VENUS: Distributed Message-oriented Middleware towards Heterogeneous Database Synchronization

Bi Li, Rong Yang, Fenghu Dou,Yang Liu,Peng Zhang,Pengxiao Li
National Computer Network Emergency Response and Coordination Center
Institute of Information Engineering ,Chinese Academy of Sciences
National Engineering Laboratory for Information Security Technologies
Email: {libi, yangrong}@iie.ac.cn

Abstract

To solve the problem of data inconsistency in distributed database, this paper designed a distributed message-oriented middleware to synchronize heterogeneous database. In detail, the paper presents a method based on triggers and log tables to get dynamic incremental data. Moreover, the paper presents a message transmission mechanism based on half message and message review to guarantee data consistency. In addition, the paper presents a method of generating full data based on incremental data's real-time computation to achieve high scalability and high availability. Finally, this paper implemented a prototype system named VENUS. The final test shows that VENUS strictly guarantees the data consistency between the destination database and the source one. Besides, the message transmission of VENUS is the most efficient compared to Kafka and RocketMQ.

Keywords—heterogeneous database; distributed middleware; database synchronization; database migration; half message; message review

1 Introduction

With the explosive growth of Internet over the past few years, whether for structured data, semi-structured data or unstructured data, the traditional stand-alone database has become increasingly unable to meet the needs. In addition, stand-alone database has a single point of failure. Once the single server fails, all the service based on it will be stopped. So the distributed database has been favored by all enterprises. The applications based on distributed database are widely used, such as electronic shopping system and banking management systems. These distributed applications usually involve multiple copies of data.

However, due to the decentralization of physical location, different operating systems on the underlying layer and different ISPs on the network layer, the local copy in the distributed environment is like an "islands of information" and always is inconsistent. Especially for the scenarios of financial, electronic and other transactions business, the data consistency between the local databases is particularly important. Taking an example of data inconsistency, the users may have just placed an order, but there is no orders for the shopping cart just a moment, which cannot be tolerated by users. Therefore, this paper studies data consistency.

In the paper, according to the data model of each local database which makes up the distributed database, the local database with the same data model is called isomorphic database. And the database with different data model is called heterogeneous database. Due to the high similarity between homogeneous databases, their synchronization is relatively simple. So the paper mainly researches synchronization of heterogeneous one. Traditional data synchronization [ZHAO et al., 2013] [W et al., 2013] is generally completed based on master-slave synchronization between the databases. In a simple isomorphic database's business scenario there are a lot of synchronization tools and this method is feasible. However, in the high complexity of the business scenarios or with the need of external data synchronization, the traditional synchronization tool is difficult to meet the flexibility and security requirements. Moreover, it cannot used to synchronize heterogeneous databases.

So, this paper designs a distributed message-oriented middleware to synchronize heterogeneous database with some characteristics, such as platform independence and the loosely coupled. But there are several challenges. The middleware must ensure message's order consistency and exactly-once message semantics. On one hand, the order between destination database's received messages and source database's sent messages must be consistent, such as a message generated by adding a record and immediately a message generated by deleting it. On the other hand, the exactly-once must be guaranteed, namely no message loss and repetition. For instance, it is unreliable that a message generated by a deletion operation is lost or a message associated with adding a record is received repeatedly. Any of the above, the data in the source database and destination database will be inconsistent. Therefore, we present several methods to solve these challenges.

For the details, the paper presents a method based on triggers and log tables to get dynamic data after the DML(Data Manipulation Language, referred to as DML in this paper) operations including inserting, updating and deleting on the source database. In addition, as for the fact that most of middleware [S and J, 2014] [C and X, 2013] [S et al., 2013] cannot guarantee exactly-once semantics when network or process failure occurs, the paper presents a message transmission mechanism based on half message and message review. Moreover ,due to the fact that most middlewares[D et al.,2014][I et al.,2015][Y et al.,2015] cannot generate full amount of messages, we present a real-time computation method based on the incremental data to realize heterogeneous data migration. It guarantees high scalability of destination database and high availability of source database service.

In the end, the paper designs and implements a prototype system named VENUS based on the distributed middleware, to achieve database synchronization from Oracle to PostgreSQL. The final test shows that VENUS strictly ensures the data consistency. Besides, the message transmission speed of VENUS is nearly 2* to Kafka and RocketMQ. That is to say, the middleware not only meets the synchronization requirement, but also improves the performance of the message transmission.

