic development.

1.2 Current research status at home and abroad

1.2.1 Current status of research in Ukraine

In terms of theoretical research: Ukrainian universities and research institutions have paid a certain degree of attention to the theoretical research of distributed system data replication protocols, and have carried out relevant research work in data consistency theory and distributed algorithms, providing theoretical support for the design and analysis of data replication protocols. For example, some universities in Ukraine will involve the basic theoretical teaching of data replication protocols in courses such as distributed computing and database systems to cultivate students' understanding and mastery of relevant knowledge.

Application research: In the field of practical application, some Ukrainian enterprises and scientific research projects have begun to pay attention to the application of distributed system data replication

protocols. For example, enterprises in the fields of telecommunications and finance have begun to study and apply data replication technology to improve the reliability and availability of the system in order to cope with possible system failures and data loss problems.

International cooperation and exchange: Ukrainian researchers actively participate in international cooperation and exchange, and cooperate with foreign research institutions and enterprises to carry out research projects on distributed system data replication protocols. Through cooperation, Ukrainian researchers can obtain the latest international research trends in a timely manner, introduce advanced technologies and methods, and improve their own research level.

1.2.2 Current status of foreign research

Basic theory and model research: Foreign countries have conducted in-depth research on the basic theory of distributed system data replication protocols, and have begun to explore basic theories such as data consistency models and distributed transaction processing many years ago. For example, the sequential consistency model proposed by Lamport provides an important theoretical foundation for subsequent research. By defining the

sequential rules of data operations, it ensures that the consistency of data in a distributed system conforms to a specific logical order, providing a theoretical basis for the design and analysis of data replication protocols.

Development and optimization of mainstream protocols:

Paxos and its variants: As a classic distributed consistency protocol, the Paxos protocol has always been one of the hot topics of research. Foreign researchers have continuously optimized and improved it, such as the Raft protocol, which simplifies the implementation complexity of Paxos while maintaining high availability and strong consistency. Raft uses clear mechanisms such as leader election and log replication to enable distributed systems to reach consensus more efficiently during data replication. It is easy to understand and implement and is widely used in distributed systems such as Apache Kafka.

Two-phase commit (2PC) and three-phase commit (3PC): 2PC and 3PC protocols are also widely used in distributed transaction processing and data replication. Researchers have conducted in-depth research on their performance, reliability, and applicable scenarios, and proposed various improvement plans to solve problems such as

blocking and data inconsistency that may occur in a distributed environment.

New technology integration: With the rise of emerging technologies such as cloud computing, big data, and the Internet of Things, foreign research has integrated distributed system data replication protocols with these technologies. For example, in a cloud computing environment, research is conducted on how to dynamically adjust data replication strategies based on the characteristics of the cloud platform and user needs to improve resource utilization and reduce costs; in an IoT scenario, considering the heterogeneity of devices and the instability of the network, a data replication protocol adapted to the IoT environment is designed to ensure reliable transmission and consistency maintenance of data among a large number of distributed devices.

Performance evaluation and optimization: Focus on the performance evaluation and optimization of data replication protocols. By building various performance models, analyze the protocol's throughput, latency, resource consumption and other indicators in different scenarios, and propose corresponding optimization strategies. For example, use simulation experiments,

actual system tests and other methods to evaluate the performance of different replication protocols in large-scale distributed systems, providing a reference for practical applications.

1.3. Main research contents

There are two architectural designs for mainstream database replication systems, one is the replication architecture within the database system, and the other is the replication architecture based on middleware. Compared with the replication architecture based on middleware, the replication architecture based on the database system has more advantages in terms of replication efficiency and can fully exert the performance of the database system. At present, domestic database system technology lags behind foreign countries, and China is more inclined to conduct research on replication architecture based on middleware. This paper chooses to adopt the replication architecture of the database system to achieve an efficient replication system in the SYBASE database system. Therefore, this paper first analyzes the various technologies required for the database replication system in an all-round way, selects some efficient technologies and improves them, and plans the application architecture of the replication system and

its implementation strategy. The main research contents of this paper are as follows:

1. Design an optimized control table for data capture to improve data capture efficiency and reduce resource usage during the capture process.

2. Design efficient data transmission components for data transmission: mainly configure a dedicated transmission process for each transmission path to improve data transmission efficiency.

3. To ensure the consistency of data copies, design and implement conflict data handling strategies suitable for database replication systems.

4. Design a flexible routing management module and reasonably configure the connection structure of the master-slave database in the distributed system through routing management.

5. Design an emergency handling procedure. When the master database fails, properly arrange for a slave database to become the master database and update the data of the new master database.

Chapter 2 Analysis of Related Technologies

2.1 Distributed Transactions

Distributed databases can support distributed transactions. Transactions consist of one or more SQL statements. Database transactions usually consist of a series of read and write operations. When the database management system (DBMS) receives a submitted transaction, it must ensure that all operations in the transaction are successfully executed and their results are permanently stored in the database. If only some operations are successfully completed, all operations in the transaction must be rolled back and the data must be restored to the unoperated state. This transaction must not affect the execution of other transactions in the database. Transactions must meet the four characteristics of ACID, namely atomicity, consistency, isolation, and durability.

Atomicity: Atomicity means that all operations contained in a transaction are either completely successful or completely failed and rolled back. Therefore, if the operation of a transaction is successful, it must be completely applied to the database; if the operation fails and cannot have any adverse effects on the database, it must be rolled back to restore to the state before the operation.

Consistency: Consistency means that the database is in a consistent state before the transaction operation, and the database remains in a consistent state after the transaction operation.

Isolation: Transaction isolation is aimed at concurrent access to data resources and specifies the degree to which transactions affect each other. When multiple users operate on a data table at the same time, each user's transaction operation should not be interfered with by others' transaction operations, and each user should be kept independent and non-interfering.

Persistence: Persistence means that when a transaction is submitted to the database, the changes made to the data in the database by this transaction are permanent, regardless of any failure or unrecoverable situation that the data encounters. In a centralized database, the control of transactions is relatively simple, because the control of transactions only involves one central node. When the data is inconsistent, the transaction can be rolled back or redone through the transaction log. However, in a distributed database system, the situation is much more complicated. It is necessary to ensure the integrity of

multiple copies of data on multiple data nodes at the same time, and to meet the ACID characteristics of transactions locally.

In a distributed database system, ideally, data is kept consistent in multiple child node databases through replication. When a data node updates data, the data of other nodes should also be updated, so that the data of multiple nodes can be kept consistent. Then a relatively simple solution is to define transactions in each node as local transactions, and global transactions as distributed transactions, including local transactions of all nodes. Local transactions are only responsible for modifying and updating a local database data. For example, the local transaction of node 1 is only responsible for the ACID characteristics of node 1, and the local transaction of node 2 is only responsible for the ACID characteristics of node 2. Only when all local transactions are completed can the global transaction be committed, so that the ACID characteristics of the global transaction can be maintained. Then the existing solution is the two-phase commit (2PC) technology. By introducing an intermediate negotiator to coordinate the behavior of the participants,

the negotiator decides whether the transaction participants perform the transaction operation.

1. The first stage (voting stage): The negotiator node asks all transaction participant nodes whether they can perform the commit operation (vote), and starts waiting for the response of each participant node. The participant node executes all transaction operations up to the time of the inquiry, and writes the Undo information and Redo information into the log. (Note: If successful, each participant has actually performed the transaction operation here) Each participant node responds to the inquiry initiated by the coordinator node. If the transaction operation of the participant node is actually executed successfully, it returns an "agree" message; if the transaction operation of the participant node is actually executed unsuccessfully, it returns an "abort" message.

2. The second stage (submission execution stage):

When the corresponding messages received by the coordinator node from all participant nodes are "agree": The coordinator node sends a "formal commit" request to all participant nodes. The participant nodes formally complete the operation and release the resources occupied during the

entire transaction. The participant nodes send a "completion" message to the coordinator node. When the coordinator receives the completion information from all participants, the transaction is completed. If the response message returned by any participant node in the first phase is "abort", or the coordinator node cannot obtain the response message of all participant nodes before the query timeout of the first phase: The coordinator node sends a "rollback operation" request to all participant nodes. The participant node uses the previously written Undo information to perform a rollback and release the resources occupied during the entire transaction. The participant node sends a "rollback completion" message to the coordinator node. After the coordinator node receives the "rollback completion" message fed back by all participant nodes, it cancels the transaction. At this time, regardless of the result, this phase will end. 2PC is a distributed transaction that tries to ensure strong consistency, so it is synchronously blocked, which leads to long-term resource locking problems. In general, it is inefficient and has single point failure problems. In extreme cases, there is a risk of data inconsistency. For this reason, the academic community introduced the 3PC

protocol, which is a non-blocking commit protocol. The non-blocking protocol mentioned here is only a relative model and cannot completely achieve non-blocking, but it can reduce blocking in a certain way.

2.2 Data Replication Technology

2.2.1 Data Replication Overview

Data replication refers to the process of replicating data across multiple different databases in a distributed environment. In this process, data from a node is copied from the node to another different node. This process is an important part of distributed computing solutions.

Database replication technology needs to meet the following characteristics:

1. Accurate data replication: If there are errors in the data replication process, the replication process is unreliable, and the accuracy of the data in the replication process must be guaranteed.
2. Online data query: The replication system should provide the function of querying at any time to verify the correctness and real-time nature of data replication.
3. Independent data replication: Database replication software cannot be installed in the main database and cannot affect the main database, otherwise it may have

unpredictable impact on the entire system.

2.2.2 Classification of replication technologies

1. According to the update transmission time, it is divided into synchronous replication and asynchronous replication.

(1) Synchronous replication: Synchronous replication is also called real-time replication. Synchronous replication is used to ensure data consistency. During the replication process, data copies are synchronized at any time. If a node fails to replicate at a certain time, the entire transaction will fail. Synchronous replication requires frequent communication between nodes, consumes a lot of resources, has low replication efficiency, and has a high probability of deadlock. The principle diagram of synchronous replication is shown in Figure 2.1 below.

