Distributed Transactions

School of Information Sciences, Manipal

Monsoon 2016

Agents in Distributed Transaction Control

Local transaction manager (LTM or plain TM) per local DBMS.

Transaction Coordinator per site that initiates a transaction.

- TC is responsible for concurrency control
  - global concurrency control protocols
- TC is responsible also for ensuring distributed recovery
  - recovery logs

Global Distributed Transaction Control

Transaction Coordinator takes responsibility for

- starting the transaction
- creating distributed transactions and assign them to nodes
- monitoring the progress of distributed transactions
- managing the outcomes - commit or abort

Concurrency Control

If there is only one copy of data at any given time instance, locking and timestamping techniques work quite well for distributed transactions too.

Maintaining the consistency of multiple copies of data requires elaborate coordination.

Concurrency Control - Locking Protocols

Single-site lock manager

Distributed lock manager

Primary copy

Majority locking protocol

Concurrency Control - Locking Protocols

Single-site lock manager

- centralized lock manager
- grants locks - employs two phase locking method

Updates are perfomed on all replicas

- Read-One-Write-All

Concurrency Control - Locking Protocols

Distributed lock manager

- all sites have a local lock manager that manage locks for data at that site.
- when a lock is required, site’s lock manager sends a request to the remote lock manager.
- when data is replicated, locks on all copies is required
  - the requesting site may wait for affirmation from lock managers at replicated sites.

Concurrency Control - Locking Protocols

Primary copy method is applicable when

- locality of reference is high.
- updates are infrequent.
- requesters do not demand the latest version of the data.

One copy is chosen as *primary copy*.

Node on which *primary copy* resides is called a *dominant node.*

Global data dictionary records this information.

Concurrency Control - Locking Protocols

Primary copy method operations

- dominant node attempts to lock its copy of the data.
- if the lock is unavailable, this update request waits.
- dominant node informs a backup node about the update.
- dominant node performs update.
- dominant node initiates same update on replicated nodes.
- if the dominant node fails, backup node takes over.

Concurrency Control - Locking Protocols

Primary copy method

- dominant node sends a update completion message to backup node.
- If backup node does not receive this message, locks data and updates the data.
- It informs others that it is the current dominant node.
- It also writes logs for later recovery
  - the original dominant node may recover eventually.

Concurrency Control - Locking Protocols

Majority locking protocol

- transaction requiring a lock sends request to at least half of the replicated sites
- When at least half of the replicas grant the lock, it proceeds with its update
- This protocol is more complex, and has messaging overheads.
- Deadlocks among sites is also hard to detect.

Distributed Commit Protocols

Two phase commit protocol

- phase 1 - voting phase
- phase 2 - resolution phase

Distributed Commit Protocols

Two phase commit protocol

Phase 1 - Voting phase

- coordinator writes <BEGIN COMMIT T> to its log
- coordinator force writes the log to its disk
- coordinator sends a <PREPARE T> message to all cohorts
- each cohort writes <READY T> to its log
- each cohort force writes its log to disk
- each cohort responds with <READY T> vote

Distributed Commit Protocols

Two phase commit protocol

Phase 1 - Voting phase

- coordinator writes <BEGIN COMMIT T> to its log
- coordinator force writes the log to its disk
- coordinator sends a <PREPARE T> message to all cohorts
- each cohort writes <READY T> to its log
- each cohort force writes its log to disk
- each cohort responds with <READY T> vote
- If a cohort cannot commit, it force writes <ABORT T> to its log and responds with <ABORT T> vote.

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

Assume the coordinators receives an <ABORT T> message from some cohort

- it force-writes <ABORT T> to its log
- sends <ABORT T> to its cohorts
- each cohort aborts the transaction by writing to local log.

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

If a cohort times out and fails to vote, the coordinator aborts the entire transaction.

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

Assume TC received all <READY T> votes from its cohorts

- TC writes <COMMIT T> recor d to its log and force-writes it.
- TC sends <COMMIT T> message to all cohorts.
- Each cohort updates its log with <COMMIT T>
- Each cohort acknowledges this with TC.
- TC writes <END T> to its log.

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

If TC fails to receive an acknowledgement from some cohort, TC assumes the cohort has failed

In the case of failures, protocols determines the next course of actions...

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

TC fails during the commit protocol

- If one of the cohorts has an <ABORT T> in its log, the transaction is aborted.
- If one of the cohorts has a <COMMIT T> in its log, the transaction is committed.
- If there is a cohort without <READY T> in its log, TC cannot have deciced to commit. So, the transaction is aborted.

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

TC fails during the commit protocol

- If one of the cohorts has an <ABORT T> in its log, the transaction is aborted.
- If one of the cohorts has a <COMMIT T> in its log, the transaction is committed.
- If there is a cohort without <READY T> in its log, TC cannot have deciced to commit. So, the transaction is aborted.
- If all cohorts have <READY T> and no site has an <ABORT T> or <COMMIT T>, it is impossible to decide the fate of T.
- T is blocked until TC recovers.

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

One of the cohorts fails, and

- once it recovers, if its log has a <COMMIT T>, it performs a *redo*(T)
- If its log has an <ABORT T>, it performs an *undo*(T)
- If its log has no <READY> or <ABORT> or <COMMIT> it must have failed before responding with its vote. TC must have therefore aborted the transaction. Therefore, it performs an *undo*(T).

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

One of the cohorts fails, and

- once it recovers, if its log has a <COMMIT T>, it performs a *redo*(T)
- If its log has an <ABORT T>, it performs an *undo*(T)
- If its log has no <READY> or <ABORT> or <COMMIT> it must have failed before responding with its vote. TC must have therefore aborted the transaction. Therefore, it performs an *undo*(T).
- if its log has <READY T>, it consults TC
  - if TC acknowledges a commit, the cohort performs a *redo*(T)
  - If TC reports that the transaction aborted, the cohort does an *undo*(T)

Distributed Commit Protocols

Two phase commit protocol

Phase 2 - Resolution phase

One of the cohorts fails, and

- once it recovers, if its log has a <COMMIT T>, it performs a *redo*(T)
- If its log has an <ABORT T>, it performs an *undo*(T)
- If its log has no <READY> or <ABORT> or <COMMIT> it must have failed before responding with its vote. TC must have therefore aborted the transaction. Therefore, it performs an *undo*(T).
- if its log has <READY T>, it consults TC
  - if TC acknowledges a commit, the cohort performs a *redo*(T)
  - If TC reports that the transaction aborted, the cohort does an *undo*(T)
  - If TC cannot be contacted, it tries to determine the fate of T from other participants. If it cannot reach them, it waits for TC to recover.