The rest of the paper is organized as follows. We briefly overview related work in section 2. We describe our design of middleware in section 3. In section 4, we evaluate the performance of VENUS. We summarize the paper in section 5 and make some discussion about future work in section 6.

2 Related work

This section mainly introduces domestic and foreign situation towards heterogeneous database synchronization, and challenges of using middleware to synchronize database.

There are two main schemes of heterogeneous database synchronization [J et al., 2001] [Z et al., 2014] at home and abroad. One is to use automatic or semi-automatic commercial tools, such as PowerBuilder attached by Pipeline, Data pump of Borland Delphi or EXP/IMP logical backup tool of oracle [F et al., 2013]. But with the tools to realize large amount of data synchronization, the outage time of source database is very long, which seriously impacts on original database business; another is to implement a software or middleware to accomplish database synchronization. The main research previously is based on the software such as index table[T et al.,2013] and trigger-based system[Z et al.,2013].ZHAO Jingling[ZHAO et al.,2013] proposed a capturing strategy based on trigger to implement the synchronization. Waqas[W et al.,2013]presented a generic framework to generate and synchronize ontologies with existing data sources.

The above methods can solve problems more effectively between isomorphic databases. But because the standard model of each component architecture and package data is not the same, heterogeneous database's synchronization by using the various components of the model becomes difficult, even impossible. So, in recent years, some researchers develop the middleware to achieve database synchronization. For example, Emil Vassev[S et al.,2013][G et al.,2014][S et al.,2016] implements a database synchronization system based on Microsoft message queuing service. However, the research based on message queue [F et al.,2015][C et al.,2016][G et al.,2015] is still in its infancy and the middleware which satisfies the distributed database synchronization is less, so then the section focuses on the feasibility of developing Kafka or RocketMQ on behalf of open source message-oriented middleware and Amazon SQS on behalf of commercial one, to synchronize heterogeneous database.

Amazon SQS [S and J,2014][L et al.,2011] is widely used commercial message-oriented middleware, which is simple, safe and does well in the scalability and availability. But SQS cannot guarantee the order of the messages. The order of messages that consumers receive may be not the same as the messages that producers send. SQS only provide at least one semantic and does not guarantee exactly-one message transmission semantics. So, it is possible that consumers receive repeated messages. SQS cannot support real-time computation of incremental messages to generate full data. Based on the above reasons, SQS doesn't fit for high reliability's requirement in the heterogeneous database synchronization scenario.

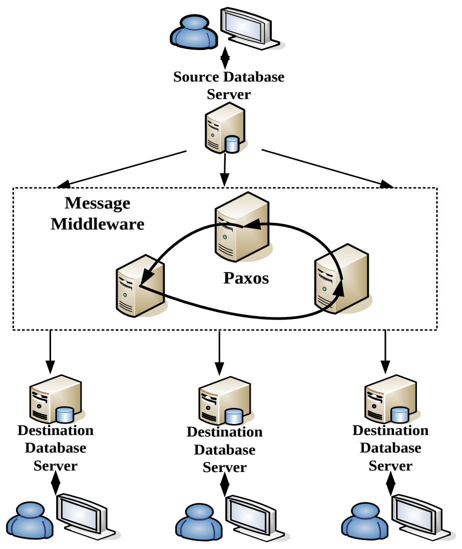
Apache Kafka [Apache,2013] is an open source distributed message-oriented middleware with high throughput. Even deployed on the general equipment, the middleware can also ensure hundreds of thousands of TPS. But like SQS, Kafka cannot guarantee the consistency of the messages. Kafka can only guarantee the order of messages in the partition, other than, between the partitions. The messages that consumers receive between the partitions may be random sequence. Kafka can support at most once and at least one message transmission semantics, but cannot support the exactly-once. Kafka's low-level API supports to subscribe messages from the position where offset equals zero. So, based on it, we can generate a full amount of valid messages. However in the case, all the message must persistent in broker nodes and the nodes can't delete every message from the beginning. Then, a lot of messages pile up on middleware's disk, which will cause the performance degradation of message transmission, and even middleware's denial of service. So Kafka doesn't suit to heterogeneous database synchronization scenario where a consistent data transmission mechanism and a method of generating the full amount of data without affecting other service is needed.

