Research of concurrency control protocol based on the main memory database

Yonghua Zhang[*]

Shijiazhuang University of economics, Shijiazhuang, Shijiazhuang, China

Received 1 October 2014, www.cmnt.lv

Abstract

The concurrent access of data can cause the inconsistency. How to control it efficiently is one of the question of the first importance in the database application system. In this paper a management mechanism called dynamic multi-grain lock and a related concurrency control protocol were given.

Keywords: concurrency access, management mechanism, dynamic multi-grain lock, protocol

1 Concurrency control overview

In the database system, the transactions can be executed serially one by one, that is, there is only one transaction running each time and the other transactions have to wait until the end of that transaction. In order to fully utilize the system resources and develop the feature of database shared resources, the multiple transactions to be executed in parallel should be allowed.

In the application system, the process for the multiple users to concurrently access the same data is called concurrent access of the data. When the multiple users access the database concurrently, there would be the situation of multiple transactions accessing the same data at the same time. If no control has been applied in the concurrent operation, it is possible to access the wrong data, which would destroy the database consistency. Therefore, the database management system should provide the concurrency control mechanism. When the multiple users access the database concurrently, there would be the situation of multiple transactions accessing the same data at the same time. If no control has been applied on the concurrent operation, it is possible to access the wrong data, which would destroy database consistency. Therefore, the database management system should provide the concurrency control mechanism. For the main memory database system, the data locking overhead and processing overhead are almost the same. The access speed of memory is much faster than that of the disk, which has significantly reduced the transaction execution time than the disk database; correspondingly, the lock occupation time has been shortened dramatically. Therefore, the essential advantages and necessity of fine grained locking have been lost. Hence the greater grain lock is generally adopted in the main memory database such as relationship level or database level, which has undoubtedly reduced the complexity of concurrency mechanism and the burden in the system, thereby improves the overall system performance.

2 Concurrent executions of transactions

If the concurrent executions of transactions are not reasonably scheduled, the transactional isolation will be destroyed in causing the inconsistency of the data storage area. Generally speaking, the concurrency control is not required if the access data set of the concurrent transactions is not correlated or intersected, or the write set of transactions and the reading set of the other transactions are not intersected. However, the data storage area is a shared resource. The sharing way of the storage information is to allow the multiple user access in modifying the same data. That means the relations between the transactions are much close and the access data is intersected to a great extent. For instance, if an update operation and an operation are conducted concurrently, the writing and reading data will be contradicted. If two or more updates are conducted concurrently, the update information may be lost. Therefore, no correct access data is obtained without the corresponding concurrent access control method. How to control the concurrent access of data efficiently is one of the question of the first importance in the database application system.

Definition 1: The definition of the schedule S is based on the transaction set $T=\{T_1,T_2,...,T_n\}$, which it is an operation series of the multiple transactions.

Definition 2: In the schedule S, the operations of each transaction are not overlapped (namely, in sequence); it will be the serial schedule.

Definition 3: The schedule S is serializable. When the conflict S is equivalent to a serial schedule, the serial schedule is generally called conflict equivalence serializability.

The basic function of a concurrent controller (or scheduler) is to generate a serial schedule in running the transactions. In the main memory database system, the majority of the concurrent control protocols are based on the serializable theory. The concurrent control protocol is used to control the schedule of the data of which its major

[*] ***Corresponding author's*** e-mail: 493738175@qq.com

objective is to maintain the data consistency and provide the maximized concurrency.

In some cases, it is able to appropriately loosen up the requirements of consistency. The main memory database is the shared data inherent in the memory. While the transactions are conducted concurrently in the different threads, it may cause the conflict to access the same data object in the main memory database. The concurrent control protocol is a strategy to solve the conflicts between the transactions.

The common conflicting modes are stated:

"read-write" conflict and "write – write" conflict.

"read-write" conflict: To access the same data object may cause the conflicts while a transaction is running the "read" operations and another transaction is running the "write" operation.

"write-write" conflict: To access the same data object may cause the conflicts while a transaction is running the "write" operation and another transaction is also running the "write" operation.

