Xiai YAN[1,2], Jinmin YANG[2], Qiang FAN[1]

Hunan Police Academy, China(1), Hunan University, China(2)

An Improved Two-phase Commit Protocol Adapted to the Distributed Real-time Transactions

Abstract: *Based on the existing two-phase commit protocol (2PC), this paper proposes a protocol adapted to the distributed real-time transaction commit, which can avoid the blocking problem when dealing with transactions by coordinator redundancy; additionally, by the "health loan" of the locked data, the concurrency of transaction implementation is expanded to save the waiting time. Experimental analysis shows that: when the average arrival interval time of transaction is small, the success rate of the improved commit protocol is significantly higher than that of 2PC.*

Streszczenie. *W artykule zaproponowano protokół bazujący na dwufazowym protokole typu 2PC. Nowy protokół pozwala na oszczędność czasu oczekiwania – jeśli średni czas odstępu między transakcjami jest mały przepływność danych jest wyższa niż w klasycznym protokole 2PC. (**Ulepszony dwufazowy protokół do zarządzania transakcjami w czasie rzeczywistym)***

Keywords: Distributed Real-time Transaction, Commit; Redundancy, Data Loan.**Słowa kluczowe:** transakcje w czasie rzeczywistym, nadmiarowość, przepływ danych.

1 Introduction

The distributed database system is the one which is physically distributed but logically centralized. In order to maintain the atomicity of distributed transactions, it requires a commit protocol. Two-phase commit protocol (2PC), as its simple, practical and low overhead features, becomes one of the most common-used commit protocol.

Typical characteristic of real-time transactions is the time constraint and all the processes have deadlines[1]. Therefore, a distributed real-time transaction commit is supported to meet the requirements of both the atomicity and the time constraints. However, due to too much waiting, the existing 2PC may cause commit blocking, which cannot satisfy the requirement of distributed real-time transactions commit.

In distributed system, redundancy is the technology used to enhance the usability. For the commit requirement of the distributed real-time transactions, in the environment of coordinator redundancy and by drawing on the idea of PROMPT[2,3] protocol, this paper presents a commit protocol which can avoid the commit blocking as well as advance the transaction implementation---2PC based on Redundancy and Loan (RL2PC).

2 Basic 2PC protocol

2.1 The commit process of 2PC protocol

Gray (1978) proposed a 2-phase commit protocol [4] (2PC), in which commit process is as shown in Figure 1.

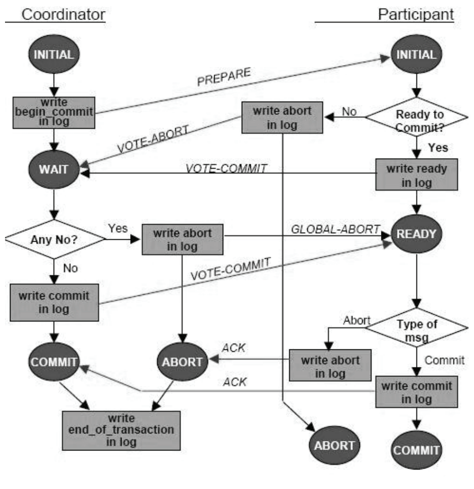
First phase: coordinator add the record <begin_commit> in the log, send the messages of <Prepare> to all participants and start Timer to step into the waiting stage; participant receive the <Prepare> news, if it is willing to commit its own part, it can send the message of <Ready>to coordinator; if it is not willing to commit it due to some reasons, it can send the message of <Abort> to coordinator, and add the message to the log..

Second phase: If all participants answer <Ready>, coordinator send <Global_Commit> to all of them, otherwise, send the command of<Global_Abort>; if time is out, it also send the command of <Global_Abort> to participants, and add the command to the log. Participants commit or undo the transactions according to the command of coordinator, and send the message of <Acknowledge>to coordinator and add the message to the log. Coordinator collect the message of <Acknowledge> from all participants, add the message to the log and terminate the transaction.

