Concurrent Control and Performance Optimization of Distributed Transaction in Heterogeneous Network Environment

Guojian Cheng[1, a], Liting Liu[1, b], Xinjian Qiang[1, c *] and Ye Liu[1, d]

[1]School of Computer Science, Xi'an Shiyou University, Xi'an, 710065, P.R. China

[a]gjcheng@xsyu.edu.cn, [b]liulitingDL@163.com, [c]xasyu@126.com, [d]yeliu@xsyu.edu.cn

*The corresponding author

Keywords: Distributed transaction; Concurrency control; Granularity control; Performance optimization; Heterogeneous network environment

Abstract. At present, most of the transaction operations are required in the heterogeneous network environment. The transaction operations not only guarantee transaction implemented correctly and effectively, but also keep transactions under the different environment of consistency and integrity. This paper mainly introduces Lock schemes for resolving the concurrent access conflicts in distributed transaction and the control methods of lock granularities applied for different cases are proposed. It provides solutions to concurrent transaction congestion in the network environment, while optimizing the performance of distributed transaction.

Introduction

A transaction may need access to data distributed among several databases governed by a single organization. This gives to rise to the concept of a distributed transaction, that is, a transaction that contains actions on several intra organization databases connected to a computer network. With the rapid development of the traditional database technology, network technology and digital communication technology, the data distributed storage and distributed processing are the main feature of the distributed database system began to receive people's attention. In a distributed database system, distributed transaction management is an important and complex process. Transactions are a serial of atomic operations that have the character of ACID [1] (Atomicity, Consistency, Isolation, Durability), We use the term distributed transaction to refer to a transaction that accesses objects managed by multiple servers [2].

The goal of transaction management must achieve the correctness of data, the concurrency of execution and the efficiency of the task completion. Therefore, the transaction is distributed, a global transaction to decompose it into several sub transactions to accomplish. Each sub transaction corresponds to a particular site, therefore the distributed transaction management also considers distributed execution, operations, communications, and other issues. Sharing network sources in particular information sources of heterogeneous network environment (such as databases or files) would appear concurrency transactions. In order to guarantee consistency of transaction in accessing concurrently objects, the execution of transactions must be scheduled so that their effect on shared data is serially equivalent.

A server can achieve serial equivalence between transactions by serializing access to the objects. However it may affect the performance of servers. So how to acquire maximal parallelism is the issue that must be emphatically resolved in distributed transactions.

Concurrency Control of Transactions

Distributed concurrency control provides concepts and technologies to synchronize distributed transactions in a way that their interleaved execution does not violate the ACID properties[3]. Distributed concurrency control makes sure that all sub-transactions of a set of distributed transactions are serialized identically in all data servers involved. Therefore, not only local

© 2016. The authors - Published by Atlantis Press

dependencies need to be taken into account, but also dependencies involving multiple data servers[4]. An executing transaction is divided into two phases: growing phase and shrinking phase. First phase, the transaction continues to acquire new locks. Second phase it releases them. This locking mechanism is called "two-phase locking"[5]. In a distributed transaction, locking scheme is widely method to solve multiple transactions that attempt to access the same object generated conflicts. When a transaction set an application for an object, a server first locks the object and then the transaction can access exclusively the object. In fact, when a client is doing read on file, it does not mind other readers read same file, only hopes clients had better not change it during reading. The improvement is achieved by giving two locks that are called shared lock and mutually exclusive lock or read and write lock. Lock compatibility see Table 1.

Table 1 Lock compatibility

Lock already set for one object	New lock requested	
	Read	Write
None	OK	OK
Read	OK	Wait
Write	Wait	Wait

How many a unit of locking is called lock granularity. The more suitable the granularity is setting, the more precise the object is locked, and system can acquire greater concurrency. If the granularity is over big, long waiting occurs when multiple transactions have their requests for the locked object. On the other hand, the more small the lock is divided, the more complex the system is managed and it may tend to deadlock.

Web Service Transaction Model

The transaction model bases on the "Web Services Transactions Specification" [6].However, due to our focus on the blocking problem when guaranteeing atomicity, we can use a much simpler transaction model in this paper, i.e., we do not need a certain Web service modelling or composition language like BPEL4WS [7].Web services are invoked by asynchronous messages instead of invoking them by synchronous. Fig. 1 illustrates the situation that we want to avoid: The Web service T_i must wait for the return value of T_j before it can finish its execution. In contrast, we want each sub-transaction to be able to autonomously complete its read phase. Thus, we allow sub-transactions only to return values indirectly by asynchronously invoking corresponding receiving Web services, but not synchronously by return statements. The model also supports a synchronous Web service. The transaction T_i calling a sub-transaction T_j and waiting for T_j 's return value to continue, as shown in Fig. 1,is replaced in our model with three transactions as shown in Fig. 2.