In the main memory database, to resolve the conflict based on the concurrent control protocol mainly depend on two strategies:

Wait: terminate the conflicting operation caused by one transaction and keep the pending state until the operation of another transaction is completed. For the concurrent transaction control study in the main memory database, three kinds of waiting strategies are derived:

1) No wait: the waiting transaction is immediately ended instead of the completion of another transaction operation.

2) Wound wait: according to the arrival time of each transaction, the transaction in running the data will be ended when the transaction in obtaining the data is arrived in an earlier way; otherwise, the transaction in obtaining the data will be ended.

3) Wait die: according to the arrival time of each transaction, the transaction continues to wait when the transaction in obtaining the data is arrived in an earlier way. Otherwise, the transaction is ended.

Revert: undo the conflicting operation. When the transaction is reset, it is able to go back to the initial stage.

3 Implementation of main memory database lock

There are two major approaches to improve the performance in MMDB. First of all, change the database storage structure, put the tuple data and data together, achieve a direct access of the metadata and decrease the overhead of the metadata; secondly, the dynamic multi-grain lock mechanism is applied in the concurrent operation. In the lock mechanism, it is required to obtain the corresponding lock before the transactions access the data object. There are two basic types of locks: read lock (shared lock) and write lock (exclusive lock). Multiple transactions can share the read lock of the same data rather than the write lock of the same data. We have adopted the management mechanism of the dynamic multi-grain lock in our research system. The lock manager has developed a data lock and is applied to record the distribution status of the data lock. The status of the data lock includes read locked, write locked and unlocked. The data lock has recorded all the transactions of the locked data. The locked data transactions are divided into read lock holder and write lock holder. At the same time, it will record all of the blocked wait transaction lists in applying the data lock. In the lock manager, there is a compatibility list. When the transaction lock is applied, the lock manager will determine whether to block the lock applicant according to the compatibility list; when the transaction lock is released, the transaction will be aroused to obtain the lock when the blocked transaction is allowed to obtain the lock. In terms of the manager, the transaction will lock the data item at the logical level. The manager will schedule the grain from the dynamics of the lock. When the conflicts are decreased, it is suggested to use the table lock; when the conflicts are enhanced, the tuple lock is applied.

There is a close relationship between the implementation of MMDB lock manager and the database organization. The MMDB system is composed of data and metadata. Each segment will correspond to a table in the database. It is known that there is a direct correlation between the metadata and segment in the table lock, between the metadata and partition in the tuple lock.

While the transaction submits the lock request and MML receives the lock requests, it needs to discover the lock grain information in the corresponding segment control block that will be converted into the table lock; if the tuple lock is detected, the tuple lock is applied. If the transaction sends the table lock request, the table lock is applied without being refined.

Dynamic Reduction of Lock Grain

The general condition of the dynamic reduction of lock grain is stated in the following part. When the lock grain is tuple lock, the lock grain information can be set as the table lock when all the locks are compatible. According to the transaction information in holding the lock, it is able to set the control block of the table lock.

Dynamic Increase of Lock Grain

While increasing the lock grain, it needs to record the tuple lock requests of the transaction in the active state. When it needs to use the tuple lock, the memorized lock will be converted into the real tuple lock. In addition, the lock grain information is set as the tuple lock in the table.

The locked grain is closely correlated with the system concurrency and the overhead of the concurrent control. In general, the locked grain is greater with fewer locked objects, smaller selectivity, lower concurrency and smaller overhead; on the contrary, the locked grain is smaller with more locked objects, greater selectivity, higher concurrency and greater overhead.

At the logical level, the distributed database is an entity. At the physical level, the distributed database is stored in the varied physical nodes. Therefore, the chief and prompt issue is to effectively conduct the concurrent control of the data in the database application system. The current task is to establish the reliable and feasible concurrent control strategy in the remote database.

4 Self-control lock

4.1 OVERVIEW OF SELF-CONTROL LOCK

The so-called self-control lock is to discard the lock manager while the lock is applied and released through the shared memory operations. Since there is no lock management progress, it is able to avoid the low efficiency in the management process. All the requests and releases are directly operated in the shared memory while the index does not require intra-process communication.

