Concurrency control technique for enhancing RDBMS efficiency, performance and activity

B. Narayana Babu, C.S.Ramanathan and R. Krishnamurthy
Department of Computer Science, Sri Sankara Arts and Science College, Kanchipuram, India.

ABSTRACT
The Efficiency, Performance and Activity of a RDBMS must be consistent. This consistency is achieved through Concurrency Control. Concurrency control deals with the issues involved with allowing multiple people simultaneous access to shared entities, be they objects, data records, or some other representation. The Concurrency Control mechanism ensures that the database maintains its consistency. Concurrency control of distributed transactions requires a distributed synchronization algorithm, which has to ensure that concurrent transactions are not only serializable at each site where they execute, but that they are also globally serializable. The advanced protocols put forward are focused on increasing the level of concurrency and simultaneously decreasing the rate of transaction restarts. Some protocols also compromise consistency in order to achieve concurrency. Several Concurrency Control algorithms have been suggested and implemented and used in a variety of real world applications. This article reveals the analysis of various Concurrency Control Techniques on the database systems and further discusses some of the Hybrid Techniques, which provide efficient Concurrency Control on database systems.

Introduction
Concurrency control deals with the issues involved with allowing multiple people simultaneous access to shared entities, be they objects, data records, or some other representation. A collision is said to occur when two activities, which may or may not be full-fledged transactions, attempt to change entities within a system of record. There are three fundamental ways that two activities can interfere with one another:

- Dirty read.
- Nonrepeatable read.
- Lost Update.

The Dirty read method is reading uncommitted data (WR conflicts), where one transaction reads data item, which has been modified by not yet committed transaction. Second is unrepeatable reads (RW conflicts) where one transaction reads a data item twice and finds it to be different since another transaction has modified it between the two reads. Third one is overwriting uncommitted data (WW conflicts) where one transaction modifies a data item that has been modified by a not yet committed transaction. This is known as lost update and it’s a potential problem during recovery.

If all schedules in a concurrent environment are restricted to serializable schedules, the result obtained will be consistent with some serial execution of the transaction and will be considered correct. So some type of scheme should be applied in a concurrent database environment to ensure that the schedules produced by concurrent transaction are serializable. Many Concurrency Control algorithms are designed to correctly process transactions that are in conflict. This article discusses some Basic methods and Hybrid methods and gives their pros and cons.

Locking-Based Concurrency Control Algorithms
These algorithms work on the concept that a shared data required by conflicting operations be locked before being accessed. Thus transaction requests a lock on the shared data being accessing them. Once the lock is set, other transactions would have to wait for the locking transaction to release the lock. When a transaction is done with the data then it releases the lock and then the waiting transactions can access the data. In this way the conflicting transactions can be synchronized. Speaking of locks, there are two types. One is a read lock (shared lock) and the other is a write lock (exclusive lock). Only read locks are compatible. Two transactions can lock a particular data in read mode and access it simultaneously.

Timestamp-Based Concurrency Control Algorithms
Timestamp based CC algorithms don’t try to maintain serializability however they try to order the transactions based according to timestamps. Timestamps are unique identifiers assigned to transactions for ordering. The timestamps should be unique as well as monotonic. A transaction manager should always generate monotonically increasing timestamps. The timestamp ordering (TO) rule is as follows:

If we have two conflicting operations as O1 and O2 belonging to transactions T1 and T2 then O1 is executed before O2 if and only if T1 < T2. T1 is known as the older transaction and T2 is known as the younger transaction.

The scheduler, while executing a new operation, checks all the conflicting operations. If the operation belongs to an oldest transaction then the operation is executed. Else the whole transaction to which the operation belonged is restarted and given a new timestamp.