RocketMQ [Apache,2016][S et al.,1996] is a real-time distributed message-oriented middleware which appears by the driving of the Alibaba specific business needs. And at present it has been widely used .But like Kafka in pursuit of high throughput and low delay of the message transmission, RocketMQ abandons the exactly-once of messages. Moreover, RocketMQ cannot get full amount of data. So RocketMQ is not suitable to be developed for heterogeneous database migration and synchronization.

In summary, most commercial and open source middlewares are only appropriate for log application scenarios which don't care about out of order or repetition. And they are not suitable for database synchronization scenarios which has particularly high requirements of data consistency. So, a middleware for heterogeneous database synchronization is imminent.

3 System design

3.1 System Architecture Overview



Source Database Server
Message Middleware
Paxos
Destination Database Server
Destination Database Server
Destination Database Server

Figure 1. System architecture

As shown in Figure 1, this paper designs a distributed middleware to synchronize database. In detail, the system uses the paxos algorithm to synchronize brokers. However, according to the CAP theory, strong consistency, availability, and partition tolerance in a distributed environment cannot be met simultaneously. So, when more than half of the broker nodes are down, the middleware cannot provide services. Distributed systems must make a choice among availability and consistency. Therefore, this middleware provides two strategies, as follows. The choice of this strategy have a crucial impact on the leader's selection, the availability and consistency of the system.

1) High availability-When more than half of the broker nodes are down, choose one of the live brokers to become the leader, to provide services.

2) Strong consistency-When more than half of the broker nodes down, wait for the faulty brokers to restore, or the middleware refuses services.

The distributed middleware decouples the source database and destination databases, so it does not affect the original business of the source one even if the broker nodes of middleware collapse. Besides, with the method proposed in this paper, the middleware can ensure the order and exactly-once semantics in the process of message transmission. Moreover the distributed middleware has highly scalable characteristics. With the growth of business requirements, it can support massive data synchronization by extending horizontally the broker nodes. With the method of generating full data, the architecture realizes database migration and synchronization from the source database to more than one destination database. It can be seen that the paper designs a highly service-based available, highly message-oriented consistent and highly architecture-oriented scalable distributed middleware towards heterogeneous databases synchronization.

3.2 Module Design

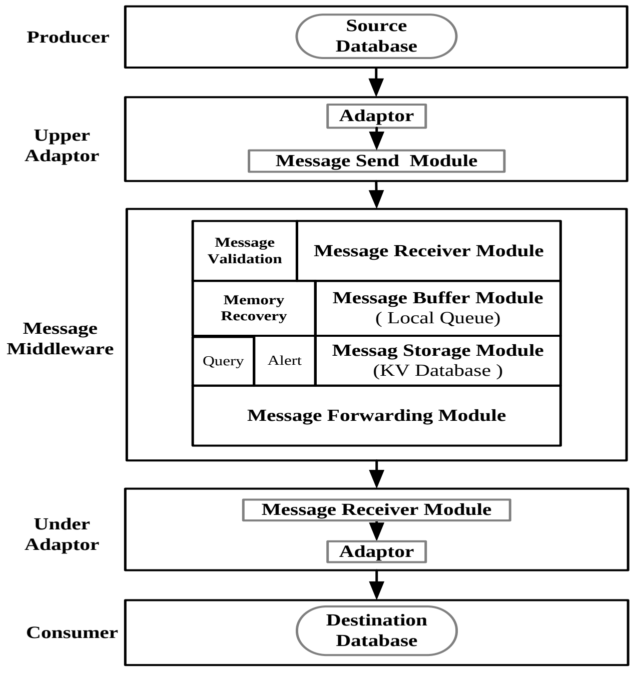


Figure 2. module design

As shown in Figure 2, this section describes the main module design of distributed middleware.

The upper adapter
Adapter in the Upper Adapter realizes the transformation rules of different data models. It plays a role of converting every record in the source database to messages corresponding to the data mode of destination database. And at last the messages will be understood easily by Adapter in the Under Adapter. Message Send Module uses Google Protocol Buffer for message serialization and sends the messages generated by the Adapter to the message-oriented middleware.