生死锁的几率比较大。同步复制的原理图如下图 2.1 所示。

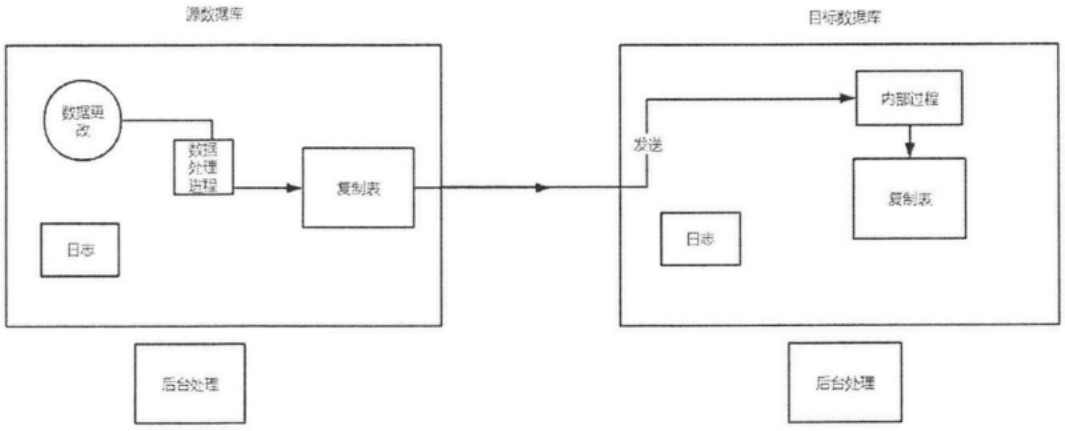

图 2.1 同步复制原理图

(2) Asynchronous replication: Asynchronous replication is also called lazy replication, which means that the source database can be updated at any time. The transaction does not need to ask the remote database node whether it can be submitted. Instead, it is saved locally first and will be replicated to other nodes at the appropriate time. At a specific time point, the data of each database may be inconsistent. When the system crashes, if a node cannot complete the replication work, asynchronous replication is still feasible, but the replication work will be carried out after the system returns to normal. The main disadvantage of this method is that it cannot guarantee data consistency at any time, but the data will remain consistent after all nodes have completed the update. If the transaction fails to replicate on a node, the transaction of that node will be rolled back and will not affect other replication nodes. Asynchronous replication has low requirements for network and device hardware, reducing communication volume. The schematic diagram of asynchronous replication is shown in Figure 2.2 below.

降低了通信量。异步复制的原理图如下图 2.2 所示。

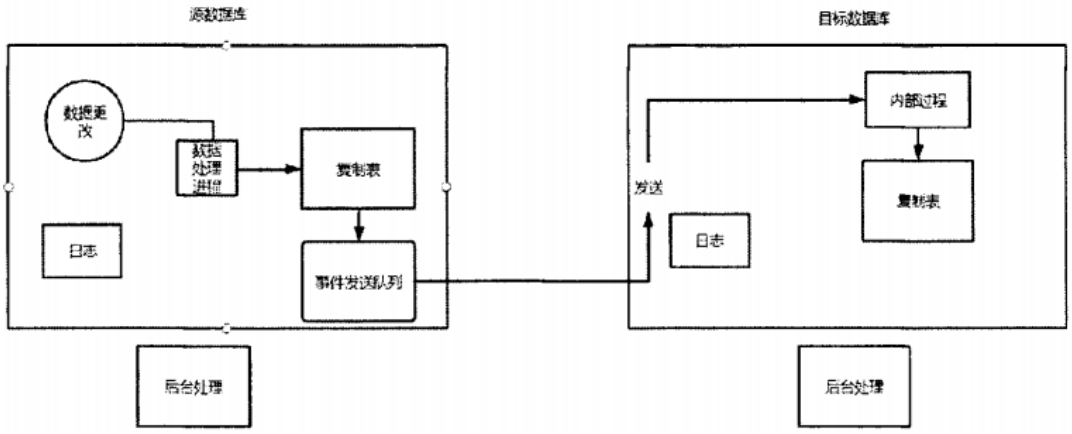

图 2.2 异步复制原理图

Synchronous replication focuses on consistency, while asynchronous replication focuses on practicality. However, asynchronous replication may lead to non-serial operations. For example, the local database updates data and retains it locally, but has not yet been transmitted to the remote database. However, the remote database modifies the data, which causes a conflict. Both the local data and the remote database data have been modified, resulting in a conflict. Therefore, current commercial databases generally use reconciliation mechanisms, such as using timestamps to add timestamps to each update. Each time an update operation is performed, the timestamps are compared. When an update conflict occurs, the valid update is determined based on the timestamp. As a result, most of the current mainstream

commercial databases use asynchronous replication to implement database replication.

Based on where the updates are performed, it can be divided into master-slave replication and peer-to-peer replication.

(1) Master-slave replication mode:

In the master-slave replication mode, there is only one master database, and the rest are slave databases. The master database receives update requests from each slave database, and then sends the data update to each slave database. In this mode, only the master database can receive update requests, and the slave database can only receive data from the master database. The performance of the entire system is limited by the master database.

(2) Peer-to-peer replication mode:

In the peer-to-peer replication mode, there is no master database or slave database. Each node can replicate data to other nodes, and each node has equal status. In this mode, since there is no distinction between primary and secondary nodes, different nodes may modify the same copy, which will cause conflicts and need to be handled.

According to the content and method of replication, it is divided into transaction replication, snapshot replication and merge replication.

(1) Transaction replication: The transactions generated during the source database replication process are copied to the target database. This type of replication replicates the data operation statements (INSERT, DELETE, UPDATE, etc.) of the source database. Transaction replication needs to ensure that the source data and the target database table structure are the same, and that the primary key exists and can execute SQL commands. After the transaction is copied to the target database, the target database needs to execute the received transaction in the local database to implement the same transaction operation as the source database. The process of transaction replication is to transfer the transaction and transfer the changed transaction to the target data. The network requirements are low and the performance requirements are not high because it does not need to copy data, thus saving a lot of resources.

(2) Snapshot replication: Snapshot replication refers to taking a photo of the data in the source database at a

specific point in time, and then generating a static file that records all the data in the source database at the current moment. This static file is copied completely according to the state of the data at the time of presentation. Snapshot replication does not require continuous "photographing" of the data in the source database, but rather takes photos at intervals and then completely copies the data in the source database to the target database. Therefore, the amount of data transmitted by snapshot replication is very large, and the network requirements are extremely high. Not only does it require bandwidth transmission speed, but the accuracy of the transmitted data must also be guaranteed.

(3) Merge replication: Merge replication is based on the "publish-subscribe" model, which has the following components:

Publishing Server: A publishing server is a server that can store all the data of a site, specify the specific data to be replicated and monitor the changes of the data during the data replication process. The publishing server is the server of the original owner of the data.

Distribution server: The distribution server is used to distribute data, record the distribution history of the

data after distributing the data, and record all synchronization records.

Subscriber: A subscriber is a server that receives data. It manages data updates and can decide whether to return data to the publisher or publish it to other subscribers based on data type and data options. A subscriber is a server that receives subscription data. Therefore, the subscriber is where the actual data is updated. The subscriber can choose whether to receive data. If it refuses, it returns it to the publisher. If it chooses to receive, it sends the data to other subscribers.

Release: A release is a collection of one or more items within a database.

Project: A project is the basic unit of replication, which can refer to a data table, a data object, certain rows, or a stored procedure.

Subscription: Subscription is a request for a replicated data copy or data object, including the specific publications to be received, the specific items to be received, and the specific method of receiving. In summary, merge replication is a subscription in the subscription server by the user, and then the subscription will monitor

the data changes in the database, merge the data changes and send them to the subscribers. If a user modification conflict is found in the data merge, merge replication will determine the user's permissions, and the user with permissions will successfully modify the data.

2.3 Sybase Replication Technology

This article implements a replication system based on the Sybase commercial database, so it focuses on the replication technology used by Sybase. Sybase is a relational database system that implements cross-network data replication based on a "publish-subscribe" model. Users "publish" the data available in the database, and then other users "subscribe" to the data and pass it to a replication database. Its replication modes are:

1. Basic master replication mode: The master replication mode sets up a master database, which sends updates to multiple slave databases.

2. Distributed primary segment: The replication server at each site distributes changes to local data to other sites and applies changes received from other sites to the locally replicated data.

3. Cooperative mode: This mode has multiple distributed master segments and a collective merge

replication table. The table on each master site contains only the data that is primary data for that site. No data is replicated to these sites. The cooperative table "collects" the data from each master site together.

4. Agency cooperation model: This model is similar to the cooperation model.

The master tables distributed on various remote nodes are concentrated into a table on the master site. At the site where these tables are merged, a replication agent is set up to process the merged table as master data. The merged data is then sent to the subscriber through the replication server.

Sybase supports replication between different databases through various interfaces, and can use any data storage system with basic data operation functions as a data server. Users use Sybase's replication agent to retrieve data from the data server and then replicate it to the target database.

2.4 Middleware-based replication architecture

The replication architecture based on middleware is different from the replication architecture of database system (i.e. the replication component built into the database system (or a thread that implements replication)).

The replication architecture based on middleware sets up middleware between databases. Users operate the middleware, and the middleware obtains the data of the master database and sends it to the slave database to achieve replication. However, since the middleware needs to maintain communication with the database, there will always be transmission loss, so the replication performance is not as good as the database system replication. Next, we will introduce a middleware product - SymmetricDS.

SymmetricDS is a platform-independent data synchronization and replication solution. SymmetricDS is written in Java and requires JRE or JDK6.0 and above. Any database with Trigger technology and JDBC driver can use SymmetricDS. The database is abstracted through Database Dialect to support different database features. SymmetricDS synchronizes through the trigger mode, records database changes in a system table, then establishes a buffer on the client, and regularly pushes changes to the receiving end. The receiving end and the sending end establish a channel, and regularly pull new data changes and play them back in the target database. This process will have certain delays and performance impacts, but as long

as the database supports triggers, SymmetricDS can be used for database synchronization.