Optimistic Concurrency Control Algorithms(OCCA)
Optimistic algorithms view a transaction consisting of read, compute, validate and write phases. In OCCA the transactions are validated just before the write phase when the modifications are to be written to the database. Thus the transactions are always allowed to initiate without any delay once submitted to the optimistic scheduler i.e. they are allowed to carry out their
Distributed Transactions which update at least one data item. A database consists of Transaction Manager (TM), Schedulers, Data Managers (DM’s), and Database. The distributed transaction can be classified into two categories: (1) Read Only Distributed Transactions (2) Update transactions and makes local copies of its modifications in its private workspace area. After that the transaction needed to be validated so as to check the compatibility of its modifications with the database. If they are compatible then changes are written else the transaction is restarted. The transaction proceeds in three phases namely Read, Validate and Write phases.

Read: The transaction reads the values from the database and writes them into a private workspace.

Validation: When the transaction is about to commit the OCCP checks whether the committing doesn’t conflict with other concurrent transactions. If a conflict exists then the transaction is aborted and restarted.

Write: When the validation phase determines that the transaction doesn’t conflict with other transactions then its private workspace is copied in to the database.

Drawbacks of Above Methods
In locking method there is chance for occurring deadlock, so deadlock detection and avoidance methods has to use to detect the deadlock and solve it.

In timestamp method even though deadlock will not occur, but there is frequent restart and abort of transaction.

In optimistic method to validate a transaction T, a participant computes the set of other validating transactions that T conflicts with, waits for these to commit or abort, and then makes its commit decision. This process can lead to frequent restart or rollback of the transaction, which is the waste of resource and time.

These are some drawbacks of the basic methods. The problem can be easily solved by combining the basic methods to hybrid method.

Distributed Algorithm
In the distributed algorithm we assume that the distributed database consists of Transaction Manager (TM), Schedulers, Data Managers (DM’s) and Database. The distributed transaction is a set of distributed database operations submitted by a user. A distributed transaction T_i consists of sub-transaction T_{j,k} for k = 1 to m, where m = number of DM’s in the distributed database environment.

The distribution transaction can be classified into two categories: (1) Read Only Distributed Transactions (2) Update Distributed Transactions which update at least one data item. A distribution transaction T_i is assigned an identity, which we will call a timestamp. When T_i visits a site, it must carry T_i’s timestamp with it, this timestamp is used to give priority to one transaction over another.

The function of TM is to translate a distributed transaction T_i into its sub transaction T_{j,k} and coordinate the interactions of the T_{j,k} with the distributed scheduler. The TM is also responsible for assigning timestamps.

Here we discuss two distributed schedulers:
(1) Two-phase request distributed scheduler
(2) One-phase request distributed scheduler

Two-Phase Request Distributed Scheduler
The TM at the site where T_i originates sends a message to all the schedulers where there are copies of data items too be updated, requesting read locks and write locks for all the data items in T_i’s readset and writeset respectively. Each scheduler tries to grant all the locks and sends back a GRANTED message if it is successful. If all the schedulers contacted reply with GRANTED, then the transactions can go head and execute.
an ABORT message for $T_i_k$ from the TM, it aborts the sub-
transactions $T_i_k$. To abort transaction $T_i_k$, site $K$ ignores all 
changes made by $T_i_k$ at site $K$ and releases all locks held by $T_i_k$.
If $T_i_k$ holds a read lock on $X$, the scheduler just releases the lock.
If $T_i_k$ holds a write lock on $X$, the DM discards the new value of 
$X$ if $T_i_k$ has created it, the scheduler releases the lock and 
changes the current state of $X$ from write to read.

When $T_i$ is issues a commit, the two-phase commit protocol, 
which ensures atomic commitment at all sites, is initiated. With 
this algorithm, the schedulers will delay responding to the TM 
with the READY TO COMMIT message until all values read 
have been committed. If $T_i$ holds write locks, the READY TO 
COMMIT is delayed until all readers of the old value have 
committed. In this method deadlocks cannot occur because of the 
priority assigned to transactions by their timestamp.