2.2 2PC protocol analysis

2PC can complete the transaction commit through two phases. It is simple and compact, which can expand the effect of the local atomic commit behavior to the distributed transaction, not only ensuring the atomicity of the distributed transaction commit, but also achieving the rapid failure recovery without damaging the log.

There are two typical defects in 2PC: ① When the participants are in the state of <ready T>, if the coordinator go wrong, coupled with at least one participant, then it is impossible to end it because the workable participants cannot know whether participants decide to commit; selecting a new coordinator to direct the action may produce an inconsistent state. At this time, participants must wait for the recovery of coordinator, if it cannot be restored within a short time, the transaction blocking will emerge. ② So much "waiting" exists, especially when the sub-transaction is in the <ready T> state, other transactions cannot access to the data items occupied by this sub-transaction but wait for the completion of the transaction commit.



Fig,1. 2PC protocol Actions

In order to improve the performance of 2PC protocol, Mohan et al proposed "the assuming reversed 2PC protocol "and “the assuming committed 2PC protocol "[5], which can reduce the volume of transmission messages as well as the log writes between coordinator and participant, but fail to solve the problem of transaction obstruction. In order to solve it, Skeen (1981) proposed a three-phase commit protocol(3PC)[6], which can meet the non-blocking conditions of transaction commit by adding an additional pre-commit state, however, due to its large storage and communication overhead, it is rarely used in practice.

3 The committing mechanism of RL2PC protocol

3.1 Model of RL2PC protocol

The obstruction of 2PC protocol is due to the fault of coordinator. In order to prevent the transaction obstruction and based on the 2PC protocol, RL2PC protocol adds a redundant node of a coordinator to act as the sub-coordinator, making the participants believe the coordinator is normal. Its model is as shown in Figure 2.

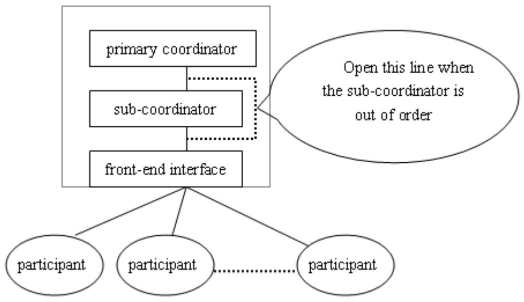


Fig.2. Model of RL2PC protocol

3.2 The operating mechanism of RL2PC under normal circumstances

When the RL2PC operates normally, participants and sub-coordinator conduct message exchange. Sub-coordinator owns the autonomy, so it plays the role of fault-free transaction coordinator and real-timely transmits the generated logs to the primary coordinator. The primary coordinator owns the same responsibility with the sub-coordinator, but doesn't direct actions. It is mainly responsible for receiving a log to rebuild a database with the same version of sub-coordinator. The synchronous data forwarding between sub-coordinator and primary coordinator is transparent for the participants, that is, for the participants, they are the consistent with the original 2PC, only knowing one coordinator.

3.3 The operation mechanism of RL2PC when the coordinator is out-of-order

In network communication, after the synchronous forwarding of logs between the primary coordinator and sub-coordinator, a connection should be conducted. Assuming there is no network partitioning and packet loss, if the connection is unsuccessful, the opposite party can be determined to have something gone wrong.

When the sub-coordinator is out of order, the interface mechanism can be used to directly respond to the primary coordinator who will then take over the transaction, so that the participants cannot feel the service outage. When the sub-coordinator recovers, the primary coordinator provides a proper database for the sub-coordinator, which can reflect the transaction update when the sub-coordinator cannot work, thus enabling the sub-coordinator to restart normal work after recovery.

When the primary coordinator fails, due to the autonomy of sub-coordinator itself, the sub-coordinator can also achieve the rest work. After the primary coordinator recovers, the cub-coordinator provides an accurate database for the primary coordinator, and then continues to work.

In this mechanism, the participants may regard the coordinator as the one who will never fail, so the obstruction caused by the commit will never emerge any more.