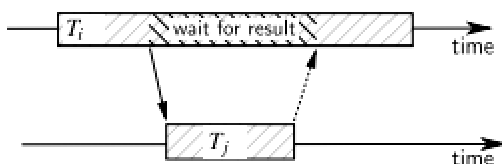

Figure 1. Synchronous calls for Web services

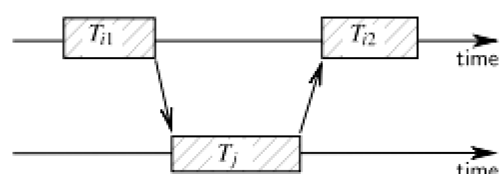

Figure 2. Modeling synchronous Web service calls by asynchronous invocations

Here, T_{i2} is a sub-transaction of T_j. T_j is a sub-transaction of T_{i1}.

The model differs from other models that use nested transactions. In some aspects including following: No sub-transaction is allowed to commit independently to the others or before the Commit

coordinator guarantees that all sub-transactions can be committed. Different from CORBA OTS, we assume that we cannot identify a hierarchy of committing decisions, where aborted sub-transactions can be compensated for executing other sub-transactions. Different from the Web service transaction model, the Initiator of a transaction in our model does not need to know all the transaction's sub-transactions in advance. A Web service may consist of control structures; Communication is message oriented.

Reduce Transaction Blocking Solution

Adjourn State. The Adjourn state avoids setting up participant timeouts for aborting a transaction by distinguishing between two states in which a database can wait for the coordinator's vote Request message: the blocking state and the non-blocking Adjourn state. The Adjourn state shows the following advantages: It does not require the setup of a transaction time-out; It does not block concurrent transactions; It provides a flexible reaction to concurrency failures of distinguishing failures that require a transaction aborts, failures that only require the repetition of a sub-transaction, and failures that allow the reuse of a sub-transaction. Here is example of the Adjourn state [8-9].

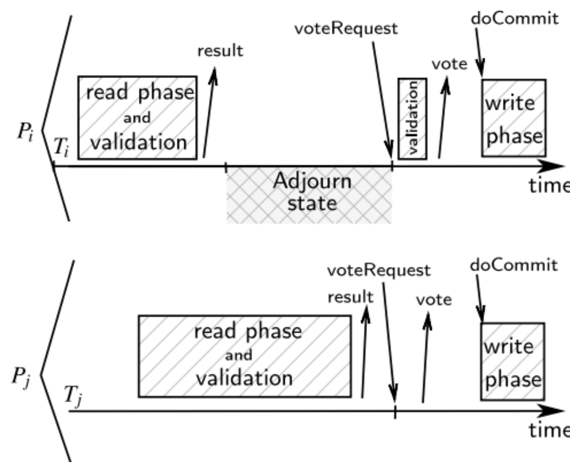

Figure 3. Two sub-transactions executed at Pi and Pj

Fig. 3 shows an example application of the Adjourn state when validation-based concurrency control is used. It shows two resource managers P_i and P_j that try to commit the sub-transactions T_i and T_j, respectively. (T_i and T_j belong to the same global transaction T). As the execution time of P_j in the read phase and the validation phase is more than P_i, P_i has to wait for a longer period of time for the vote Request message to arrive. However, P_i does not know about the delay of P_j In order to avoid blocking of concurrent transactions after the validation phase, P_i into the Adjourn state and unblocks the occupied resources. After P_j has successfully sent the result, the coordinator demands the vote of P_i and P_j. Since T_i has entered the Adjourn state, P_i must perform the second validation phase before T can leave the Adjourn state and can send the vote to the coordinator.

Commit Tree. If a sub-transaction T_i might be restarted as T_i' in case of concurrency conflicts. In this case, T_i and other sub-transactions T_j that has been invoked by T_i and are either invoked by T_i with different parameters or are not at all invoked by T_i', can are aborted. In order to abort these sub-transactions, the coordinator must learn about the invocation hierarchy. For this purpose, the invocation hierarchy and the commit status of the involved sub-transactions is stored in a data structure called "Commit tree". To generate the Commit tree, each participant that sends a result from the Initiator attaches the IDs of all invoked sub-transactions[10]. The Initiator then creates the Commit tree for a transaction and passes it to the coordinator, which is responsible to maintain the tree in case of restarts. However, a sub-transaction does not send a message to a sub-transaction that is a parent sub-transaction in the invocation hierarchy (with the exception of the parent being the Initiator).

Each Commit tree belongs to exactly one global transaction and stores the following variables: the global transaction ID; a tree structure containing Commit tree nodes; the sub-transactions of have sent the result; a list open sub-transactions of known transaction IDs, for which the result has not yet been received by the Initiator. Furthermore, each Commit tree node store: the sub-transaction ID of the sub-transactions represented by this node; the ID of the resource manager running the sub-transaction; the caller transaction ID of the parent sub-transaction; 0 or more IDs of invoked sub-transactions.