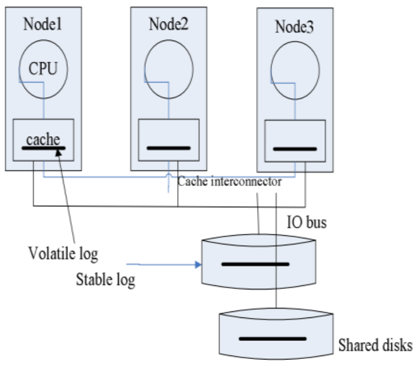

FIGURE 1 Database System Structure

In terms of the self-lock, the data items are correlated, namely each data item corresponds to a LCB. A LCB will record the current lock types and two transaction tables that one is used to record the current transaction in holding the lock while the other is used to record the transaction in waiting the lock. Seemingly, there is no big difference for the mechanism and the multi-processor developers to realize the semaphore. In fact, there are the big differences: firstly, it is different from the semaphore, the database lock must be controlled so as to ensure the failure atomicity of the transaction; secondly, the database lock is also used to support the other locking modes other than the shared lock and exclusive lock (Figure 1).

4.2 IMPLEMENTATION OF SELF-LOCK

4.2.1 Operation of self-lock

In view of the application conditions of the lock, a request includes the name and type of the lock. After the hash processing of the name of the lock, it will be converted into the corresponding lock control block (LCB). All the updates are conducted through a key component. If the requested type is compatible with the current type of LCB and there is no conflicting wait process, the record about the process and the request type is added in the holder lists and it is successfully locked. Otherwise, the record is added into the wait process lists and the system will return back the unapproved information to the requester. The strategy of releasing the lock is similar. While finding the corresponding LCB, the corresponding record will be deleted in the holder list. If there are the compatible lock requests in the waiting process, the request will be approved. The strategy is similar with the traditional lock mechanism. In the traditional lock mechanism, there is the corresponding lock manager for each lock to conduct the relevant operations rather than using the key component. The lock operation code can be independently conducted and it is able to ensure the specific LCB consistency through the key component. The communication efficiency is much higher than that of inter-process communication.

4.2.2 Recovery

When the different records of the same page are locked through the transactions at the different nodes, it is required to obtain the renewal and cancelling information so as to ensure that the transaction is not quitted for the node failure. To renew the record is temporary while to cancel the record is to store the information. For instance, before the record is transferred to another node in one page, the corresponding transaction will be recorded in the log on account of the renewal and cancelling information for the processed database object. In order to ensure that the renewal record of the transaction is recorded before it is transferred; the transaction can hold a short-term lock until the renewal information is recorded.

4.2.3 Ensure the failure atomicity of the locked space

Due to the influence of the consistency protocol, LCB is appeared in one node. While two transactions at the different nodes obtain the same compatible lock, LCB will be located at the node in obtaining the lock. Therefore, when one node is broke, some information of the transaction will be lost rather than all of the LCB. In order to ensure the failure atomicity of the transaction, the measures are taken: before the transaction obtains the lock, it needs to record the transaction identifier (Tm); in addition, before the lock is requested, it will temporarily record the name of the lock and apply the transaction's Tm. These logs are used to ensure the consistency of the space while the node is failed.

4.2.4 Relevant Performance Analysis

The self-lock request can be directly applied to the shared memory in greatly improving its efficiency. In the SD and SM database, the mechanism of the high lock grain is similar. Through the interruption rate evaluation of the large-scale share memory multi-processor system, it is seen that for the general locking condition, CL's overhead is increased with the conflicting linear growth. The interruption rate is a constant. For the hot lock, the interruption performance is almost an order of magnitude greater than that of CL.

5 Recoverable user-level spin lock

What is spin lock? It is a lock mechanism to protect the shared resources. When the conflicts are reduced, the efficiency of the spin lock is much higher than that of the semaphore. Spin lock is busy waiting process in consuming the CPU resources. However, it will not cause the sleeping of the thread and the user state will not be switched to the system status. Spin lock is applied when the lock hold-up time is shorter with more CPU resources. In order to implement the spin lock, it needs to set the spinning times and prevent the endless loop. In terms of the recursive call, the spin lock is forbidden of which it will trigger the dead lock. For many spin locks, the atomic operation is not available in obtaining the lock and registering the holding information. When the key component is terminated or the processing speed is slow down, it is unable to correctly determine the holder. It is able to conduct the recoverable user-level spin lock since it is able to correctly determine the information of the holder of the lock and be applied to the re-installation of the protected data item and the subsequent lock release operation. In this way, the system rebooting is avoided.