Message-oriented middleware
As a message transmission channel, middleware is the core of the heterogeneous database synchronization, both to ensure high consistency and performance of message transmission. Firstly, Message Send Module in the Upper Adapter requests to establish a connection with Message Receiver Module. Once the connection is established,

Message Receiver Module begins to receive messages. Message Validation compares the hash value built in the received message with one obtained by immediately calculating the message. If the two hash value is different, Message Receiver Module will request Message Send Module in the Upper Adapter to resend. If the hash value is the same, messages will be cached to Message Buffer Module. After that, Message Storage Module inserts the messages into the key-value (referred to as KV in the paper) database. Memory Recovery according to the status of the data delivery, uses LRU algorithm to replace the memory, avoiding that the memory is insufficient. Finally, Message Forward Module monitors the specified port and waits for the connection from the Under Adapter.
In the middleware, a business corresponds to a local queue in the Message Buffer Module. Through the whole queue, we guarantee the order of messages. Moreover, middleware support the query of metadata, including the status of messages production and consumption. And middleware offers abnormal alarm to indirectly guarantee system's correct and stable operation.

The Under Adapter
Message Receiver Module in the Under Adapter uses the interface provided by the Middleware to subscribe message from Middleware and Google Protocol Buffer for message deserialization. Then Adapter according to the defined rules executes the corresponding DML operations on destination database to quickly storage records.

3.3 Method Design

A method of capturing incremental data based on triggers and log tables

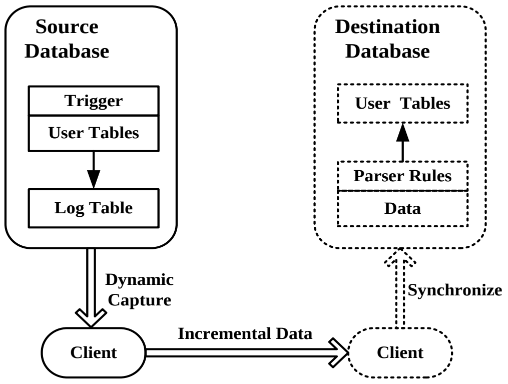


Figure 3. A method of capturing incremental data based on triggers and log tables

Because most of current mainstream databases support triggers to capture users' DML operations, so this paper puts forward a method based on triggers and log tables, which format is shown in Table 1. The process of the method is shown in Figure 3.By defining triggers, the method records control information of each change in the log tables after DML operations.This method includes two stages: table configuration and configuration of triggers. The Table configuration refers to increase some iconic fields in the every table in the source database, such as Sequence, recordID and isDeleted.And then create log tables in the source database. The record format of every table in source database is shown in Table 2. Trigger configuration means to configure triggers for each source table to monitor DML operations. Once changes of records in the source database occur, the event defined by the trigger will execute, namely to update iconic fields in the source table, and then the fields in the log tables.

Table 1. THE FORMAT OF THE LOG TABLE

TableName	*Sequence*	…….

Table 2. THE RECORD FORMAT OF EVERY TABLE

Primary key	*recordID*	*isDeleted*	*Sequence*	……..

The brief description of the idea is as follows. The method defines Sequence–a globally incremental serial number. After each DML operation on a table, the field's value in the table named this will add one to record each change. And with a filed named recordID, it is to mark a unique record. We also increase a filed named isDeleted in all the source tables to express whether the record is deleted or not. When the value of filed named this equals one, it indicated this record is deleted and zero means a real record. The program always records a value named lastSequence, a value of Sequence to take notes the latest already captured changes. According to the value of lastSequence, the program scans a log table, which always records the current largest Sequence of every table, to find the tables where those Sequence's value is greater than lastSequence's value .For this reason, the program scans tables to capture dynamic incremental records in these tables. This method is implemented in the Adaptor module of the Upper Adapter.

A message transmission mechanism based on half message and message review

The system designs a message transmission mechanism based on half message and message review, which is defined as follows, to ensure data consistency, even if a network failure occurs or the process fails.

1) Half message-The producer, namely the source database client, has successfully sent the message to MOM server, but MOM has not yet received the producer's second confirmation of this message.

2) Message review-If the MOM server finds that a message is in a "half message" state for a long time, it will actively ask the producer to inquire whether to send the message to consumer.

As shown in the Figure 4, the flow of this method is described below.

1) The producer sends a message to the MOM server.

2) The MOM replies to the producer whether the message is successfully persisted in the KV database. Then this message is called half message.

3) The producer begins executing the local transaction logic and writes the status to the log stream.

4) The producer submits a second confirmation, such as Commit or Rollback, to MOM based on the results of the local transaction. If the transaction is successfully written, the producer sends Commit state. Then the MOM server receives the status and marks the half message as deliverable. Finally, the consumer, that is, the destination database client, will eventually receive the half message. If the transaction is not written successfully, the producer sends a Rollback status. The MOM then receives the status and removes the half message from the KV database. Finally, the consumer will not receive the message.