$$\text{TM} \xrightarrow{\text{Request}} \text{Scheduler}$$

$$\text{Result} \xrightarrow{\text{Granted}} \text{Trigger}$$

Fig. 2 Message exchanged for successful lock

$$\text{TM} \xrightarrow{\text{Request}} \text{Scheduler}$$

$$\text{Result} \xrightarrow{\text{Blocked}} \text{Block}$$

Fig. 3 Messages exchanged when a transaction is initially 
blocked or Preempted

**The one-phase request distributed scheduler**

In one-phase request algorithm all the lock request need to be 
granted at the same time. A read lock on data item $X$ is 
only granted if the current status $X$ is $S_{1R}$ (Any data item not 
currently being read or written) or $S_{2RW}$ (several reader reading 
old or new value and one writer), the timestamp of the read 
request is compared with the timestamp of the transaction 
updating $X$. If the read transaction is older than the updating 
transaction, then the read transaction accesses the old value of $X$ 
otherwise it access the new value of $X$. If the current state of $X$ 
is $S_{1RW}$ (several reader and one writer), the read transaction 
access the old value of $X$ only if it is older than the transaction 
updating $X$; otherwise, it is inserting into the waiting queue 
associated with the data item $X$.

A write lock on data item $X$ is granted if the current state of 
$X$ is $S_{1R}$. If the current state of $X$ is $S_{1RW}$ or $S_{2RW}$, the scheduler 
compares the timestamp of the requesting transaction with the 
updating transaction. If the requesting transaction is older than 
the updating transaction, then the scheduler aborts the updating 
transaction and grants the write lock to the requesting 
transaction; otherwise, the scheduler inserts the request on a 
waiting queue which is associated with data item $X$. When write 
lock on data item $X$ is granted, the local scheduler examines the 
reader of the old value of $X$. Any transaction reading old value 
of $X$ which is younger than the update transaction will be 
aborted.

The waiting queue for each data item is in timestamp order. 
When the scheduler changes the current state of a data item, it 
examines its waiting queue, and begins trying to serve this 
requests before looking at the startup queue.

Aborts can be initiated by the scheduler or can be broadcast 
by the controlling TM. When the scheduler needs to restart a 
transaction, it must sent an ABORT message to the controlling 
TM which then broadcast the abort to all other sites executing 
the transactions. When an aborting transaction holds a read lock 
on data item $X$, it just releases the lock. When aborting 
transaction holds a write lock on $X$, it releases the lock, discards 
the new value of $X$ if one has been created, and in all cases 
returns $X$ to state $S_{1R}$. If $X$ was in state $S_{2RW}$ then any transactions holding a read lock on the new value of $X$ must also 
be aborted. Aborted transactions request are placed in the 
waiting queue of the appropriate data items in timestamps order. 
Therefore, although an aborting transaction may cause several 
others to abort, the next time the get their lock they will be 
timestamps order and subsequent aborts should be avoided.

The commit of the update access on $X$ can be performed by 
the scheduler if no other transactions still hold read locks on 
the old value of $X$; otherwise, the scheduler will delay the commit 
and change the current state of $X$ from $S_{1RW}$ to $S_{2RW}$. This stage 
of transaction blends in with the two phase commit protocol use 
to ensure atomic commitment and recovery. During this, the TM 
sends out messages to all participating sites asking if the sites are 
ready to commit. Those sites which respond with READ TO 
COMMIT message must have changed all written values to state 
$S_{2RW}$. Note that while its written data items are in this state, a 
three transactions can still be aborted if an older transaction arrives at 
the site requesting write access to one of these data items. Thus, 
if two transactions are executing at any site at the same time, 
their actions will be executed in timestamp order. Since the two 
phase commit protocol makes sure all sites are ready before 
commit can take place, two transactions either never exists at the 
same time at one time, or they do, and then execute in 
timestamp order at all sites where they overlap. Fig. 4 shows for 
successful request:
Conclusion

The distributed algorithms discussed in this article work based on locking and avoid deadlocks by using timestamps to establish an execution order when conflicts arise. The two-phase algorithm requires pre-declaration of locks, which restrict concurrent access as well as the one-phase algorithm, causes transaction to abort. Some other method like ODL is found to perform better than distributed algorithm.

Reference