5.1 Implementation of the recoverable spin lock

First of all, it needs to add a data structure in each process to record the required lock. It is called want domain. In this way, it is helpful to determine the process set in holding the lock. During the process of determining the holder, the lock may be requested by the other processes in causing the uncertainty of the process set. Therefore, it needs to add a semaphore for the lock in determining the holder. If it is in progress, the other processes may give up the request of the lock and want domain. The pseudo code of the corresponding data structure is stated as follows:

```
struct SafeSpinLock{
        int lock;
ProcessID owner;
        int cleanup_in_progress;
};
struct LockAccessRecord{
SafeSpinLock*wants;
};
```

The corresponding cleaning process is stated as follows:

Set the lock's cleanup_in_progress to prevent the missing of the want domain, and get the lock in the cleaning process.

1) Decide all the possible processes in holding or obtaining the lock

2) Conduct the loop operation until either of the two conditions is satisfied:

A. The status of the spin lock becomes clear in being idle or held by the active process.

B. The process set is empty. The lock is not held by any active process. If A is available, the locked status is immediately determined. Otherwise, it has to wait for a period of time. Some processes are excluded and re-organized. If A is not satisfied for all the time, B is finally satisfied. The lock may be idle or being held by a dead process. If the lock holder information is no_progress in the dead process, it is known that the process is ended in obtaining the lock and registering the information, or releasing the lock. Under such conditions, the lock will be released at ease; if the holder information is known, it can conduct the recovery operation and release the lock again. When there is the deal process or the waiting time is too long for the lock request, it needs to call the cleaning process in releasing the lock or conducting the corresponding recovery operation.

5.2 Relevant Performance of spin lock

Since the spin lock has escaped the system, its operation efficiency is higher than that of the semaphore. If it is unable to correctly get the holder information, it needs to restart the system and recover the information when the holding process is ended. It is a huge overhead in overriding the original advantages of spin lock. Based on the above algorithms, it has no need to restart the system, greatly improving the processing efficiency based on the dead process and giving full play to the merits of spin lock.

6 Proposal of the protocol

According to the current research on the main memory database and the characteristics of main memory database, the matching concurrent control protocol is proposed in the paper. The protocol model is introduced to ensure the serializability of the transaction and preventing the other transaction to read the dirty data without delivering the transaction.

Traditionally, the lock based on the concurrent control protocol includes read lock and write lock. The share and non-share relations are applied between the locks. In this way, there are three kinds of the blocking approaches in the system:

write block
read block
write block

In the system model, there is a new relation between read lock and write lock, that is to say the orderly sharing relation in cancelling the above mentioned first and third blocks. For instance: in order to cancel the read-write block, the other transaction t_i can still conduct the write lock on the data object when the transaction t_j has conducted the read lock on a data object. At this time, it is known to be the orderly sharing relation between t_i read lock and t_j write lock. In order to ensure the serializability of the transaction, the designed protocol shall conform to the following rules:

Rule 1: If there is the orderly sharing relation between the obtained lock of the transaction t_i and the holding lock of the transaction t_j,

The corresponding t_i operation is conducted until the corresponding t_i operation is completed (controlled by the scheduling procedure).

t_i is submitted until the ending/submission of t_j.

The ending of a transaction will cause the consecutive ending of the transactions in reading the "dirty data", which it will have a great influence to the system performance. For the second blocking approach, the orderly sharing relation is not available in our established locking model. Based on the protocol, the transaction can just read the submitted data.

Suppose $T = \{t_1, t_2,...,T_n\}$ is the transaction set of the system index and $D(t_i)(i=1,2,\ldots,n)$ is the access set of the transaction t_i. As well, $RL(x, t_i)$ represents that $t_i \in T$ conducts the read lock on the data object $x \in ED(t_i)$; $WL(x, t_i)$ represents that t_i conducts the write lock on x; $OS(RL(x, t_i)), WL(x, t_i)$ represents there is the orderly sharing relation between the holding read lock of x based on t_i and the write lock of x based on t_j. PRI(t_i) represents the priority of the transaction t_i. The protocol can be briefly described as: when the transaction $t_i \in T$ is implemented, $x \in D(t_i)$, t_i will conduct the locking operation on the x. In the submitting or ending process, t_i will release the holding locks.