5) In the special case of network exception or process exception, if the secondary confirmation submitted in step 4 above does not reach MOM, the MQM will initiate a message review after a fixed period of time.

6) After the producer receives the request of message review,it will check the local transaction of this message.

7) The producer submits the second confirmation again by checking the final status of the local transaction. Then the MOM still operates the half message according to step 4.

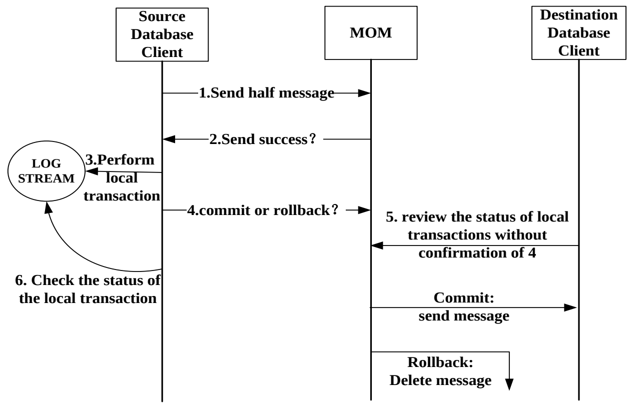

Source Database Client
MOM
Destination Database Client
1.Send half message
2.Send success?
LOG STREAM
3.Perform local transaction
4.commit or rollback?
5. review the status of local transactions without confirmation of 4
6. Check the status of the local transaction
Commit: send message
Rollback: Delete message

Figure 4. A message transmission mechanism based on half message and message review

As mentioned above, this method removes the message through the Rollback state, preventing the producer from sending it repeatedly.And send the message through the Commit status to avoid loss of the message. Thus, by defining a half message and a local transaction, even if there are some exceptions, the data of the source and destination databases are ultimately consistent.

A method of generating full data based on real-time computation of incremental data

With the idea of real-time calculation of incremental data, this paper designs a method of generating full data to persist all the valid data in the middleware. This method is implemented in the message-oriented middleware.the middleware provides two interfaces. One is to get full data and other is to get incremental data. When the Under Adapter request messages with a offset equaling -1, it means that the destination database needs a full amount of data of the source database, which applies to the database migration scenario; When a non-nefative offset, it means that the destination database need incremental data after DML operations and then the Under Adapter continues sending dynamic data , which applies to the scenario of database synchronization.

Table 3. A METHOD OF GENERATING FULL DATA BASED ON REAL-TIME COMPUTATION OF INCREMENTAL DATA

Message status	*KV database status*	*Operation*
isDeleted=1,invalid	nonexistent	Ignore the message
isDeleted=1,invalid	existent	Delete the message from KV database
isDeleted=0,valid	existent	Ignore the message
isDeleted=0,valid	nonexistent	Insert the message into KV database

The brief introduction of the method is as follows.When Message Storage Module retrieves messages from the Message Buffer Module, it according to status of KV database and message to determine whether to storage the message in the KV database or not. As shown in the Table 3, if the isDeleted value of the message equals 1, it indicates the message is generated by the record deletion and the record is invalid now. If the value is 0, it indicates it is a message generated by an operation of adding a record and the record is existent in the source database.The following is to illustrate the process of the method,receiving a message named messgae A for example.

1) If a message with the same recordID of message A is not found in the KV database and isDeleted value built in the message A equals 1, we ignore the message, because this is a repeated message for the fact that this is a message associated with an operation of record deletion, but there is no same recordID message in KV database now. This may be caused by a network error.

2) If found and a value of 1, we delete the message with the same recordID from KV database, because this is a message associated with a deletion.

3) If found and a value of 0, we ignore the message, because this is a repeated message for the fact that this is a message of adding a record, but the message is existent in KV database now. This may be caused by an exception.

4) If not found and a value of 0, we persist the message into KV database for the fact that this is a message of adding a record.

