ECS 265 DISTRIBUTED DATABSE SYSTEMS

CONSENSUS ON TRANSACTION COMMIT. TODS'06

MADE BY-

ARCHIT GARG

Agenda

- What is the paper about?
- Two Phase Commit
- Paxos Commit
- Conclusion

Introduction

The distributed transaction commit problem requires reaching agreement on whether a transaction is committed or aborted.

In this presentation we will be looking at the following algorithms for committing a transaction:

- Two Phase Commit
- Paxos Commit Algorithm

Assumptions

- ❑ The algorithms are executed by a collection of processes that communicate using messages
- ❑Each process executes at a node in a network
- ❑A process can save data on stable storage that survives failures.
- ❑Different processes may execute on the same node
- ❑The cost model counts internode messages, message delays, stable-storage writes, and stable-storage write delays.
- ❑The failure model assumes that nodes, their processes, can fail, messages can be lost or duplicated, but not (undetectably) corrupted

Correctness Properties

Correctness Properties are those properties that the aforementioned algorithms must satisfy. There are two properties that must be satisfied :

Safety:

- Describes what is allowed to happen
- Time independent
- Not bounded on message delay

Liveness:

- Describes what must happen.
- Time Dependent

What is a non-faulty node?

A non-faulty node is defined to be one whose processes respond to messages within some known time limit.

ACM Transactions on Database Systems, Vol. 31, No. 1, March 2006

Transaction Commit

A Transaction Commit is referred to saving the data permanently or committing the data permanently to the stable storage at the end of a transaction.

The information in the transaction becomes visible to other users only after a commit takes place.

A Transaction commit is performed by a collection of processes called resource managers.

Safety Requirements

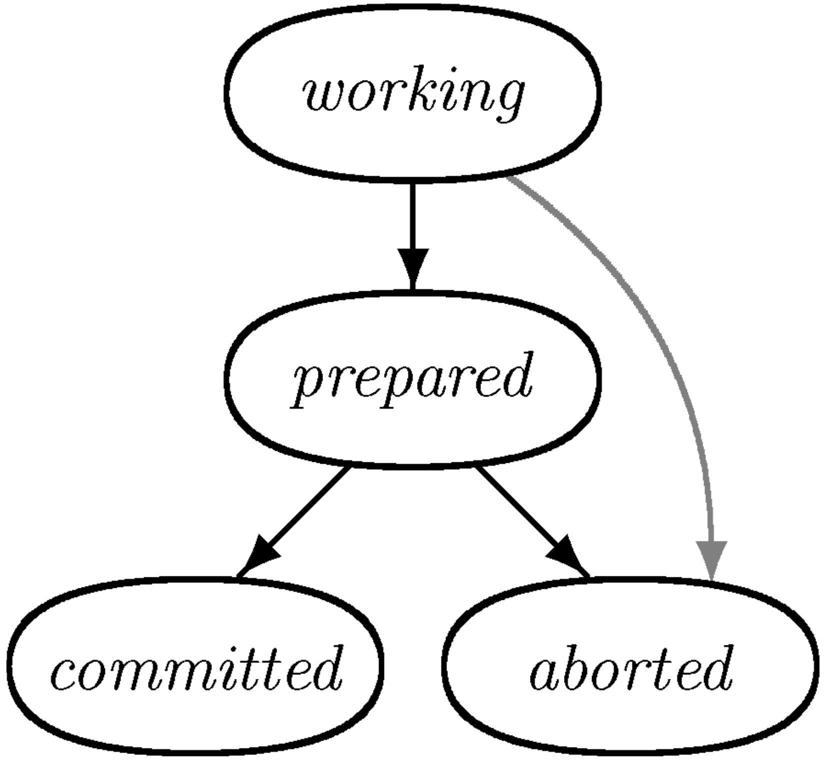
- *Stability*. Once an RM has entered the *committed* or *aborted* state, it remains in that state forever.
- *Consistency*. It is impossible for one RM to be in the *committed* state and another to be in the *aborted* state.

These two properties imply that, once an RM enters the *committed* state, no other RM can enter the *aborted* state, and vice versa.

Each RM also has a prepared state.

The requirements imply that the transaction can commit if all RMs reach the *committed* state, only by the following sequence of events:

—All the RMs enter the *prepared* state, in any order;
—All the RMs enter the *committed* state, in any order.
—Any RM in the *working* state can enter the *aborted* state.