4 The method to solve the conflict between transaction commit and transaction implement in RL2PC protocol

4.1 Conflict of the distributed real-time transaction

In the management of distributed transactions, due to the concurrent operation of multiple transactions run, the system must control the interaction between concurrent transactions. The technology of locking is the most common-used concurrent control, which enables the global transactions to be implemented serially according to a certain order by locking data items. In the read / write mode, the lock is generally divided into 2 types, that is, shared lock (shared-S) and exclusive locks (exclusive_X)[7]. The compatibility relation between locks is as shown in Table 1, Y means compatible, while N means incompatible.

Table 1. the compatibility relation between locks

	shared-S	*exclusive_ X*
shared-S	Y	Y
exclusive_ X	N	N

Table 1 indicates that the transactions can be divided into read-write conflict, write-read conflict and write-write conflict.

For a real-time transaction, an important factor is to solve the conflicts between transactions. The life cycle of transaction can be divided into two stages, namely the implementation phase and the commit phase [8]. Conflict may occur between the two transactions in the implementation phase, or between a transaction in the implementation phase and another transaction in the commit phase. Currently, the former conflict can be solved by multi-versioning technique, but a good solution to the latter one has not been found.

In the existing protocols, data in the ready state are all inaccessible. In other words, the commit time of a transaction will largely affect the implementing time of another transaction, thus affecting the system's ability to handle transaction deadline.

4.2 Resolution of the transactions access conflict in RL2PC protocol

Assume that the concurrent control algorithm of transaction is based on the lock. The core idea of RL2PC protocol to resolve the transactions access conflict is: on condition that ensuring the accuracy and consistency of transactions, the data locked by the sub-transaction of implementing transactions which is in the ready state of commit, in other words, the sub-transactions in the commit phase can loan data to the concurrent implementing transaction (on the condition that the lock has not been released). The relationship between commit transaction and implementing transaction is like that between the "lender" and "borrower".

A commit transaction close to the deadline may fail due to missing it before the commit, and the loan of this kind of transaction may lead to failure of series. Consequently, the protocol provides that only health transactions can lend their data which is locked and in the ready state.

The protocol is defined as follows:

Definition 1: If the transaction T_2 must be committed after the commit of transaction set T_1, the commit of transaction T_2 depends on the transaction set T_1, that is, $commit(T_2) \Rightarrow commit(T_1)$, denoted by $T_2 CDT_1$.

Definition 2: The set where all commits depend on the transaction T is called as the commit dependence set of T, denoted by $CDS(T)$.

Definition 3: If the abort of transaction T_2 depends on the transaction T_1, that is, $abort(T_2) \Rightarrow abort(T_1)$ denoted by T_2ADT_1.

Definition 4: The set where all aborts depend on the transaction T is called the abort dependence set of T, denoted by $ADS(T)$.

Definition 5: enable F to be the health factor of the transaction T , $F = timeleft / \min time$, where $timeleft$ is the remaining time from the current time to the deadline; $\min time$ is minimum time that a transaction commit is completed .

Definition 6: Set $\min F$ as the threshold of data processed by the access allowed transaction in the commit state in the system; if for a transaction, $F_T \geq \min F$, then determine the transaction is healthy.

When data access conflict emerges, the solution is shown as follows:

① read-write conflict. The implementing transaction T_2 applies to add write-lock on the data item, but the transaction T_1 ready to commit has done it. At this time, T_2CDT_1 , $CDS(T_1) = CDS(T_1) \bigcup \{T_2\}$, T_2 obtains the write-lock.

② write-write conflict. The implementing transaction T_2 applies to add write-lock on the data item, but the transaction T_1 ready to commit has done it. If $F_{T1} \geq \min F$, $ADS(T_1) = ADS(T_1) \bigcup \{T_2\}$, T_2 obtains the write-lock, or it will wait.

③ write-read conflict. The implementing transaction T_2 applies to add read-lock on the data item, but the transaction T_1 ready to commit has done it. If $F_{T1} \geq \min F$ and $ADS(T_1) = ADS(T_1) \bigcup \{T_2\}$, T_2 obtains read-lock, or it will wait.

