CMPSCI 677 Operating Systems **Spring 2014**

Lecture 15: March 24

Lecturer: Prashant Shenoy *Scribe: Stephen Lee*

15.1 Distributed Transactions

Tranasctions protect shared data by allowing processes to access/modify data as a single atomic operation. It is an 'all-or-nothing' approach wherein a failed transaction restores the state of the data to the start of the transaction. However, transactions follow a broader set of properties a.k.a ACID property that was defined by Jim Gray. These properties are:

Atomic: "All or nothing" approach where transaction are atomic. A failure in an operation inside a transaction results in failure of the transaction.

Consistent: Any transaction will bring the system from one consistent state to another. It cannot violate the rules defined in the application.

Isolation: The effect of an incomplete transaction is invisible to other transactions.

Durability: Changes are permanent once a transaction completes.

To guarantee ACID property, all read/write must be inside a transaction. In case of a transaction failure, Abort transaction primitive allows rollback of the entire transaction and returns the state of the data to the start of the transaction. A naive approach to guarantee these properties is to apply a global lock at the start of BEGIN transaction and release the lock at the END transaction. At the END transaction, the transaction commits the modified changes.

There are two types of transactions in distributed systems:

Nested Transaction Transaction are nested and logically decomposed into a hierarchically of subtransactions. Each subtransactions are excuted on independent databases.

Distributed Transaction Data can be distributed across multiple databases. In a distributed transaction, its subtransactions operate on the same database but the database itself can be distributed across multiple machines.

Two methods are proposed to implement transactions:

Private Workspace When process starts a transaction, it conceptually gets a "private workspace" containing a copy of all files and objects where it can make changes to the data. If the transaction aborts, the private workspace is deleted. If the transcation commits, the parent's copy is updated to reflect the changes made in the private copy.

Write-ahead Logs Each transaction makes modfication in place, but it writes the old and new values to a log. If the transaction aborts, system can undo the changes by replaying the log in a reverse chronological order. The log is usually distributed on different machines (i.e rollback the changes made to the start of the transaction).

15.1.1 Concurrency Control

The goal of concurrency control is to allow system to execute several transactions simultaneously such that at the end of the transactions, the data being operated upon is left in a consistent state ie. the final result should seem as if the transcations were run serially. Concurrency control can be implemented in a layered fashion:

Transaction Manager It is the top layer. It is responsible for creating transactions, checking whether a change is consistent, and rollback if necessary.

Scheduler Once a transaction begins, the scheduler deals with the locks. It may get a lock of a specific record, a field, or the entire database if the database supports fine-grained locking.

Data Manager It executes the read or write on the database.

In a distributed concurrency control scenario, each machine has its own scheduler and data manager, but there is only one global transaction manager.

15.1.2 Serializability

"Serialiability" is used to describe whether the execution of transactions looks the same as if they are executed serially.

- Every transaction consists of one or more operations
- A schedule of transactions is a list of operations ordered by time. It indicates the order in which the operations are executed. Since transactions are executed concurrently, operations of different transactions may interleave with each other.
- A schedule is "serializable" if its outcome is the same as if the transactions are executed serially.

15.1.3 Optimistic Concurrency Control

In optimistic concurrency control algorithm, is based on the idea that nothing will go wrong. All operation are simply carried out and synchronization takes place at the end of a transcation. In case of a conflict with other transactions, one or more transaction is forced to abort. Optimistic concurrency control is based on the insight that conflicts are rare in most cases.

Advantages:

- Deadlock free because no lock is used
- Achieves maximum parallelism

Disadvantages:

- Rerun if transcation aborts
- At high loads, probability of a transcation operating on the same data increases resulting in more conflicts

This algorithm is not used widely in commercial systems.

15.1.4 Two-Phase Locking

Two-phase locking is a common technique that guarantees serializability in concurrency control. It follows the following behavior when acquiring locks:

- When a transaction is in its growing phase, it acquires locks and no locks can be released
- When a transaction is in its shrinking phase, it releases locks no further locks can be granted for that transaction

In a Strict two-phase locking, all locks are released at the at the end of the transcation. The advantages of strict two-phase locking are:

- Eliminates cascaded aborts ie. undo commited transactions because it saw a data item that should not have been seen.
- All lock acquisitions and releases can be handled by the system without the transaction being aware of them. In the case of 2PL, the releases have to be handled by the transaction

Deadlock can happen in this algorithm. It can be avoided by defining the order in which locks are acquired.

15.1.5 Timestamp-Based Concurrency Control

Timestamp-Based Concurrency Control is based on Lamport's clocks. Each transaction Ti is a given a timestamp ts(Ti). If Ti intends to do any operation that conflicts with another transaction Tj with timestamp ts(Tj), then ts(Ti) is compared with ts(Tj). If ts(Ti) <ts(Tj), then transation Ti is aborted, and then later restarted with a larger timestamp.

The idea here is that a transaction with a larger timestamp occurs later in time than a transaction with lower-valued timestamp. Since, transaction with lower-valued timestamp is preceding in time, it is aborted.

To implement TS-Based concurrency control, for each record in the database two values are kept: - Max-rts(x) Max timestamp of a transaction that read x, ie. the most recent transaction timestamp that read x - Max-wts(x) Max timestamp of a transaction that wrote x ie. most recent transaction timestamp that wrote x Using these values, decision to abort or commit a transaction can be made. While reading x, if ts(T) <max-wts(x), then abort T else process T and update max-rts(x). Similarly while writing x, if ts(T) <max-rts(x) or ts(T) <max-wts(x), then abort T else process T and update max-wts(x).