The two liveness properties for Transaction commit is as follows:

Nontriviality - If the entire network is nonfaulty throughout the execution of the protocol:

- If all RMs reach the *prepared* state, then all RMs eventually reach the *committed* state
- If some RM reaches the *aborted* state, then all RMs eventually reach the *aborted* state.

Nonblocking – If a sufficiently large network of nodes is nonfaulty for long enough, then every RM executed on those nodes will eventually reach either the *committed* or *aborted* state.

Two Phase Commit Protocol

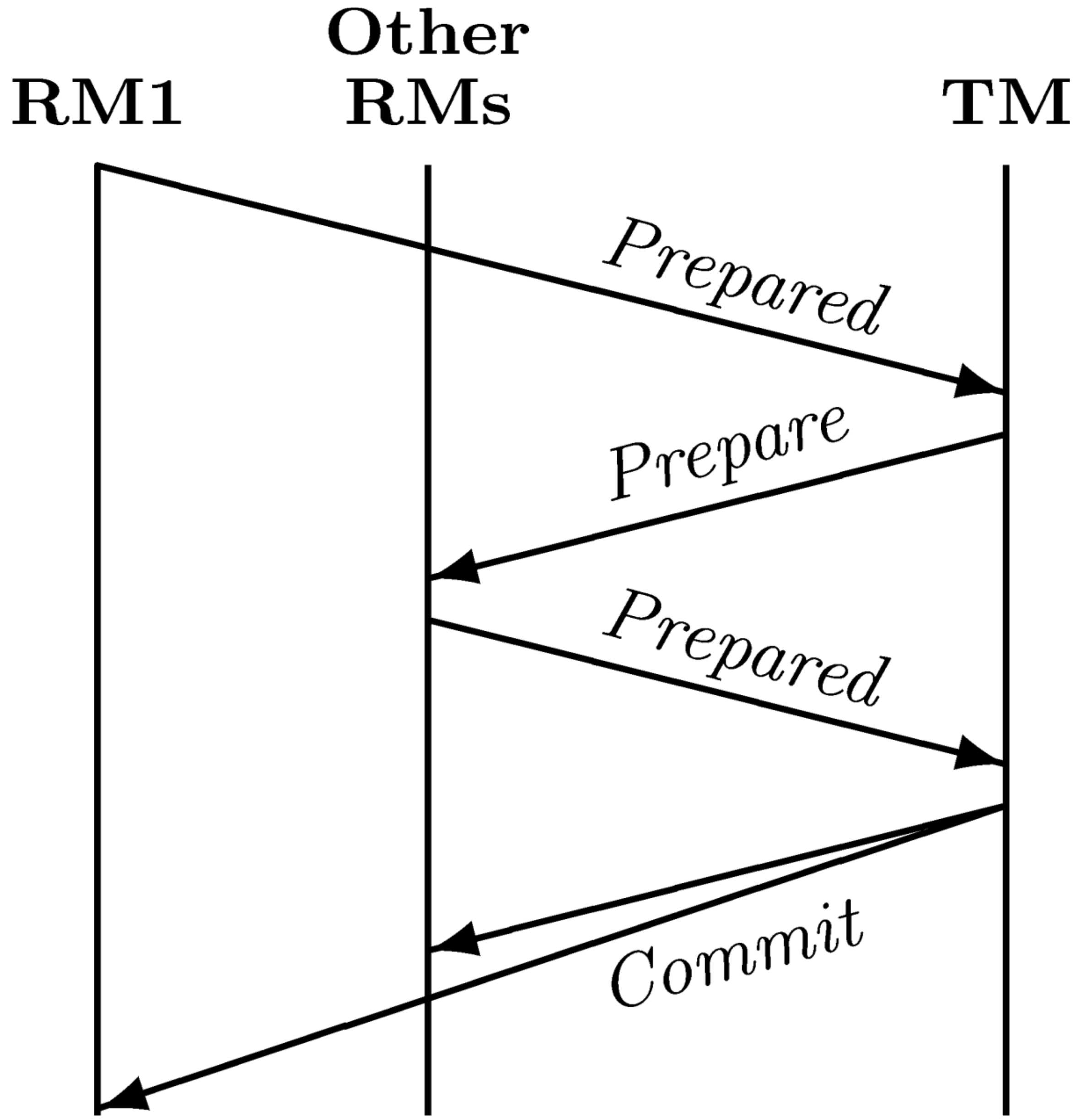
The Two-Phase Commit protocol uses a *transaction manager* (TM) process to coordinate the decision-making procedure.