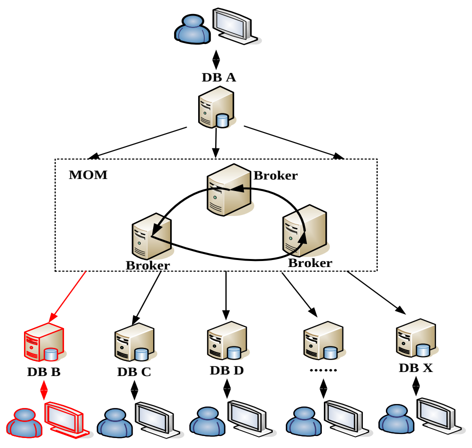


Figure 5. A method of generating full data based on real-time computation of incremental data

In additon,when the database needs to be extended or the database fails, the middleware supports the real-time access to full data. As is shown above in Figure 5, if the destination database B according to the growth of business needs to expand, for example, database B extends to database C and D, database C and D just call the full data interface to achieve high scalability. For this reason, if database A has a single point of failure, it is easy to start a new database to replace database A to achieve high fault tolerance.

4 Performance evaluation

The Oracle database is one of the current mainstream commercial database. However, the operation of Oracle database is complex and the maintenance work is difficult. Nevertheless, PostgreSQL database is an open source database system and is easy to operate and maintain. Therefore,in pursuit of low cost and easy maintenance, this paper implements a synchronization prototype system named VENUS from Oracle to PostgreSQL based on the designed distributed middleware.

The experiment includes incremental data capture test , data transmission test and data consistency test. The data capture test compares the capture speed of inserting, updating and deleting. The data transmission test compares the performance of VENUS with kafka and RocketMQ. The data consistency test checks database consistency when broker fails.

4.1 Test Environment

Table 4. TEST ENVIRONMENT

hardware	***server configuration***
Operating system	Red Hat Enterprise Linux Server release 6.4（Santiago）
CPU	Intel(R) Xeon(R) CPU E5-4620 0 @ 2.20GHz（64 cores）
Memory	DDR3 126GB
Hard disk	SATA 822GB

In the Oracle database and PostgreSQL database, we establish the same structure of a table named student, which includes four fields: ID (number), name (varchar), address (varchar) and sex (varchar). Then we create triggers and log tables on Oracle database. Middleware is deployed on the same stand-alone server and its configuration is as shown in Table 4 . We use 10g version of Oracle and 9.1.1 version of PostgreSQL. In this test, Kafka version is 0.8.2 and RocketMQ version is 3.4.6. The number of Kafka partitions is 1 and replication factor is set to 1, which is the same as RocketMQ. Kafka and RocketMQ all use synchronous replication strategy.

4.2 Incremental data capture test

When we continuously inserts 100,000 of data in Oracle Database, the Upper Adapter records the start time at which the first incremental data are detected and the end time at which the last incremental data are detected. Similarly, when we update 100,000 of data and delete 100,000 data, the time is also recorded in the Table 5 .

Table 5. INCREMENTAL DATA CAPTURE PERFORMANCE TEST

Operation	***start time***	***end time***	***100,000 data capture time(millisecond)***
Insert	351	18512	18161
Update	26952	45036	18084
Delete	59842	78200	18358

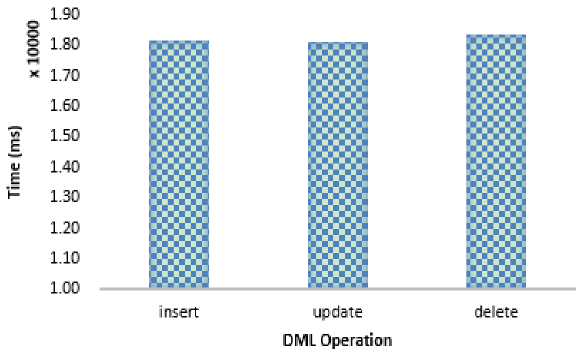


Figure 6. Incremental data capture test

As shown in the Figure 6, the horizontal axis indicates operations of inserting, updating and deleting in turn. The vertical axis represents the capture time of 100,000 operation and the unit is millisecond. Experiments show that the incremental data capture speed of three oprations is almost equal. And the detection of 100,000 incremental data can be completed by about 18s. The experimental results also indirectly demonstrate the high efficiency of soft deletion, instead of really deleting a record from Oracle database. The method reduces the access to database, avoiding the impact of the original database services

4.3 Data transmission test

In this experiment, we records the start time when the first message sends and the end time when the last message is received. The data transmission is calculated by end time minus start time.This experiment develops a producer and consumer based on the API of Kafka and RocketMQ, whose producer multiplexes some code of the Upper Adapter and sends incremental data. The same as VENUS, this experiment records the start time at which the producer sends the first data and the end time at which the consumer receives the last data o. The transmission time of the three middleware is shown in the Figure 7. The horizontal axis indicates Kafka, RocketMQ and VENUS. The vertical axis indicates transmission time of corresponding operations and the unit is millisecond.