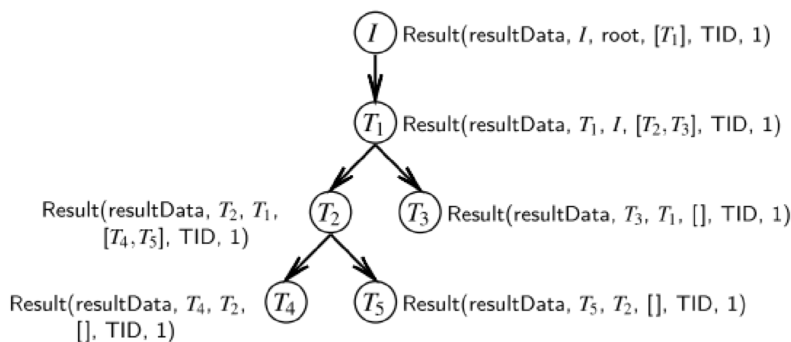

Figure 4. Commit tree example

Fig. 4 shows an example Commit tree. Each result message of a sub-transaction Ti includes the result data, the ID of Ti, the ID of the parent sub-transaction that has invoked Ti, a list of sub-transactions that have been invoked by Ti, the global transaction ID TID to which Ti belongs, and a sequence number. When the initiator receives the result of T1, the node T1 is created. Since the sub-transaction T1 has invoked the sub-transactions T2 and T3, the vote for committing to the sub-transactions T2 and T3 is also required to commit the whole transaction. Therefore, these nodes are added to the Commit tree as well. The initiator builds this Commit tree dynamically and determines when all sub-transactions needed for starting the atomic commit protocol execution is finished. Since the information about invoked sub-transactions is sent along with a result message of the parent transaction and the parent's result can be received later than the child's result, it may be the case that a sub-transaction's result cannot be immediately assigned to a node connected to the Commit tree. In this case, the result is stored in a list of unassigned nodes and this node is connected to the Commit tree after the corresponding parent sub-transaction's result has arrived.

Summary

This paper mainly the concurrent control of distributed transaction management as the theme, through collecting and reading the relevant literature, summarized the different locking scheme to solve the applications conflict of concurrent, and distributed transaction model based on Web services which improve the unreliable network transaction performance, and reduce concurrent transaction blocking "Adjourn state" method, optimization of concurrency control "Commit tree" structure. Table 2 summarizes the advantages of the above methods.

Table 2 Methods of advantage

Method	Advantage
Locking granularity	Different locking granularity control is adopted for different practical applications, and the efficiency of transaction is improved.
Adjourn State	Rational utilization of resources, saving the waiting time of the transaction processing, avoiding the occurrence of obstruction
Commit Tree	Coordinate the sub transactions of different levels to prevent the occurrence of concurrent conflicts

Acknowledgements

The Key Project Supported by Industrial Science and Technology in Shaanxi Province under Grant No. 2015GY104. The Project Supported by Natural Science Basic Research Plan in Shaanxi Province of China under Grant No. 2014JQ5193. The Project Supported by Youth Science and Technology Innovation Fund of Xi'an Shiyou University under Grant No. 2014BS13.

References

[1] Seppo Sippu, Eljas Soisalon- Soininen. Distributed Transactions. Data-Centric Systems and Applications. 2014, pp.299-325.

[2] Jianjiang Li, Qian Ge, Jie Wu, Yue Li, Xiaolei Yang and Zhanning Ma. Research and implementation of a distributed transaction processing middleware [J]. Future Generation Computer Systems (2016).

[3] Coulouris George, Dollimore Jean and Kindberg Tim. Distributed Systems Concepts and Design. China Machine Press (2008).

[4] Ling Liu, M. TAMER Ö ZSU. Distributed Concurrency Control. Encyclopedia of Database Systems. 2009, pp.879-883.

[5] Sha, Baode Fan, Xiuchuan Wu and Lanfang Lou. Research of Controlling Method of the Lock Granularities in Distributed Transactions. International Journal of Systems and Control. Volume 1, 83-90(2009).

[6] Stefan Bosse, Florian Pantke. Distributed computing and reliable communication in sensor networks using multi-agent systems[J]. Production Engineering (2013), p.71.

[7] Jan Mendling. Business Process Execution Language for Web Services [J]. EMISA Forum (2006), p.26:.

[8] R. L. Smelyansky. Model of distributed computing system operation with time[J]. Programming and Computer Software (2013), p.395:

[9] Sebastian Obermeier, Stefan Böttcher, Martin Hett, Panos K. Chrysanthis and George Samaras. Blocking reduction for distributed transaction processing within MANETs[J]. Distributed and Parallel Databases (2009), p.253:.

[10] Manjula K A, Karthikeyan P. Distributed Computing Approaches for Scalability and High Performance [J]. International Journal of Engineering Science and Technology (2010), p.26.