4.3 The dependence relationship between the implementing transaction and commit transaction.

When the implementing transaction T_2 accesses to the data shared by the transaction T_1 which is in the ready state, the following three conditions may emerge in the commit process:

① the implementing transaction T_1 receives the global decision before the data of transaction T_2 being processed. If the transaction T_1 receives the command of global commit, transaction will be processed according to the normal schedule, and $ADS(T_1)$ will be deleted; if the transaction T_1 received the command of global abort, all the transaction in $ADS(T_1)$ will be aborted, and then $ADS(T_1)$ will be deleted.

② Transaction T_1 and T_2 obtain the global command after being processed. The implementing transaction T_2 must wait until T_1 receives the global command or it passes the deadline. If the former case occurs first, the first principle will be used after T_1 waits its command; if the latter case occurs first, T_2 aborts and will be deleted from $ADS(T_1)$ and $CDS(T_1)$.

③ Transaction T_2 aborts before T_1 receives the global command. At this time, all the updates of T_2 will be recalled, while T_2 will be removed from the independence set $ADS(T_1)$.

5. Experimental analysis

5.1 Experimental Environment

Utilize VC6.0 programming to achieve the RL2PC protocol and operate the simulation software SIM + + on the platform with the CPU of P4 3.2G, memory of 1G and the operating system of Redhat Linux 9.0. Assume that the database is composed by a number of pages, each page contains a fixed number of records and the records processed by transactions are all evenly distributed. The parameters used in the experiments are as shown in Table 2.

Table 2. Simulated test parameters

parameters	**numerical value**
Database nodes:	10
Database pages:	500
Computation time of CPU pages:	10ms
The access time of disk pages:	20ms
The number of data items of per page:	6
The write probability of transactions:	0.3
The number of page scope of per transaction	[10,30]
The average execution time of write transactions	340ms

In the experiment, two indicators--- the transaction success rate (TSR) and delayed deadline rate (DDR) are utilized to conduct evaluation.

$$TSR = \frac{TS}{TA}; \quad DDR = \frac{DDT}{TA}$$

In the formula, TS is the number of transactions which have been successfully committed, DDT is the number of transactions whose deadline has been delayed, TA is the number of transactions which have been received by the system.

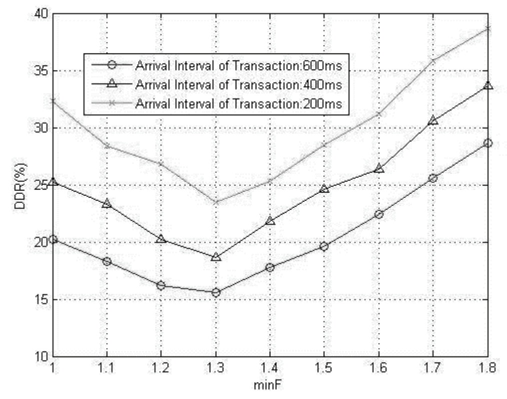


Fig.3. The DDR value with different health factors

5.2 The selection of health factors

Only health transactions can loan data, therefore, the selection of health factors will directly affect the performance of RL2PC protocol. When the health factor is slightly smaller, it is easier to loan the locked data, but due to the the transaction is close to the deadline, so it may lead the series abort; while when the health factor is larger, it is difficult for to loan the locked data, which will reduce the performance of RL2PC protocol. The previous analysis indicates that when the health factor $\min F \to \infty$, the commit process of the RL2PC protocol has become a standard 2PC.

Different health factors are selected to test the DDR of transaction and the test results are as shown in Figure 3. When the averagerrival interval of transactions is respectively 200ms, 400ms and 600ms, the DDR reaches its minimum value when $\min F = 1.3$, indicating that $\min F = 1.3$ is the optimal health factor under the current test environment.

5.3 Comparative analysis on the performance of RL2PC and 2PC

On the condition that the average arrival interval of transaction is different, commit protocol of RL2PC and 2PC are respectively adopted to conduct transaction commit, where the health factor of RL2PC is $\min F = 1.3$. The results of comparative analysis on the transaction success rate are as shown in Figure 4.