A SymmetricDS engine is called a Node. The database connection information is configured by the database connection string, database username, and database password provided in the properties configuration file. SymmetricDS can synchronize any database table that the database connection can access, as long as the database user is assigned the appropriate database permissions. Database triggers enable SymmetricDS to capture data changes. SymmetricDS automatically installs triggers based on the user's configuration. Database triggers record data changes in the DATA table (DATA table is a system table in SymmetricDS). The database is designed to be non-intrusive and as lightweight as possible. After the SymmetricDS triggers are installed, all data changes generated by DML statesment executed by external applications will be captured. Note that the user's application does not need to add additional libraries or make any changes; SymmetricDS does not need to be online to capture data. Data is sent to remote nodes via HTTP or HTTPS. Data can be sent in one of two ways, depending on the type of transmission link between the configured node

groups. A node group can be configured to push changes to other nodes in a node group, or it can be configured to pull data from other nodes in a node group.

Figure 2.3 below shows the three nodes of the database. One node will represent a master database, and the other two nodes will represent two separate slave databases.

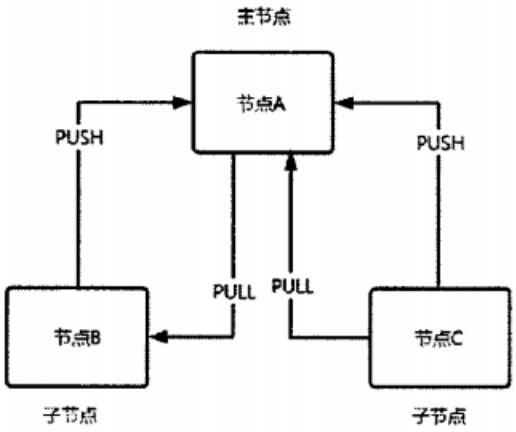

图 2.3Symmetry 流程图

2.5 Common methods of data replication

Given the different ways of capturing changes in data replication, data replication methods generally cover the following five categories: snapshot-based capture, trigger-based capture, log-based capture, timestamp-based capture, and API-based capture.

2.6 Data Conflicts and Solutions

In the process of asynchronous replication, each database accepts the modifications made to the data by other databases while also accepting the modifications made to the database by local transactions. In this process, the local database modifies the data, the system records the local transaction, and sends the local transaction to the remote target database for execution. However, due to the possible existence of a time window, it is very likely that the target database has modified the data after the local transaction is committed and before the remote target server is committed, which leads to data conflicts. In addition, due to the use of timed incremental replication technology and the lack of concurrency control mechanism, operations such as adding, deleting, modifying, and querying may cause conflicts in database systems with large concurrency. Therefore, conflict resolution methods need to be added in actual application development.

2.6.1 Data Addition Conflict

For conflicts caused by data addition operations, you can use table grouping. Table grouping means granting table addition permissions to a certain person or important decision maker in the table. For example, in a school or class, designate an adder for the entire school, and the

adder divides each class into set tables. As long as the intersection between different table sets is an empty set, there will be no data addition conflict.

2.6.2 Data Update Conflict

Update conflict refers to the situation when multiple copies of the same data are modified into different data by different people, resulting in inconsistent data. The principle of incremental database replication is to record the value before and after the modification when modifying the database data, find the database record of the value before the modification during the replication process, and then replace it with the modified value.

Resolve the corresponding conflicts :

1. Resolve conflicts through reasonable design procedures

In the process of designing an application, reasonable planning can be used to avoid conflicts. Different priorities can be set for different areas or different operation objects, and the ones with higher priorities will be operated first, thus resolving conflicts.

2. Resolve conflicts by grouping by field

When a record may have more than one updater, you can further group the fields, group the fields according to the

requirements, and specify a unique updater for each field group. In this case, the principle of database replication is: before updating a field, first record the key attribute value in the record and the values of each field in the field group where the updated field is located, and then record the key attribute value and the new value of each field in the field group. When performing the replication operation, search and update based on the key attribute value and the original value of each field in the field group.

2.6.3 Data Deletion Conflict

Weak consistency can also cause deletion conflicts in database data. Data deletion conflicts refer to conflicts caused by the deletion of data from one of the multiple copies of a record when the value of the data is modified by another copy or copies. Since the deleted copy data has been deleted, the data cannot be copied under the premise of incremental replication. However, the deletion conflict problem can be solved by adding a deletable marker. By adding a marker to indicate that the data can be deleted, if someone encounters this data in the future and finds that it is a deletable marker, the data modification operation will not be performed. Therefore, data with a deletable

marker will not cause data deletion conflicts. This method of adding a marker can solve the problem.

Chapter 3 Requirements Analysis and Overall Planning of Distributed Database Replication System

3.1 Demand Analysis

3.1.1 Data consistency requirements

(1) Strong consistency requirements: In some key business scenarios, such as transfer operations in financial transaction systems and order status updates in order processing systems, the distributed database replication system must ensure that the data of all

replicas remains completely consistent at all times. This means that no matter which replica node the user reads data from or performs a write operation, the result obtained should be the same as the result of the operation on other nodes. For example, in the bank transfer process, if the balance of an account on a certain node is updated, then the balance of the account on all other replica nodes should also be updated immediately and remain consistent to avoid miscalculations of funds or transaction disputes caused by inconsistent data.

(2) Eventual consistency requirements: For some business scenarios that do not have particularly strict requirements for real-time consistency, such as user dynamic updates on social media platforms and article view counts in news and information applications, it is possible to allow inconsistent replica data for a certain period of time, but consistency must be achieved in the end. For example, in social media, after a user posts a new dynamic, users in different regions may see the update of this dynamic in their local copies at different times, but as the system's automatic synchronization mechanism operates, the dynamic data in all copies will eventually remain consistent. This eventual consistency requirement

can enhance the performance and availability of the system to a certain extent, because there is no need to perform strong consistency synchronization operations every time the data is updated, which reduces the system's overhead and response time.

3.1.2 Availability Requirements

(1) High availability requirements: Distributed database replication systems need to be able to continue to provide data read and write services stably in the face of various types of failures (including node failures, network failures, storage failures, etc.). For example, when a node shuts down due to hardware failure, the system needs to be able to automatically transfer the workload of the node to other normal nodes to ensure that the user's business operations are not disturbed. During the e-commerce promotion period, a large number of users access the product database at the same time. If some nodes fail, the system needs to be adjusted quickly to ensure that users can browse product information, place orders, and perform other operations smoothly, avoiding economic losses caused by system unavailability.

(2) Fault recovery requirements: Once a system fails, whether it is a single node failure or a local network

failure, the system needs to be able to recover quickly. After the fault is eliminated, the failed node needs to be able to automatically rejoin the system and synchronize data with other nodes to restore to normal working status. For example, in a distributed database cluster, if a node is temporarily offline due to network fluctuations, when the network is restored, the node needs to be able to automatically obtain data updates that occurred during the offline period from other nodes and become a normal member of the cluster again to ensure the integrity and availability of the entire system.

3.1.3 Performance Requirements

(1) Read and write performance requirements: The system needs to meet the performance requirements of concurrent read and write operations of large-scale users. For read operations, data should be obtained from local or nearby replica nodes as quickly as possible to reduce the delay in data transmission. For example, in a scenario where a content distribution network (CDN) is combined with a distributed database, when a user accesses web content, he or she should be able to quickly read data from a database replica node that is close to their own geographic location to increase the page loading speed. For write operations,

the delay of write operations should be minimized while ensuring data consistency, thereby increasing the system's throughput. In the order processing system of a large e-commerce platform, there may be a large number of order write operations every second. The system needs to process these write operations efficiently to prevent delays in order processing due to write operation delays, thereby affecting the user's shopping experience.

(2) Data synchronization performance requirements: During the data replication process, data synchronization between replica nodes should be carried out efficiently to minimize the impact on the overall system performance. Data synchronization should be able to automatically adjust the synchronization strategy and bandwidth usage based on factors such as network conditions and node load to achieve the best performance balance. For example, in an enterprise's internal distributed database system, there may be differences in network bandwidth between nodes in different departments. The system should be able to prioritize the synchronization speed of key business data between high-bandwidth link nodes based on actual circumstances, while reasonably arranging the synchronization tasks of low-bandwidth link nodes to

ensure that the data consistency and performance of the entire system are not overly affected.

3.1.4 Scalability Requirements

(1) Node expansion requirements: As the business develops and the amount of data increases, the system should be able to easily add new nodes to the distributed database cluster. After the new node is added, it should be able to automatically synchronize data with the existing nodes and share the system's read and write load. For example, in a rapidly developing Internet social platform, as the number of users continues to increase, the original database nodes may not be able to meet the growing storage and performance requirements. At this time, the system's capacity and processing power can be expanded by adding new database nodes. The new nodes can be quickly integrated into the system and start working without the need for large-scale downtime adjustments to the entire system.

(2) Data volume expansion requirements: Distributed database replication systems should be able to adapt to the ever-increasing amount of data. Both structured data (such as table data in relational databases) and unstructured data (such as documents, pictures, videos, etc.) should be efficiently stored and managed in the system. For example,

in a big data analysis platform, a massive amount of new data is generated every day, including user behavior data, sensor data, etc. The system needs to be able to dynamically expand storage capacity and data processing capabilities to cope with the explosive growth of data volume while ensuring data reliability and consistency.

3.2 Overall planning

3.2.1 Architecture Design

(1) Master-slave replication architecture: When the master-slave replication architecture is adopted, there is a master node (Master) and multiple slave nodes (Slave) in the system. The master node is responsible for processing all write operations and synchronizing data changes to the slave nodes in the form of logs or transactions. The slave nodes are mainly responsible for processing read operations, thereby achieving read-write separation and improving system performance. For example, in an online game database system, the master node receives players' game data update operations, such as character level upgrades and equipment acquisitions, and then synchronizes these updates to the slave nodes. The slave nodes are responsible for processing read operations such as other players' queries for the character information, which can

reduce the load on the master node and improve the overall read and write efficiency of the system. The advantages of this architecture are that it is relatively simple to implement and data consistency is easy to control; the disadvantages are that the master node may become a performance bottleneck and there is a risk of single point failure.