The TM has the following states:

- *init* (its initial state)
- *Preparing*
- *Committed*
- *Aborted*.

The Two Phase Commit protocol is as follows:

- An RM enters the *prepared* state and sends a *Prepared* message to the TM.
- The TM enters the *preparing* state and sends a *Prepare* message to every other RM.
- An RM that is still in the *working* state can enter the *prepared* state and send a *Prepared* message to the TM.
- *After recieveing prepared* message from all RMs, the TM can enter the *committed* state and send *Commit* messages to all the other processes.
- The RMs can enter the *committed* state upon receipt of the *Commit* message from the TM.



Two Phase Protocol : Abort

An RM can spontaneously enter the *aborted* state if it is in the *working* state.

TM can spontaneously enter the *aborted* state unless it is in the *committed* state.

After the TM aborts, it sends an *abort* message to all RM.

RM enters the *aborted* state.

Spontaneous aborting can be triggered by a timeout.

Failure and Restart

Process failure and restart is easy to handle.

Each process records its current state in stable storage before sending any message.

When a failed process is restarted, it can simply restore its state from stable storage and continue executing the algorithm.

Process failure and restart is equivalent to the process pausing.

Cost of Two Phase commit

—The initiating RM enters the prepared state and sends a *Prepared* message to the TM. (1 message)

—The TM sends a *Prepare* message to every other RM. (*N* − 1 messages)

—Each other RM sends a *Prepared* message to the TM. (*N* − 1 messages)

—The TM sends a *Commit* message to every RM. (*N* messages)

Therefore, a total of: 3N-1 messages

If the TM is on the same node as RM then, the cost of intranode messages can be discounted making a total cost of 3N-3 messages.

Limitations

Two phase commit protocol is a blocking protocol.

A node will block while it is waiting for a message

A single node will continue to wait even if all other sites have failed.

The resources are tied up *forever*.

The protocol is conservative. It is biased to the abort case rather than the complete case.

Paxos Commit

The Paxos algorithm is a popular asynchronous consensus algorithm.

It uses a series of ballots numbered by nonnegative integers, each with a predetermined coordinator process called the *leader*.

One instance of Paxos is executed for each resource manager, in order to agree upon a value(Prepared/aborted) proposed by it.

Participants: The Resource Manager

- N resource managers(RM) execute the distributed transaction, then choose a value (Locally chosen Value) for prepared state iff willing to commit.
- Every RM tries to get its value accepted by a majority set of acceptors
- Each RM is the first proposer in its own instance of paxos

The Leader

- Coordinates the commit algorithm
- All instance of Paxos share the same leader
- Assumed always defined and unique.

The Acceptors

- All the instances of paxos share the same set A of acceptors
- 2F+1 acceptors involved in order to achieve tolerance to F failures
- Each acceptor keeps track of its own progress

Phase Messages:

A process that believes itself to be a newly elected leader initiates a ballot, which proceeds in the following phases.

- *Phase 1a*. The leader chooses a ballot number *bal* for which it is the leader and sends a phase 1a message for ballot number *bal* to every acceptor.

- *Phase 1b*. When an acceptor receives the phase 1a message for ballot number *bal*, it responds

—The largest ballot number for which it received a phase 1a message
—The phase 2b message with the highest ballot number it has sent.

- *Phase 2a*. When the leader has received a phase 1b message for ballot number *bal,*

—*Free* : None of the majority of acceptors reports having sent a phase 2b message, so the algorithm has not yet chosen a value.
—*Forced*: Some acceptor in the majority reports having sent a phase 2b message.

- *Phase 2b*. When an acceptor receives a phase 2a message for a value *v* and ballot number *bal*, it *accepts* that message and sends a phase 2b message to the leader. The acceptor ignores the message if it has already participated in a higher-numbered ballot.

- *Phase 3*. When the leader has received phase 2b messages for value *v* and ballot *bal* from a majority of the acceptors, it knows that the value *v* has been chosen and communicates that fact to all interested processes with a phase 3 message.