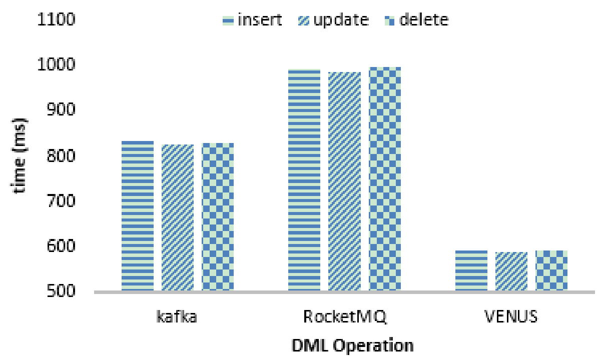


Figure 7. Performance Test

As shown in Figure 7 ,experimental results show:

1. The transmission speed of message corresponding to the inserting, updating and deleting operations has not much difference which is because the number of messages generated by three operations is almost equal.
2. The message transmission of VENUS is the most efficient and the message transmission speed is nearly 2* to that of Kafka and RocketMQ. So VENUS realizes the real-time message transmission.

4.4 Data consistency test

This section tests whether the destination database is consistent with the source database when broker fails. This experiment in turn inserts, updates and deletes 10w records in the Oracle database. Simulate the broker's failure by pulling out the host power, and then restart Broker to view the database's status.

Experiments show that kafka and RocketMQ lost some messages, because kafka and RocketMQ are set to asynchronous brush strategy. VENUS doesn't lose messages, because VENUS's Data Dtorage Module uses synchronous brush strategy. Kafka and RocketMQ receive duplicate messages, while VENUS will not receive duplicate ones, because VENUS based on the primary key named Sequence in the KV database ensures that the message is idempotent. The experimental results show that the student in the destination database is exactly the same as the student table in the source one. Therefore, in the case of failure of the broker server, VENUS guarantees the data consistency.

5 Conclusion

On conclusion, in pursuit of the data consistency of distributed database, this paper implements a synchronization prototype system from Oracle to PostgreSQL. And this paper puts forward a method based on triggers and log tables to get incremental data. Moreover, the paper introduces half message and message review to guarantee data consistency. In addition, the paper presents a method of generating full data to realize the expansion and fault tolerance. Finally, the test results prove that the middleware in the paper improves the performance of message transmission based on the assurance of data consistency.

6 Future work

In this paper, the distributed middleware for heterogeneous database synchronization only realizes the synchronization from relational database to relational one. With the application development of social networking services, more and more developers have tried to abandon traditional relational database, and use the relatively stable and reliable NoSQL database. Therefore it makes sense to achieve a tool to assist developers accomplishing migration and synchronization from the relational database to non-relational one. So the future work in this paper include designing a distributed middleware model to synchronize data from a relational database to non-relational one. And based on this model, we will develop a prototype system to realize data migration and synchronization from Oracle database to HBase database.

Acknowledgments

The research work is supported by the National Key R&D Program with No.2016YFB0800302, Key Research and Development Plan with No.2016YFB081304 , National Natural Science Foundation of China (No. 61402464), Youth Foundation of National Computer Network Emergency Response Technical Coordination Center (No.2016QN-19.)

References

[ZHAO et al.,2013] ZHAO Jinling, TAN Xianhai, WANG Yalan, HE Lei.Implementation of change capture and Dynamic Synchronization System of distributed heterogeneous database based on XML [J]. RAILWAY COMPUTER APPLICATION, 2013 (10): 37-40.

[W et al.,2013] Ahmed W, Aslam M A, Lopez-Lorca A A, et al. Using ontologies to synchronize change in relational database systems[J]. Journal of Research and Practice in Information Technology, 2011, 43(2): 89.

[Apache,2013]Kafka[online]2013,https://kafka.apache.org/

[Alibaba,2016]RocketMQ[online]2016,https://www.oschina.net/p/RocketMQ

[S and J,2014]Mathew S, Varia J. Overview of amazon web services [J]. Amazon Whitepapers, 2014.