(2) Peer-to-peer replication architecture: In a peer-to-peer replication architecture, each node has equal status and can initiate read and write operations. Nodes coordinate data replication and consistency maintenance through consistency algorithms (such as Paxos, Raft, etc.) or message passing mechanisms. For example, in a distributed file storage system, each node can store file data and communicate with other nodes to synchronize data updates. When a file on a node is modified, it broadcasts the modification information to other nodes, and other nodes perform corresponding update operations after receiving the message. The advantages of this architecture are that there is no single point of failure, high system reliability, and good scalability; the disadvantages are that data consistency maintenance is complex and difficult to implement.

(3) Hybrid architecture: A hybrid architecture is designed by combining the characteristics of master-slave replication and peer-to-peer replication. For example, under normal circumstances, the system uses a peer-to-peer replication architecture to improve performance and scalability. When data conflicts or system failures occur, it switches to master-slave replication mode and uses the authority of the master node to quickly restore data consistency. In a distributed database system of a large enterprise, the database nodes of various departments can work in a peer-to-peer manner and share data read and write loads. However, when a department performs large-scale data updates or the system detects data inconsistencies, the system automatically designates a master node to coordinate data synchronization and repair to ensure data accuracy and consistency.

3.2.2 Data replication strategy settings

(1) Full replication strategy: When the system is initialized or a new node is added, a full replication strategy is adopted. That is, the new node copies all data from the source node or other replica nodes. For example, in a newly built distributed database cluster, the newly added node needs to obtain the initial data of the entire

database. At this time, full replication can be used. The advantage of this strategy is that it can obtain a complete copy of the data at one time to ensure the integrity of the data; the disadvantage is that when the amount of data is large, the replication process may consume a lot of network bandwidth and time, which has a great impact on system performance.

(2) Incremental replication strategy: During the operation of the system, an incremental replication strategy is adopted for daily data updates. That is, only the data that has changed since the last replication is replicated. For example, in an e-commerce database, each time the product information is updated (such as price adjustment, inventory change, etc.), only the changed data is synchronized to other replica nodes instead of re-copying the entire product database. This strategy can greatly reduce the network overhead and time of data replication and improve the performance and efficiency of the system. However, it is necessary to accurately record the changes in data and establish an effective incremental data transmission mechanism.

(3) Log-based replication strategy: Data replication is achieved by recording logs of database operations (such

as binary logs, redo logs, etc.). The master node sends the operation log to the slave node or other replica nodes, and the slave node updates its own data copy based on the operation instructions in the log. For example, in the master-slave replication of the MySQL database, the master database sends the binary log to the slave database, and the slave database parses the SQL statements in the log and executes them, thereby achieving data synchronization. The advantage of this strategy is that the data replication is highly accurate and can ensure data consistency; the disadvantage is that the management and parsing of the log requires certain computing resources, and if a failure occurs during the log transmission process, it may cause data synchronization delays or failures.

3.2.3 Consensus Algorithm Design

(1) Application of Paxos algorithm: Paxos algorithm can be applied in scenarios with extremely high consistency requirements. For example, in a distributed financial trading system, in order to ensure the consistency of each transaction on all replica nodes, the Paxos algorithm is used to coordinate decisions between nodes. When there is a transaction request, each node determines whether the transaction is accepted and how to update the data replica

through multiple rounds of message transmission and negotiation of the Paxos algorithm, thereby ensuring strong consistency of transaction data in the presence of node failures and network delays.

(2) Application of Raft algorithm: For general distributed database systems, especially scenarios that focus on system availability and performance balance, the Raft algorithm can be used. In a distributed storage system, nodes elect a leader through the Raft algorithm. The leader is responsible for receiving read and write requests from clients and copying data update logs to other nodes. Other nodes act as followers and update data copies according to the leader's instructions. When the leader fails, the system quickly selects a new leader through an election mechanism to ensure the continuous operation of the system. For example, in a distributed file system, the Raft algorithm can effectively manage the replication and consistency maintenance of file data, and can quickly restore system functions when a node fails, thereby improving the overall reliability and availability of the system.

3.2.4 Fault handling mechanism design

(1) Node failure handling: When a node failure is detected, the system should immediately start the fault handling mechanism. For the master-slave replication architecture, if the master node fails, a new master node needs to be elected from the slave nodes. During the election process, the consistency and integrity of the data must be ensured to avoid the expansion of system failures due to data confusion during the election process. For example, in a database system based on master-slave replication, after the master node goes down, the slave nodes select a slave node with the latest data and better performance as the new master node through heartbeat detection and election algorithm, and notify other slave nodes to update the master node information. Then the new master node begins to receive write operations and synchronize data to the slave nodes. For the peer-to-peer replication architecture, when a node fails, the other nodes will automatically adjust the data replication strategy to bypass the failed node for data synchronization. After the failed node recovers, it will be included in the data synchronization scope again.

(2) Network failure handling: When a network failure occurs (such as partition, packet loss, delay, etc.), the system

should be able to adjust the data replication strategy according to the network conditions. If communication between nodes is interrupted due to partial network partition, the system can adopt a local priority strategy within the partition to ensure the availability and consistency of data within the partition, while trying to repair the network connection and synchronize data after the network is restored. For example, in a distributed database system that spans multiple data centers, if the network connection between two data centers fails, the nodes within each data center can continue to read and write data and maintain data consistency locally. When the network is restored, cross-data center data synchronization is performed to ensure data consistency across the entire system. In the case of severe network packet loss or delay, a cache and retransmission mechanism can be used to ensure the integrity and accuracy of data replication.

Chapter 4 Design and Implementation of Replication System

As the core element of the replication system, the replication server has two key components: the change data capture component and the data transmission component. The change data capture component is mainly used to obtain the data or operations that need to be processed in the database, while the data transmission component is used to transmit the changed data to the target database. In this section, the design of these two components and the implementation of the replication system based on the SYBASE database will be described in detail.

4.1 Overall Design Ideas

Research is conducted on the data replication system based on data changes, with the focus on properly handling the issues related to data replication in distributed systems

under the master-slave replication model. For example, suppose that in a large enterprise, there are multiple sub-data centers within the enterprise, the main site is set up at the headquarters, and there are three sub-data centers, namely A, B, and C. There is a type of data that is updated and modified by the company headquarters based on relevant data, so the three sub-sites need to obtain the same data copy from the head office. At this time, there is such a situation that if a sub-site fails and cannot receive data transmission, how should the data be restored if the node can be repaired in the future? In the design, snapshots of the master database at different times are recorded according to the timestamp. When recovering from the database, the snapshot is found according to the timestamp corresponding to the time of the failure for recovery. In this way, the design of the timestamp is particularly important. The timestamp is a unique value generated by the system based on the system time. Its size depends on the order of the system time. Every time the data is updated, the timestamp is added to the transaction table. The transaction table will be introduced below. By using timestamps, we can use timestamps to determine whether data has changed and whether data needs to be

replicated. In another case, if the master site fails, a subsite needs to be able to become the master site to provide services for other subsites in the entire replication system, and it needs to be able to flexibly add or delete subsite nodes to achieve reasonable deployment, so a complete routing design is crucial. How to capture data when it changes and how to transmit it between the master database and the slave database are the core contents of the design. The following will introduce the design of the data change capture component and the data transmission component in detail.

4.2 Design of Data Change Capture Component

The data change component captures the operations performed in the transaction table and the data objects of the operations, and stores the corresponding transaction information in the control table. The data change capture component contains two subcomponents, one is the capture operation component, which can capture specific operation types, namely insert, update, and delete operations, and the other is the data storage component, which is used to store the data corresponding to the operation and transaction information in the control table.

4.2.1 Control Table Design

The data capture method based on the control table method is the most effective. In view of the disadvantages of the global control table, which has large resource consumption and low performance, a new improved method is designed, that is, to create a dedicated control table for each replicated table. This method has better performance. The information that needs to be recorded in the control table is as follows:

1. The operation information in the control table needs to be reproduced when it is transmitted to the target database, so the control table must at least cover the type of operation and the specific data of the operation.

2. If the target database fails during the replication process, the data cannot be copied to the target data table, so the operation needs to be rolled back locally to restore the data to the previous state, so the control table also needs to record the data before the operation. When the operation is inserting, the data table does not need to record the previous state, and the previous state is empty, so there is no need to save the previous data. But when the operation is deleting and updating, the information in the

data table changes, so the previous data content needs to be recorded.

3. Based on the above requirements, the content of the control table should be as concise as possible, and the data successfully copied to the target data table should be deleted from the control table. Based on these requirements, this article adopts the following improvement method, which subdivides the control table into transaction table and archive table. The transaction table is used to store the transaction information to be executed on the target database and the primary key information involved. The archive table is used to store the data before the update and delete operations. Configure a transaction table and archive table for each replication table. Since the control table is divided into two tables, an additional table needs to be designed to record the mapping relationship between the two tables. The transaction table, archive table and mapping table will be designed separately below.

(1) Transaction table design

The purpose of the transaction table is to ensure that the transaction information on the table can be executed with the same transaction operation in the target database,

so the transaction table needs to record: the primary key of the changed data, specific transaction information, and whether the primary key is updated. Transaction information covers specific transaction numbers, sequence numbers, operation types, and timestamps. The transaction number is the unique identifier of the transaction, represented by a field, but when an operation in a transaction involves many records, it is impossible to locate the specific record, and an additional sequence number is required to identify each specific record. The transaction number and sequence number can uniquely determine the specific data information. Operation types include insert, update, and delete. Timestamps are used as a comparison mark to resolve data conflicts when data conflicts occur. They will be described in detail below. Whether the primary key is updated will affect the results of replication in a distributed environment, and whether the primary key is updated also needs to be recorded. The structure of the transaction table is shown in Table 4.1.