When t_i tries to conduct the write lock on x, t_i can still obtain the write lock on the data object if x is read or write locked by the other transactions. As well, $OS(RL(x, t_j))$, $WL(x, t_j)$ or $OS(RL(x, t_i))$, $WL(x, t_j)$. When t_i is ready to be submitted, it still keeps the waiting status when Rule 1 is not satisfied or the date of t_i is not expired. Otherwise, it will be submitted when Rule 1 is satisfied. If the date of t_j is expired, t_i has the top priority. It will be submitted through the transactions when it is ended previously or has the OS relations.

When t_i tries to conduct the write lock on x, the transaction t_i can still obtain the read lock on the data object when x is conducted the read lock through another transaction t_j.

If x is read locked by t_i, the data object is write blocked through the other transactions, that is to say.

If $PRI(t_i) \ll PRI(t_j)$, t_i can just obtain the read lock when t_j is submitted or ended.

If $PRI(t_i) \gg PRI(t_j)$, it needs to estimate whether the transaction t_i can be completed at the deadline (to judge whether it is completed at the deadline, it can compare the present time with the filled deadline of the transaction and the running time estimates). If t_j can be completed at the deadline, t_j will be ended and t_i will be allowed to conduct the read lock on x; if t_i cannot be completed at the deadline, t_i will be ended.

It is seen that the introduction of the orderly sharing relation can greatly reduce the blocking situation in the system, but the submission of the current transaction may be postponed. The major design objective of ARTDEIS is to satisfy the time limit of the initiative real-time transaction rather than the response speed. Therefore, the delay of the submission does not go against its major design principle or stop the read/write operation of the other transactions. When some high priority transaction tries to end a low priority transaction, it will estimate whether the high priority transaction can be completed at the deadline so that it can prevent the running transaction being ended by the other ending transaction (It is called wasting end). Before the transaction is submitted, all of the databases in conducting the write operation are locked. Therefore, the other transactions cannot read the uncommitted data so as to ensure the serializability of the transaction.

7 Summary and innovations

In this paper, the author investigates the concurrent control mechanism in the traditional database system and analyzes the general control approaches. Based on the characteristics of the main memory database, the matching concurrent control protocol is proposed to ensure the serializability of the transaction.

References

[1] Wang H, Pan Z 2004 Analysis of Data Structure in Main Memory Database *Modern Electronics Technique* **27**(3)

[2] Ma H, Yang B, Yao J 2003 Applied Multi-Dimensional Index Structure in the Main Memory Database System *Computer Engineering and Applications* **39**(29)

[3] Liu Y, Liao G 2003 Applied Recovery System for Real-Time Main Memory Database *Journal of Chinese Computer System* **24**(3)

[4] Wang R, Huang Q, Tang L, Li W, Chen Q, Long K 2006 Memory Management of Embedded TCP/IP Protocol *Microcomputer Information* 3-2 69-72

[5] Zhou Y 2010 Improved Locking Protocol in Main Memory Database *Computer Systems Applications* **19**(12)

[6] Zhang Y, Lan L 2012 Concurrent Access Algorithms Based on the Multi-Grain Locking Transactions *Microcomputer & Its Applications* 05

[7] Sang C 2013 Research on Consistency of Multi-Tenant Database [*D*]*Shandong University* 2013

[8] Pandis I, Johnson R, Hardavellas N, Ailamaki A 2010 Data Oriented Transaction Execution *VLDB*

[9] Curino C, Jones E P C, Madden S, Balakrishnan H 2011 Workload Aware Database Monitoring and Consolidation *SIGMOD*

[10] Zheng Y, Sang C, Meng X, Li Q Tenant Oriented Lock Granularity Adjustment Strategy in the Shared Storage Multi-tenant Database *ICNDS JDCTA*

Author

Zhang Yonghua, 1979.06.20, Shijiazhuang, China.

Current position, grades: Director, Engineer in Shijiazhuang University Of Economics.
University studies: Shijiazhuang University Of Economics and Xidian University.
Scientific interest: database and database concurrency control, internet, data encryption.
Publications: 11 papers.