Paxos Commit Algorithm

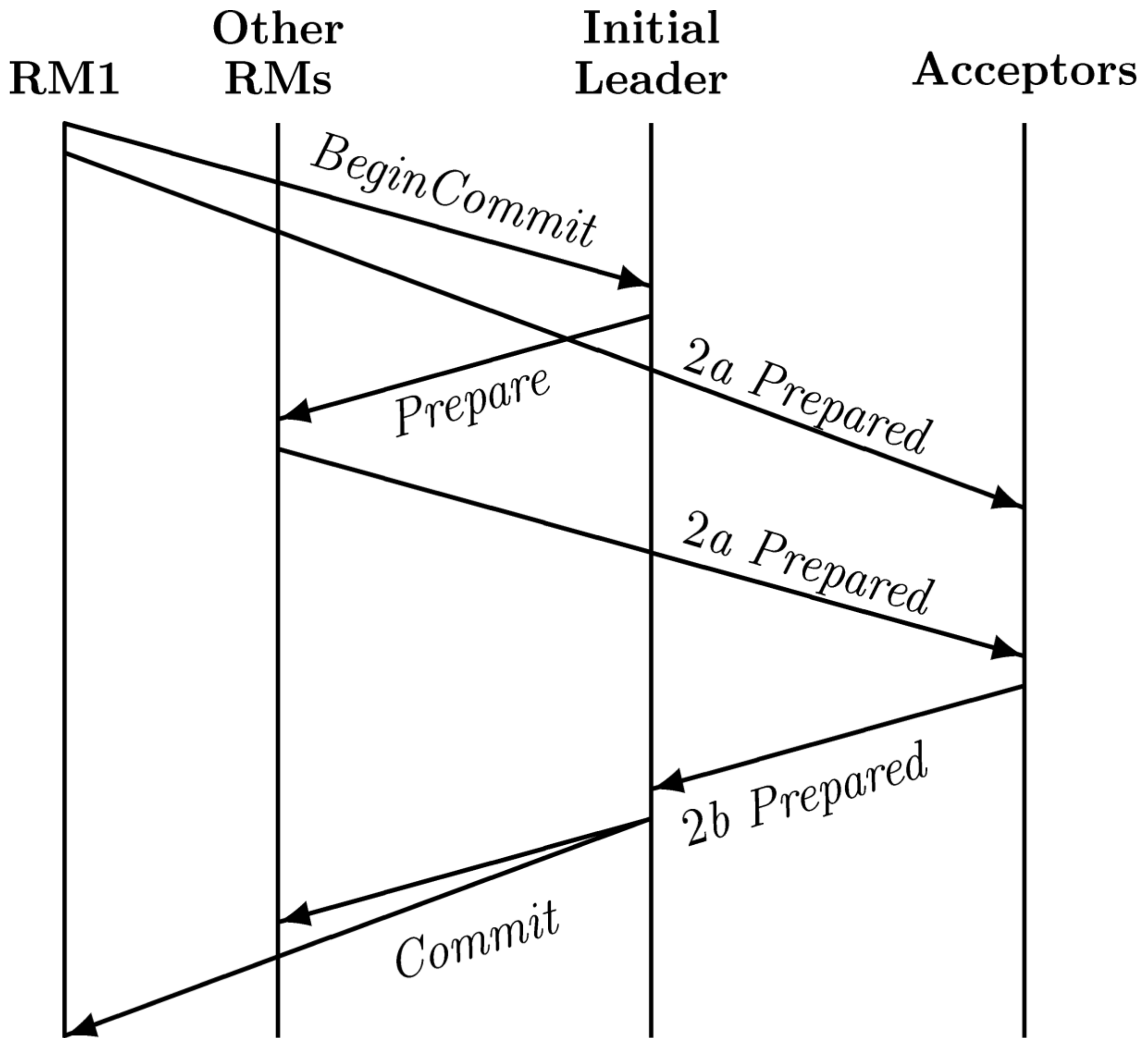
RM decides to prepare and sends a *BeginCommit* message to the leader.

The leader then sends a *Prepare* message to all the other RMs.

If an RM wants to prepare, it sends a phase 2a message with value *Prepared* and ballot number 0 in its instance of the Paxos algorithm. Otherwise, it sends a phase 2a message with the value *Aborted* and ballot number 0.

Acceptor sends its phase 2b message to the leader.

The leader knows the outcome of this instance if it receives *majority of* phase 2b messages for ballot number 0, then it can send its phase 3 message announcing the outcome to the RMs.