表 4.1 事务表结构

字段名称	数据类型	说明	备注
Key	与复制表保存一致	复制表主键	来自复制表
Transaction_id	Integer not null	事务号	
Sequence_id	Integer not null	序列号	
Timestamp	Data	时间戳	
Trans_types	Integer not null	操作类型	1=插入 2=更新 3=删除
Is_key	Bool	主键是否更新	0=未更新 1=更新

（2）归档表设计

归档表是为了当更新或删除操作时记录需要还原的数据，所有归档表中需要记录两部分信息：更新或删除前的信息以及事务信息。通过事务信息就可以将归档表唯一找到控制表，也可以通过事务信息唯一找到具体数据。归档表设计结果如下表 4.2 所示。

表 4.2 归档表结构

字段名称	数据类型	说明	备注
Data	继承自复制表	复制表的所有数据信息	来自复制表
Transaction_id	Integer not null	事务号	
Sequence_id	Integer not null	序列号	

4.2.2 Capture operation module design

The capture operation module must ensure that it can capture immediately when the source data table is updated. Generally speaking, general middleware often adopts the form of triggers, but this method is very inefficient. The most effective way is to rely on the support of applicable algorithms inside the database to capture. There is no need to set up additional triggers. By designing a marker inside the data structure, the marker is processed when the database carries out the query plan. This is the processing method adopted by this system. This module designs a key

data structure - the information body structure. The information body structure contains the table name in the data table, the table session information, the actual data address and the copy flag. The copy flag is the most critical part. If the copy flag is 1, it means that the table participates in the copy, so it will be captured by the capture operation module. The information body structure is shown in Figure 4.1 below.

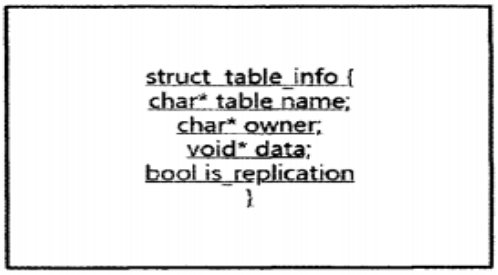

图 4.1 信息体结构

4.2.3 Data Storage Module

After the capture operation module captures the specific operation, it needs to perform different processing according to different operation types and then store them.

1. Insert module

The implementation of the insert module is relatively simple. When the captured operation is insert, the captured

primary key information, operation type, transaction number, sequence number and other information are stored in the transaction table. Insertion has nothing to do with the source data, so the changed data is the inserted new data, and there is no need to record any information in the archive table.

2. Update module

The update module involves whether the primary key is involved in the update. If the primary key is involved in the update, it means that the table structure is different, so transaction replication cannot be performed. Under normal circumstances, the old data needs to be deleted and the new data needs to be inserted. If the primary key is not involved in the update, then the transaction replication is performed directly to update the data of the target database. However, this traditional method increases the complexity of the system . This system also considers whether the primary key is involved in the update, but there is no need to delete the old data and insert the new data. In view of the design of the archive table, the data and transaction information are stored in the archive table before the update. The update statement is executed in the copy table in the local database, and

the updated primary key information, operation type (update) and transaction information are saved together in the transaction table. If the primary key is updated, the transaction information of the transaction table and the archive table is the same. The new primary key information is located in the transaction department, and the archive table stores the old primary key information. The transaction table and the archive table are combined, and the old primary key information is obtained from the archive table. The specific data information can be found on the target data table without inserting and deleting operations. This simplifies the operation and makes the system processing more concise. The data storage process is shown in Figure 4.2.

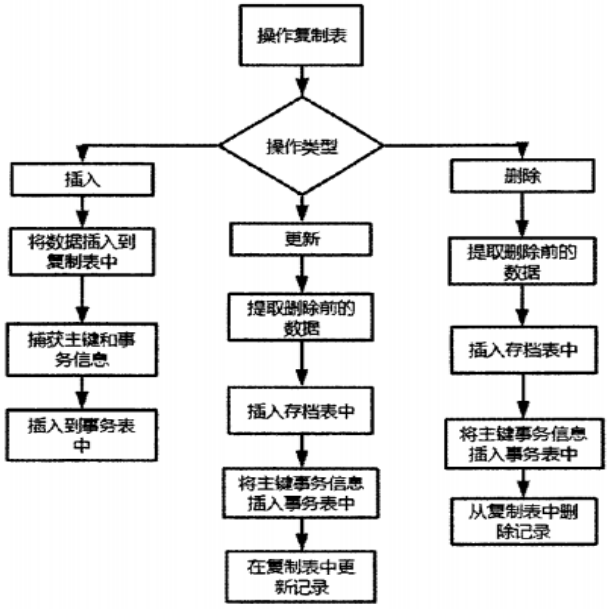

图 4.2 数据存储流程

4.3 Data transmission component design

After obtaining the changed data, the changed data needs to be transferred to the target database, so a data transmission component needs to be designed. The data transmission component needs to clarify which target databases the changed data needs to be sent to when data changes occur in the local database. The specific transmission is completed by the replication server, and a rule needs to be formulated for the transmission of the data path. The data transmission path needs to record the information of the source database, the target database,

and the replication table. The database information must include the specific database name and host IP. The source database information is sent to the target database through the data transmission path.

The designed data transmission rules are shown in Table 4.3.

表 4.3 数据传输规则表

字段名	类型	说明	备注
Master_db	Varchar(256)	源数据库名	包含路径信息及数据库名称
Master_ip	Varchar(256)	主机 IP 地址	
Target_ip	Varchar(256)	目标 IP 地址	
Target_db	Varchar(256)	目标数据库名	包含路径信息及数据库名称
Server_no	Integer not null	复制服务器编号	
Table	Varchar(256)	复制表	

4.3.2 Replication Server Design

The replication server is the core of this system, which includes a series of replication service processes. Through the replication server, the source database and the target database can be connected, and the data update can be copied to the data table of the target database according to the content on the replication table. This system has selected a transmission solution based on an intermediate

agent, and the replication server needs to fully support this solution. The replication server can be deployed on any server when the network condition is good, and it needs to have no dependency on any database. Therefore, it is necessary to configure relevant parameters for each replication server. When the data replication thread is started, the configuration parameters are read to make a specific connection. The parameters include: the number of the replication service process, which is unique and can be used to find the corresponding server. The propagation cycle, which specifies the working cycle of the data replication process, and the replication service process will run at fixed intervals according to the time of the propagation cycle. Through the replication server configuration parameters, the corresponding server can be found uniquely, and then the corresponding transaction table and replication table under the server can be found. The replication server will regularly access the replication table and replication table under the server, and will connect to the target database if a new transaction is generated. The interval of regular access is determined by the parameters. The entire process of the replication

server is shown in Figure 4.3.

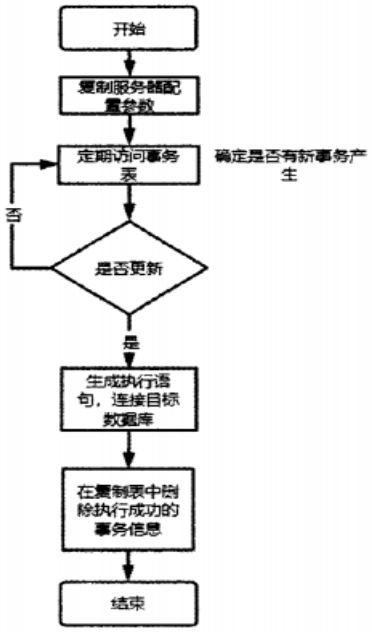

图 4.3 复制服务器工作流程

4.3.3 Data transmission process design

This process is the core functional module of the entire replication system. Based on this data replication process, data synchronization from the master database to the slave database can be achieved. First, the slave database sends a subscription application to the replication server of the master database through the replication server. If the master database fails to detect the application, the connection will not be initiated and the data replication will be canceled. If the master

database detects the application successfully, it passes the validity test. Since transaction replication is used, transaction information is stored in each replication table. The system will then determine whether the current replication table in the slave database belongs to a newly added data table, that is, the current replication data does not exist in the slave database, then the data replication process in the slave database system will clear the replication table, and the backup database will apply for the transmission of complete data through the replication server. If it is detected that the replication table already exists, the replication operation will continue. So if the table structure changes, how to perform the replication operation. At this time, the modification flag field value in the replication table will be used to determine whether the table structure has changed. If it has changed, the data structure of the record table in the child database will be allowed to change. The flag field value will be used to determine whether to accept the change in the table structure. If the field value is true, it means that the data table structure is allowed to change, and the replication continues. Because the design table structure changes, the table data of the local database must be

deleted. The local replication server initiates a subscription application, and the master database sends the complete table information to the replication table and then generates a new table locally through the data replication process at the end of the replication. The data replication process will obtain the maximum timestamp information of the current data from the replication table and write it to the local information table for the local timestamp value of the next replication data. If the field value is false, it means that the data table structure is not allowed to change, and the data replication operation will terminate. The flow chart is shown in Figure 4.4 below:

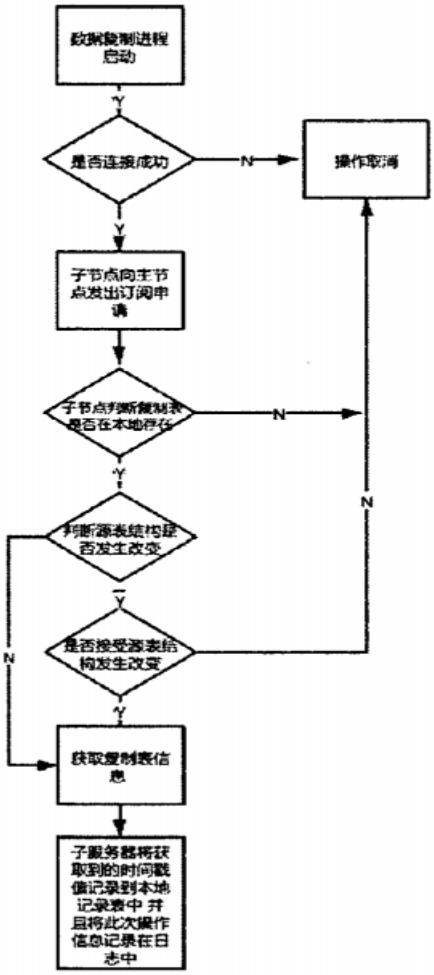

图 4.4 数据复制进程流程图

4.3.4 Record sheet design

In the process of data transmission, how to record the local timestamp and how to recover the data when the machine fails requires the use of the global record table of the local database. The information in the global record table is extremely important. It exists on any database server

in the entire system and records important information of the server. It is an important basis for network communication and the record table needs to meet the following three conditions.