[C and X,2013] Zhang C, Liu X. HBaseMQ: A distributed message queuing system on clouds with HBase[C]//INFOCOM, 2013 Proceedings IEEE. IEEE, 2013: 40-44.

[S et al.,2013]Hernández S, Fabra J, Álvarez P, et al. A reliable and scalable service bus based on Amazon SQS[C]//European Conference on Service-Oriented and Cloud Computing. Springer Berlin HeIDelberg, 2013: 196-211.

[D et al.,2014] Patel D, Khasib F, Sadooghi I, et al. Towards in-order and exactly-once transmission using hierarchical distributed message queues[C]//Cluster, Cloud and GrID Computing (CCGrID), 2014 14th IEEE/ACM International Symposium on. IEEE, 2014: 883-892.

[I et al.,2015] Sadooghi I, Wang K, Patel D, et al. Fabriq: Leveraging distributed hash tables towards distributed publish-subscribe message queues[C]//Big Data Computing (BDC), 2015 IEEE/ACM 2nd International Symposium on. IEEE, 2015: 11-20.

[Y et al.,2015] Liang Y, Tang X, Bing L, et al. Study on Service Oriented Real-Time message-oriented middleware[C]//Semantics, Knowledge and Grids (SKG), 2015 11th International Conference on. IEEE, 2015: 207-211.

[J et al.,2001] Lehman T J, Cozzi A, Xiong Y, et al. Hitting the distributed computing sweet spot with TSpaces[J]. Computer Networks, 2001, 35(4): 457-472.

[Z et al.,2014] Cai Z, Ji S, He J, et al. Distributed and asynchronous data collection in cognitive radio networks with fairness consideration [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(8): 2020-2029.

[F et al.,2013] Wang J F, Shi Z F, Qu Z, et al. Research and Implementation of Oracle Database Synchronization Timing[C]//Advanced Materials Research. Trans Tech Publications, 2014, 989: 4917-4919.

[T et al.,2013] Karnagel T, Dementiev R, Rajwar R, et al. Improving in-memory database index performance with Intel® Transactional Synchronization Extensions[C]//High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on. IEEE, 2014: 476-487.

[Z et al.,2013] Zhang Z, Lu C. Research and implementation for data synchronization of heterogeneous databases[C]//Computer and Communication Technologies in Agriculture Engineering (CCTAE), 2010 International Conference On. IEEE, 2010, 3: 464-466.

[S et al.,2013] Ji S, Cai Z. Distributed data collection in large-scale asynchronous wireless sensor networks under the generalized physical interference model [J]. IEEE/ACM Transactions on Networking, 2013, 21(4): 1270-1283.

[F et al.,2015] Ringeval F, Eyben F, Kroupi E, et al. Prediction of asynchronous dimensional emotion ratings from audiovisual and physiological data[J]. Pattern Recognition Letters, 2015, 66: 22-30.

[C et al.,2016] Kim B C, Jang C. A Distributed Instant Message System Architecture using Media Control Channel [J]. Journal of the Korea Institute of Information and Communication Engineering, 2016, 20(5): 979-985.

[G et al.,2015] Wang G, Koshy J, Subramanian S, et al. Building a replicated logging system with Apache Kafka [J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1654-1655.

[Y et al.,2015] Lu Z Y, Guo Z B, Du X F, et al. A Method of Data Synchronization Based on Message Oriented MIDdleware and Xml in Distributed Heterogeneous Environments[C]//Proceedings of International Conf. on Artificial Intelligence and Industrial Eng. no. Aiie. 2015: 210-212.

[S et al.,1996] Passint R S, Oberlin S M, Fromm E C. Messaging facility with hardware tail pointer and software implemented head pointer message queue for distributed memory massively parallel processing system: U.S. Patent 5,581,705[P]. 1996-12-3.

[L et al.,2011] Tran N L, Skhiri S, Zim E. Eqs: An elastic and scalable message queue for the cloud[C]//Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on. IEEE, 2011: 391-398....

[G et al.,2014] Taboada G L, Dominguez J T, Biempica R D. Method and middleware for efficient messaging on clusters of multi-core processors: U.S. Patent 8,839,267[P]. 2014-9-16.

[S et al.,2016] Tarre M S, Rantzau R, Dutta D, et al. Multi-datacenter message queue: U.S. Patent Application 15/154,141[P]. 2016-5-13