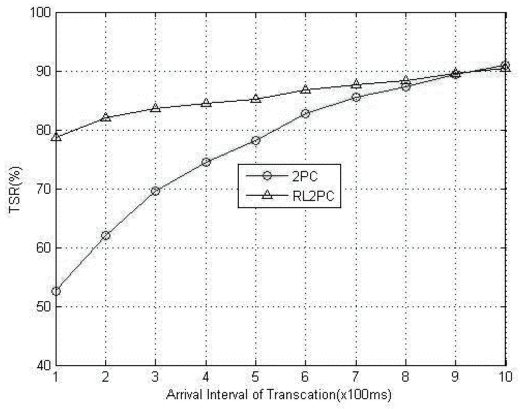


Fig.4. Comparison of transaction success rate between RL2PC and 2PC

The results of Figure 4 show that when the average arrival interval of transactions is smaller, that is, when transaction processing is a little intensive, the transaction success rate of RL2PC is significantly higher than that of 2PC. This is due to the fact that intensive transaction may lead to great conflict between transactions, and RL2PC, on one hand, ensures the reliability of coordinators by the coordinators' redundancy, thus reducing the communication overhead of commit and avoiding the blocking problem, on the other hand, healthily loans the locked data to increase the concurrency of transaction implement and save the delay time, thereby increasing the success rate of the transaction. When the average arrival interval of transactions increases, the conflict between transactions becomes weaker and the advantageous performance of RL2PC will be weakened due to the increase in fault-tolerant overhead, however, its performance is still approximately equal to that of 2PC.

6 Conclusions

The timeliness of real-time transactions commit must be taken into account. In this paper, based on the existing two-phase commit protocol, a modified protocol adapted to the distributed real-time transaction commit is proposed, which has been improved from two aspects: on the one hand, by the coordinator redundancy, it ensures the reliability of coordinator and avoids the blocking caused by the commit; on the other hand, when the sub-transaction commit is in the ready state, it can healthily loan the locked data, which can advance the implement time of other transaction and expand the concurrence of transaction implement.

Acknowledgment

This paper is supported by the Application Innovation Project of Ministry of Public Security of the People's Republic of China (2011YYCXHNST072), and Scientific Research Project of Hunan Province scientific and technical Department.

REFERENCES

[1] J.R.Haritsa , M..Livny and M.J.Carey Earliest Deadline Seheduling for Real-Time Database Systems. Proeeedings of the IEEE Real-Time Systems SymPosium, 1991, PP:232-242
[2] J Haritsa, K Ramamritham, R Gupta. The PROMPT real-time commit protoco1. IEEE Trans on Parallel and Distributed Systems，2000，11(2)：160-181
[3] K Lam, C Pang, S H Son. Resolving executing-committing conflicts in distributed real-time database systems . The Computer Journal，1999，42(8)：674- 692
[4] J Gray. Notes on Data Base Operating Systems, in R Bayer, M Graham, G seegmuller(eds). Operating Systems: An Advance Course, New York: Springer-Verlag,1979,P393-481
[5] C Mohan, B Lindsay, R Obermarck. Transaction Management in the R+ Distributed Database Management System. ACM Transactions on Database Systems, 1986, 11(4)：378-396
[6] D Skeen. Nonblocking Commit Protocols, Proc . ACM SIGMOD Int. Conf. on Management of Data，1981：133-142
[7] K Ramamritham, C Pu. A Formal Characterization of Epsilon Serializability. IEEE Transactions on Knowledge and Data Engineering. 1995, 7(6)：997～1007
[8] P A Bernstein, N Goodman. Concurrency Control in Distributed Database Systems.ACM Comput. 1981,13(2):185-222
[9] Andrew S.Tanenbaum. Computer Nerworks(Fourth Edition)[M]. Published by Tsinghua Unversity Press.2004

***Authors**:Xiai Yan, Associate Prof, Hunan Police Academy, Department of Information and Technology, E-mail:yanxiai222@yahoo.com.cn*
Jinmin Yang, Prof, Software College of Hunan University, E-mail: rj_jmyang@hnu.cn
Qiang Fan, Associate Prof, Hunan Police Academy, Department of Information and Technology, E-mail: 695515549@QQ.com