1. It is necessary to ensure that the operation can be implemented on the local database based on the operation information or transaction information in the record table. The record table needs to contain the operation information or transaction information, as well as the changed data.

2. If a sub-database fails, the control table on the sub-replication system records the last timestamp and operation information before the failure, waiting for the recovery of the sub-database. If the operation is modification and deletion, the data before the change needs to be recorded. If it is insertion, there is no need to save the previous information.

3. Based on the above requirements, the design of the record table should control the information as little as possible. The record table is a table used to record local data. After the replication table is transferred, it needs to be joined with the replication table, which takes a long time. Therefore, the size of the control table determines the

duration of the operation. The design of the record table is shown in Table 4.4 below:

表 4.4 记录表设计

列名	含义
server_dbalias	数据库别名
server_description	数据库描述
server_dblink	数据库链接表示
server_host	主机 IP 地址
server_dbname	数据库名字
server_available	数据库有效标志
Repc_tab_part_replication_condition_expression	数据复制过滤条件
Repc_tab_structure_changed_permission_sign	复制表结构改变标志
Repc_tab_purge_timestamp	数据物理副除时间戳

4.4 Replication of emergency procedure design

When the master server encounters a devastating problem, a sub-server must be selected as the master server in the current system according to actual needs, and other sub-servers must switch from the damaged master server to the new master server to obtain data to ensure the normal operation of the entire replication system. So, how to ensure that the data of the current master server is the latest? This requires sending a request to the latest copy in the current system to obtain data so that the current master server has the latest data. Next, a program will be

set up to enable the master server to obtain the most recent data copy and set the current timestamp of the master server to the timestamp of the latest data. When a sub-server is determined to be the master server, it will send information to each sub-node in turn through routing based on the timestamp information in the local record table, and compare the timestamp value of the last replication of each sub-node. If the sub-node with the largest timestamp and its corresponding sub-server are determined, and this timestamp value is greater than the timestamp value of the master node, it means that the data of the current sub-node is newer than the master server, then it is necessary to immediately synchronize and update the data table of the master server. After completing the data replication work, the timestamp in the local record table is updated. The example is shown in Figure 4.5 below.

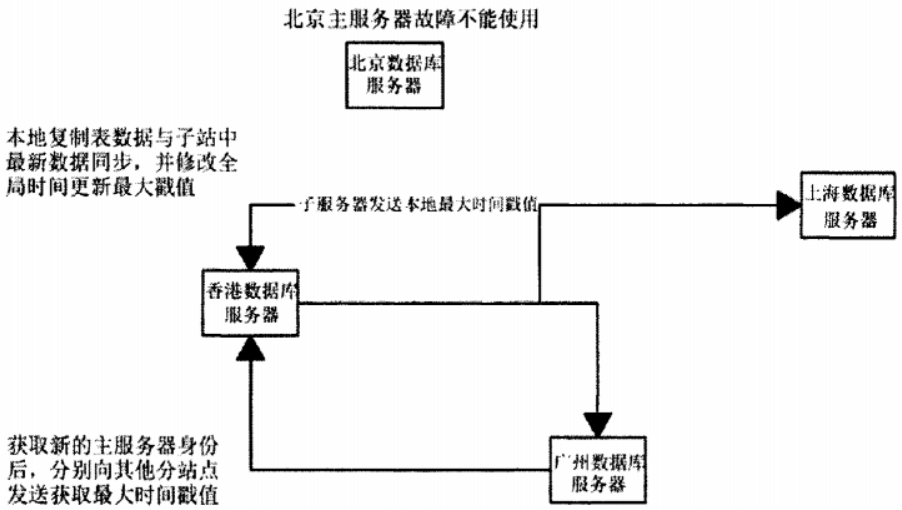

图 4.5 新增主服务器同步实现全局时间戳设置

4.5 Routing Management

In actual applications, enterprises usually have more than one slave database, and there may be multiple slave databases. Moreover, the data flow does not all originate from the master database. Perhaps the data of a slave database comes from another slave database. When the master database fails, a slave database needs to quickly take on the role of the master database to ensure the normal operation of the entire distributed database system. Therefore, it is necessary to plan reasonable routing management. Routing is a one-way message flow from the source replication server to the target replication server. You can create a route for each replication server.

When creating a route, the source replication server will do the following:

1. Create an outbound queue to hold messages for the target node.

2. Starts a thread that logs in to the target replication server and transfers transactions from the outbound queue to the target replication server.

3. Before creating routes, you need to distinguish which routes are direct routes and which routes are indirect routes. Indirect routes pass messages to the target replication server through one or more intermediate replication servers. Whether to use direct routes or indirect routes will have a significant impact on system performance.

4.5.1 Direct Routing

Routes without intermediate nodes are called direct routes. Systems with direct routes build network connections between source and target replication servers. For example, in the following figure, a five-node enterprise is presented as a star configuration with one master node and four replication nodes. Each replication node has a route from the master node. All four routes from

the master node are direct management routes. Therefore, the master replication server has four stable queues and four threads connected to the four replication nodes through the network. If replication node E wants to submit a request function to master node D, in addition to the direct route from E to A, the system will also need a direct route from A to D. This is shown in Figure 4.6 below.

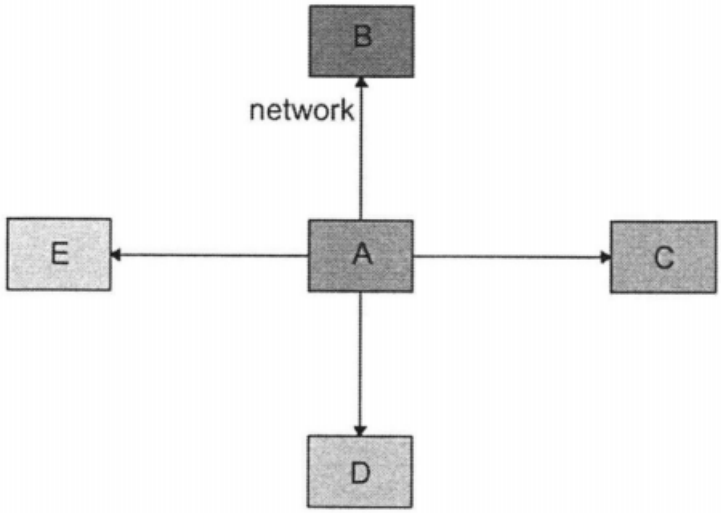

图 4.6 通过直接路由配置连接的节点

4.5.2 Indirect Routing

Routes with intermediate nodes are called indirect routes, that is, the route connection is not directly connected, but needs to go through an intermediate connection. The example diagram is a replication system with 5 nodes, with a master node and 4 slave nodes. Each

slave node has a route from the master node, but only two nodes are direct routes, and the other two are indirect routes. As shown in the figure, nodes A to B and A to C are direct routes, nodes B to D and C to E are direct routes, and nodes A to E and A to D are indirect routes. In an indirect route, the master node sends the message sent to the target node to an intermediate node, which uses a route (direct or indirect) to connect to the target node. To build an indirect route, create a direct route between each consecutive replication server in the direction of the expected indirect route. Once all direct routes are in place, you can create indirect routes. For example, to create an indirect route from A to C, first create direct routes from A to B and B to C. Then create indirect routes based on the existing direct routes. By setting up indirect routing, you can reduce the processing workload on the primary node and spread the load in the middle.

表 4.5 直接路由和间接路由路径表

直接路由	间接路由
A 到 B	A 到 D
A 到 C	A 到 E
B 到 D	
C 到 E	

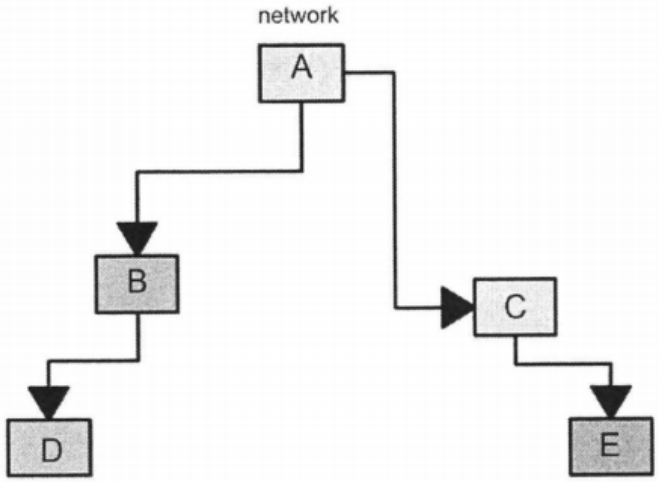

图 4.7 通过直接路由配置连接的节点

When using indirect routing, the master replication server is able to deliver the portion of the subscription that is common to each target node through the same intermediate node. When subscriptions overlap, the master replication server is required to send only one row-modified message to the intermediate replication server that is common to each target node. In this example of nodes with overlapping subscriptions, whenever a change

occurs at headquarters A, the intermediate replication servers in B and C receive the row modification. The branch replication nodes at B and C also need to perform A's modifications. Because B and C are set up to distribute changes to D and E, the A master replication server sends only one copy of each change to B and C. By utilizing two indirect routes from B to D and from C to E, the number of direct routes is also reduced. Although indirect routing helps distribute computing resources among nodes in the network, the overall speed of data transmission is somewhat reduced because messages are queued by multiple replication servers. Direct routing is best when the number of replication nodes is small. When using indirect routing, the number of intermediate nodes should be minimized to achieve the best transmission time. When using indirect routing, the master replication server is able to deliver the portion of the subscription that is common to each target node through the same intermediate node. When subscriptions overlap, the master replication server is required to send only one row-modified message to the intermediate replication servers that are common to each target node. In this example of nodes with overlapping subscriptions, whenever a change occurs at headquarters A,

the intermediate replication servers in B and C receive the row modification. The branch replication nodes at B and C also need to perform A's modifications. Because B and C are set up to distribute changes to D and E, the A master replication server sends only one copy of each change to B and C. By utilizing two indirect routes from B to D and from C to E, the number of direct routes is also reduced. Although indirect routes help distribute computing resources among nodes in the network, the overall data transmission speed is reduced to some extent because messages will be queued by multiple replication servers. When the number of replication nodes is small, it is best to use direct routes. When using indirect routes, try to minimize the number of intermediate nodes to achieve the best transmission time.