Cost of paxos commit

—The first RM to prepare sends a *BeginCommit* message to the leader. (1 message)

—The leader sends a *Prepare* message to every other RM. ($N - 1$ messages)

—Each RM sends a ballot 0 phase 2a *Prepared* message for its instance of Paxos to the $F + 1$ acceptors. ($N(F + 1)$ messages)

—For each RM's instance of Paxos, an acceptor responds to a phase 2a message by sending a phase 2b *Prepared* message to the leader. However, an acceptor can bundle the messages for all those instances into a single message. (F+1 messages)

—The leader sends a single *Commit* message to each RM containing a phase 3 *Prepared* message for every instance of Paxos. (N messages)

Therefore a total of (N+1)(F+3)-4 messages are sent

Co-location

For Two-Phase Commit, colocation means that the initiating RM and the TC are on the same node.

For Paxos Commit, it means that each acceptor is on the same node as an RM, and that the initiating RM is the on the same node as the initial leader. In Paxos Commit without colocation, we assume that the initial leader is an acceptor.

Fast Paxos

We can eliminate phase 3 of Paxos by having each acceptor send its phase 2b messages directly to all the RMs. This allows the RMs to learn the outcome in only four message delays, but a total of $N(2F+3)$ messages are required.

Letting the leader be on the same node as an acceptor eliminates one of those messages. If each acceptor is on the same node as an RM, and the leader is on the same node as the first RM, then the initial *BeginCommit* message, $F + 1$ of the phase 2a messages, and $F + 1$ of the phase 2b messages

can be discounted, leaving $(N - 1)(2F + 3)$ messages.

Paxos versus Two Phase Commit

Two Phase Commit	Paxos Commit
TM makes both commit/abort decision and stores on stable storage	Uses acceptors’ stable storage and replaces TM by a series of leaders
Two Phase Commit can block indefinitely if TM fails	Paxos doesn’t blocks indefinitely if a leader fails.
TM can unilaterally decide to abort	The leader can make abort decision only for an RM.
Two Phase commit will abort if any RM aborts	Paxos commit only requires Phase 2b from a majority of acceptors i.e. 2F+1.

ACM Transactions on Database Systems, Vol. 31, No. 1, March 2006

Transaction Creation and Registration

In a real system transaction commit may have to dynamically allocate RM. To accommodate a dynamic set of RMs, we introduce a *registrar* process that keeps track of what RMs have joined the transaction.

Paxos Commit runs a separate instance of the Paxos consensus algorithm to decide upon the registrar's input, using the same set of acceptors.

The registrar is generally on the same node as the initial leader, which is typically on the same node as the RM that creates the transaction. In Two-Phase Commit, the registrar's function is usually performed by the TM rather than by a separate process.

Transaction Creation

Each node has a local transaction service that an RM can call to create and manage transactions.

To create a transaction, the service constructs a *descriptor* for the transaction, consisting of a unique identifier (uid) and the names of the transaction's *coordinator* processes.

The coordinator processes are all processes other than the RMs that take part in the commit protocol—namely, the registrar, the initial leader, the other possible leaders, and the acceptors.

The descriptor tells the process the names of the coordinators that it must know to perform its role in the protocol.

Joining a Transaction

An RM joins a transaction by sending a *join* message to the registrar.

The RM that creates the transaction sends the descriptor to any other RM that might want to join the transaction.

Upon receipt of a *join* message, the registrar adds the RM to the set of participating RMs and sends it an acknowledgment.

Receipt of the acknowledgment tells the RM that it is a participant of the transaction.

Committing a Transaction

When an RM wants to commit the transaction, it sends a *BeginCommit* message to the registrar rather than to the initial leader. (In Two-Phase Commit, the *BeginCommit* message is the *Prepared* message of the first RM to enter the *prepared* state.)

The registrar then sends the *Prepare* messages to the other RMs that have joined the transaction. The registrar no longer allows RMs to join the transaction, responding to any subsequent *join* message with a negative acknowledgment.

Conclusion

Two phase commit is classical transaction commit protocol. It not fault tolerant because it uses a single coordinator whose failure can cause the protocol to block.

Paxos uses multiple coordinators and makes progress if majority of them are working.

In the normal, failure-free case, Paxos Commit requires one more message delay than Two-Phase Commit. This extra message delay is eliminated by Faster Paxos Commit, which has the theoretically minimal message delay for a nonblocking protocol.

The assumptions made for a nonblocking system cannot be implemented in a purely asynchronous system.

In modern local area networks, messages are cheap, and the cost of writing to stable storage can be much larger than the cost of sending messages. So in many systems, the Consensus on Transaction Commit benefit of a nonblocking protocol should outweigh the additional cost of Paxos Commit.

The stronger definition of transaction commit is not implementable in typical transaction systems, where occasional long communication delays must be tolerated.