4.6 Implementation of Sybase-based Database Replication System

4.6.1 Replication System Architecture

The overall architecture is divided into four layers, from top to bottom:

The first layer is the tool layer, which covers core tools and third-party tool categories. It can use command line tools to pass messages to the second-layer modules.

At this layer, you can create a database, connect to a database, or execute SQL and other commands. This tool layer is at the top layer. Routing management can be configured at this layer, different databases can be created under different nodes, and connection operations can be performed on databases of different nodes.

The second layer is the database's external programming interface, which includes programming interfaces and communication modules. The second layer inherits the functions of the upper layer, achieves some underlying communication protocols, such as TCP/IP, and encapsulates the underlying database connection interface.

The third layer is the database layer, which includes the servers in the database. It receives various communication requests from the upper communication module and then sends the requests to the underlying database. It covers communication servers, DBMS servers (connecting to Sybase databases) and external access servers (for connecting to non-Sybase servers)

The fourth layer is the database kernel layer, in which the operation acquisition module can be implemented.
The architecture design of each module is shown in Figure

4.8:

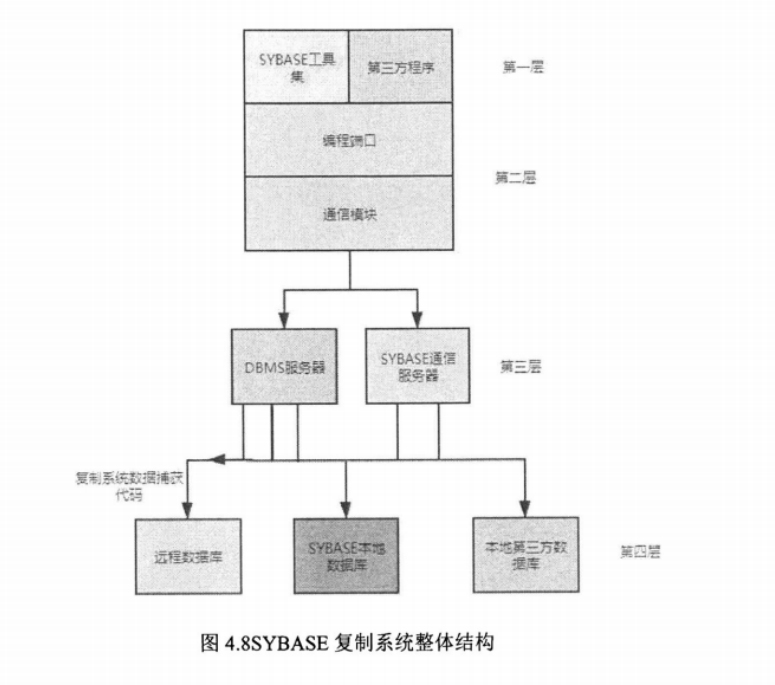

图 4.8SYBASE 复制系统整体结构

4.6.2 Calling the programming interface

The main code of this system is concentrated in the first layer of the tool layer of the architecture. It needs to use programming interfaces and communication modules to communicate with the third layer database. Through the programming interface, it can connect to the server and then execute the written SQL commands. In the C/S mode (client/server) operation, the client sends a request to the server, and then the server starts the database response request. The server returns the successful or failed operation information of the database to the client.

If successful, the relevant data after the operation is returned, and if failed, the failure information is returned. However, it may take several seconds for the replication request to be sent from the client to the server to receive the response returned by the server. In the past synchronous replication processing, the client cannot perform any operation before receiving the response information returned by the server. However, in the asynchronous processing mechanism adopted by this system, there is no need to wait for the response returned by the server, and the next operation can be continued, which greatly improves the performance of the entire system. Therefore, the asynchronous processing mechanism is adopted. When the replicator completes its non-database operation, the control is returned, and the function can be used to complete the request to the database. The function takes the control away from the replicator, and the replicator enters a waiting state until the database operation is completed or exceeds the user-defined response time limit.

When achieving the designed system functions through calling programming interfaces, it is key to consider

capturing changed data, distributed transaction processing, and processing of vector map data.

1. Capture change data through programming interface calls.

This process captures change data and stores it in the replication table. There are mainly the following steps.

(1) Establish a connection with the server

First, establish a connection with the server through the corn function. The server can be any DBMS server. The connection parameters can be configured through the set conn parm function. The duration and mode of the connection can be set. When communication with the server is no longer needed, the connection can be disconnected by calling the disconn function.

(2) Data updating and acquisition

Common data operations include query, insert, update, and delete. You can use query to perform query operations, insert to implement data insertion operations, update to implement data update operations, and delete to implement data deletion operations. After the replicator calls the query function, it will query the specific information of the data. For the insert operation, it is also necessary to specify the format information of the data by setting

the length, precision, range, data type, etc. Finally, the defined actual data is sent to the server by calling the bind function. For ordinary data records of non-vector data, the format information of the defined data can be returned by calling the get function, and the flowchart of the data update capture programming interface call can be shown in Figure 4.9 below.

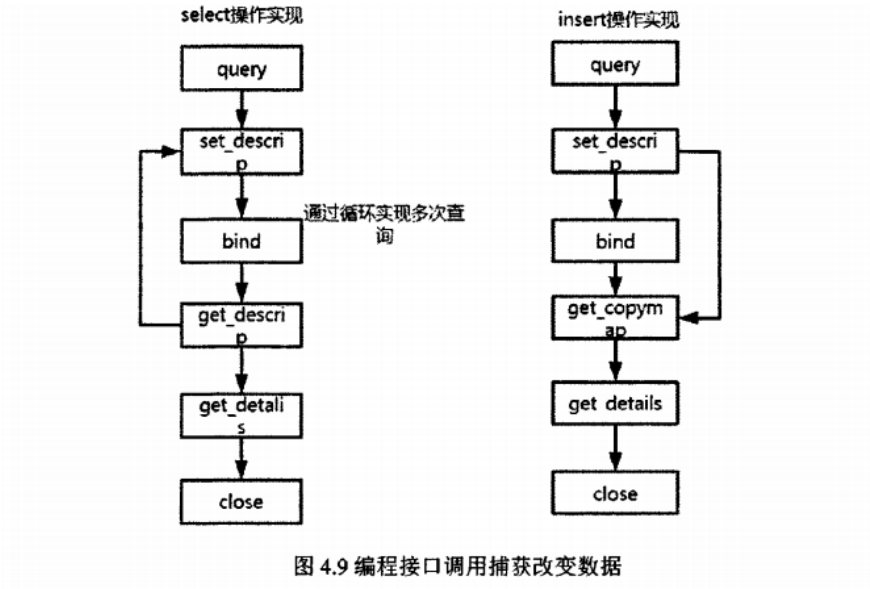

图 4.9 编程接口调用捕获改变数据

1. Distributed Transaction Processing

The distributed transaction processing of Sybase programming interface is implemented by adopting two-phase commit. In the first phase, the query and connection establishment of distributed transaction are performed, and the second part is commit and release. The distributed

transaction first registers the ID of the distributed transaction by calling the regis_ia function, and then queries the transaction by the query function. This function inputs the return value of regis as a parameter, then executes the commit operation, and finally executes the release id to release the transaction.

The specific process is as follows:

(1) In the first stage, the replicator performs initialization work through the init function, then calls the conn function to establish a connection with the server, and then registers the ID of the distributed transaction through the regis id function. However, this is just registration at this time, and the system has not allocated this transaction. The return value information of regis_id is passed in through the query function. The system now formally allocates this new transaction and allocates system space and the initialization information in the init function. At this time, it is in the pre-commit state.

(2) In the second phase, based on the pre-commit status of the first phase, the transaction is formally committed by calling the commit function. After the transaction is completed, the release id is called to release resources

and then the discormet function is called to close the connection of the distributed transaction operation.

The two-stage process is shown in Figure 4.10 below.

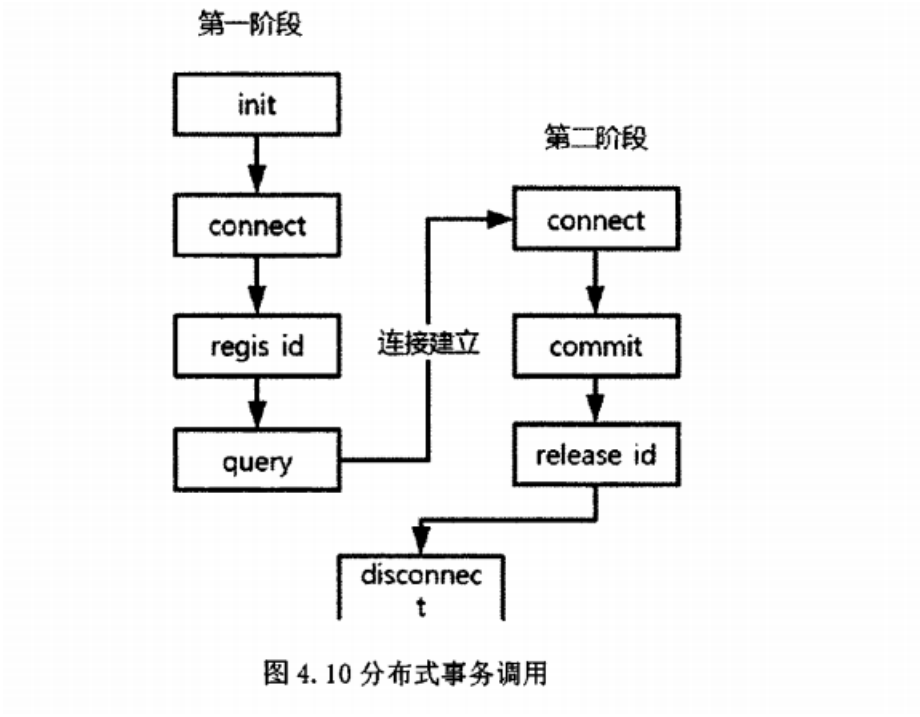

图 4.10 分布式事务调用

2. Processing of large amounts of map vector data

In actual development work, we will encounter a large amount of vector data such as maps. The scale of each map may reach GB level, so it is impossible to transmit through ordinary buffers. When the data is transmitted, additional operations are required to process such data by calling the programming interface. In the above programming interface

call to capture data, the query function is designed for query operations, the insert function is used for insert operations, the set-desc function is used to set the descriptor, the bind function is used to bind the copied data parameter information, and the wait function puts the replicator into a waiting state. If there is such big data in the operation data, it can be processed by disassembling the data into segments, transferring part of the data each time when calling the programming interface to transmit data, and then checking whether there is any untransmitted data each time, and repeating the process in a loop, so that the map-type big data can be processed. The process of processing such big data is shown in Figure 4.11 below.

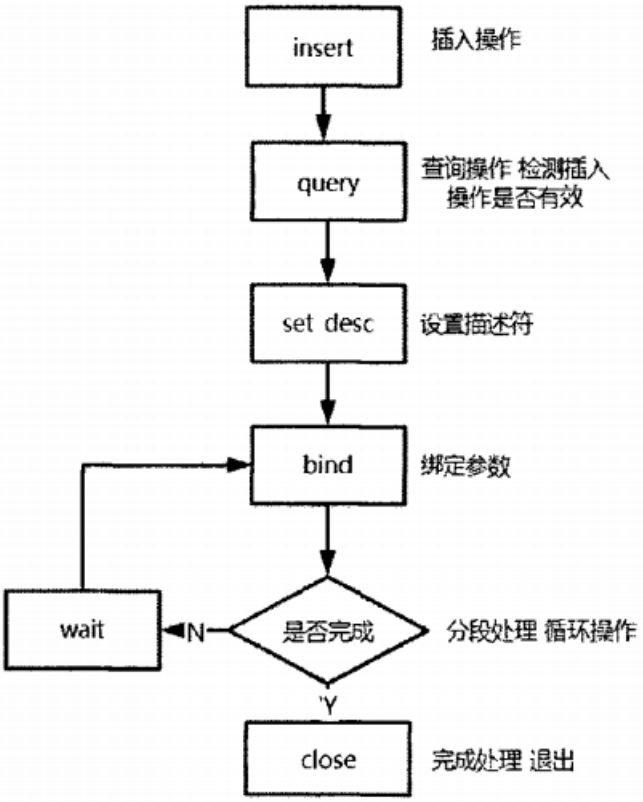

图 4.11 对地图矢量大数据的处理

4.7 Summary of this chapter

This chapter elaborates on the design and implementation of the key modules of the replication system based on the Sybase system. The basic replication mode of the replication system is determined by designing a timestamp-based data replication method. Then the key module data transmission component of the replication system is designed, and the core is the replication server. Then the replication process, replication table and local information table are designed. Then the functions of the

replication system are implemented in the middle layer and the upper layer through the Sybase programming interface call.

5. Performance Evaluation of Data Replication Protocol in Distributed Systems

5.1 Consistency-related indicators

5.1.1 Data consistency

(1) Strong consistency assessment

1. Definition: Strong consistency requires that all replica data remain completely consistent at all times without deviation. It can be measured by comparing the content of the same value on different nodes at the same time. For example, in a distributed database system, within a very short time after the write operation is completed, check whether the corresponding values on each replica node are exactly the same.

2. Evaluation method: You can use a dedicated consistency test tool. After the system writes a new value, this tool will read the value from multiple nodes at the same time and compare it. You can also embed a consistency check module in the system code to trigger the check during

key operations (such as after the write operation is completed). For example, in the evaluation of the distributed database replication protocol of the financial trading system, after each account balance is updated, the balance value is immediately obtained from all replica nodes for comparison, and the number and proportion of inconsistencies are counted.

(2) Final consistency assessment

1. Definition: Eventual consistency allows replica data to be inconsistent for a period of time, but will eventually become consistent. It is necessary to evaluate the time it takes for replica data to become consistent and the degree of impact on system functions during the inconsistency period.

2. Evaluation method: Simulate a high-concurrency write operation scenario, record the time when the values on each node begin to become inconsistent and the time when they finally reach consensus. For business functions that are more sensitive to inconsistencies (such as inventory management in e-commerce systems), observe whether the business logic can run normally during the inconsistency period or the frequency of errors. For example, in a distributed inventory management system, record the time

points when the inventory values of goods become inconsistent on different nodes, and the time when consistency is restored through data synchronization, and count the number of orders that have errors such as overselling during the inconsistency period.

5.2 Availability-related indicators

5.2.1 System Availability

1. Definition: System availability refers to the ability of a system to provide services normally in the face of various failures (such as node failures, network failures, etc.). It can be measured by the ratio of system uptime to total uptime.

2. Evaluation method: Through fault injection testing, simulate node failures (such as shutting down a node) and network failures (such as setting network delays or packet loss) to observe whether the system can continue to provide value reading and writing services. At the same time, record the time it takes for the system to resume normal service from the occurrence of the failure. For example, within a one-month test cycle, count the total time the system cannot provide services due to failures and calculate the availability ratio. Availability = (total

running time - total failure time) / total running time × 100%.

5.3 Performance Overhead Related Indicators

5.3.1 Network bandwidth consumption

1. Definition: During data replication, data needs to be transferred between nodes, which will occupy network bandwidth. Evaluate the network bandwidth usage of the replication protocol under different data update frequencies and data volumes.

2. Evaluation method: Use traffic monitoring tools on network devices to record network traffic during data replication. Compare network bandwidth consumption under different replication strategies (such as full replication and incremental replication) and different data update frequencies. For example, in a distributed storage system, when updating numerical data at different rates, observe the network bandwidth occupied by the full replication strategy and the incremental replication strategy to evaluate which strategy saves more bandwidth resources.

5.3.2 Storage Resource Consumption

1. Definition: Storing multiple copies of a value consumes storage resources, including disk space, memory,

etc. Evaluate the storage space occupied by each node's storage copy and the size of the memory cache value.

2. Evaluation method: Check the storage system information of each node and count the disk space usage for storing value copies. For memory cache, you can use system performance monitoring tools to obtain the memory size occupied by cached values. For example, in a distributed database system, record the growth of disk space used to store value copies on each slave node, as well as the size changes of cached popular values in memory, to evaluate whether the consumption of storage resources is within a reasonable range.

5.3.2 Computational resource consumption

1. Definition: The operation of the data replication protocol consumes the computing resources of the node, such as CPU usage. Evaluate the CPU usage of the node during the data replication process, especially when processing operations such as value updates and consistency checks.

2. Evaluation method: Use system performance monitoring tools, such as top or htop, to observe the node CPU usage for value replication-related operations under different loads (such as different data update frequencies and concurrent read and write request numbers). For

example, in a high-concurrency value write operation scenario, record the changes in node CPU usage when performing replication protocol-related operations (such as logging, data transmission, and consistency algorithm execution).

5.4 Read and Write Performance Indicators

5.4.1 Read Operation Performance

1. Definition: Evaluates the speed and efficiency of reading values from a node, including read latency and throughput. Read latency refers to the time interval from issuing a read request to obtaining a value, and throughput refers to the number of values that can be successfully read per unit time.

2. Evaluation method: Use performance testing tools to simulate a large number of read requests, record the response time of each read request, and calculate the average read latency. At the same time, count the total number of values successfully read within a certain period of time to calculate the read throughput. For example, in a distributed cache system, use tools to simulate 1,000 read requests per second, record the response time of each

request, and count the number of values successfully read per second to evaluate the read operation performance.

5.4.2 Write Operation Performance

1. Definition: Evaluates the speed and efficiency of writing values to the system, including write latency and throughput. Write latency refers to the time interval from issuing a write request to the completion of the write operation on all replicas, and throughput refers to the number of values that can be successfully written per unit time.

2. Evaluation method: simulate a high-concurrency write request scenario, record the response time of each write request, and calculate the average write latency. Count the total number of values successfully written within a certain period of time and calculate the write throughput. For example, in a distributed database system, simulate 500 write requests per second, record the time from the start of each request to the completion of all replica updates, and count the number of values successfully written per second to evaluate the write operation performance.

Chapter 6 Summary and Outlook

With the continuous development of database technology, the enterprise environment is gradually tending towards a decentralized operation mode, and this trend is also driving enterprises to move towards distributed database systems. In order to achieve data sharing in a distributed environment, ensure that users can quickly and efficiently access the database, and ensure the consistency of shared data, the requirements for replication technology are constantly increasing. By investigating the technologies involved in database replication, and conducting demand analysis and design for the replication system, and then conducting system architecture design and core component design and development, the successful construction of the replication system under the distributed system was finally proved through comparative experiments.

6.1 Summary of this work

This paper mainly carries out the following work:

1. In-depth exploration of the overall theory of distributed systems, explaining the basic concepts of distributed systems and distributed transaction management.

2. Introduced data replication technology and common replication forms, and proposed solutions to data conflicts. Introduced replication system planning based on middleware SymmetircDS.

3. The demand analysis and overall planning of the replication system were carried out, and the replication server, the core component of the system, was designed in detail. The overall replication was completed by selecting a timestamp-based data replication solution and then replicating transactions through replication tables. The data replication process and routing management solution were discussed in detail.

4. The methods and ideas for performance evaluation of distributed system data replication protocols are given from multiple aspects, including strong consistency evaluation, eventual consistency evaluation, system availability, network bandwidth consumption, storage resource consumption, computing resource consumption, read operation performance, and write operation performance, providing a valuable reference for related research and practice of distributed system data replication.

6.2 Outlook

The entire replication system is implemented through the design of this article, but further exploration is still needed to optimize it, as follows:

1. This article implements the replication system based on transaction replication, which can detect data changes caused by the three operations of insert, delete, and update. This is only at the DML level and does not provide support for DDL operations. If the replication is based on tables, that is, the data nodes do not have the same table structure, then transaction replication will be difficult to implement and needs to be improved to a snapshot-based implementation or other more efficient implementation methods.

2. Further study the theory of distributed systems and research the replication technology of current database vendors to improve this system and enhance replication efficiency and availability.

References