CMPSCI 677 Operating Systems **Spring 2014**

Lecture 18: April 2

Lecturer: Prashant Shenoy *Scribe: Sweetesh Singh*

18.1 Atomic Multicast

The principle of atomic multicast says that the system guarantees that either all nodes receive the multicast message or no node receives the message. It is analogous to "All or Nothing" approach.

18.1.1 description

Problem One of the main problems of this approach is fault in the system. If a machine fails, then all nodes must revert back to their previous state.

Solution Each message is uniquely associated with a group of processes. Every time the system sends a message, it guarantees that all the processes in that group will receive the message. If, one process crashes, the system has to detect it, and notify all the members of the group about the crashed process, and the rest of the processes form a new group discarding the crashed process. later the system can reboot the crashed process, which can then join the group again.

18.2 Distributed Commit

In distributed commit, the policy is, either everybody commits to the transaction or nobody commits to the transaction.

18.2.1 Two Phase Commit

This policy has a coordinator process which coordinates the operation. Coordinator initiates voting, and if all processes, including the coordinator, agree with that transaction then it is permanently committed, otherwise, even if one process disagrees, that transaction is aborted. One problem with this policy is if coordinator crashes, the system still has to ensure commitment of the transaction. One way to deal with this is, make all processes decide upon the transaction, but the issue with this policy is that processes will not know what vote the coordinator has casted. Unless they taken into account of coordinator's vote, other processes cannot decide upon the transaction.

18.3 Recap of Fault Tolerance

18.3.1 Two Replication techniques to handle fault tolerance

18.3.1.1 Technique 1

Requests are distributed amongst k replicas . In case of failure of a replica , the requests are redistributed . This technique handles crash fault tolerance . It is a simple and the commonly used technique to handle faults.

18.3.1.2 Technique 2

Each request is sent to all the replicas . All replicas process requests and produce results . Then the replicas vote to make a decision . This handles Crash and Byzantine Faults . However it is much harder a nd more expensive(3k replicas) to implement .

18.3.2 Two Phase Commit (2PC) and Three Phase Commit (3PC)

In Two Phase Commit ,The first phase includes the Coordinator quering all the database replicas on whether a transaction has to be committed or aborted . In the decision phase the results from the replicas are used to make a decision . Even if a single replica wants to abort , the transaction is aborted . This ensures the safety property .

However the failure of the Coordinator blocks the system . In order to handle this blocking a technique called Three phase commit is used. The first two phase is similar to the 2PC . In the third phase , the coordinator tabulates the results and sends them to all the replicas . The replicas send an acknowledgement on receiving this message . Only after receiving the acknowledgement does the coordinator send a commit message.

If the coordinator crashes , the replicas ask each other if they have heard from the coordinator and even if a single replica has a tabulated result (in the precommit stage) the replicas can go ahead and commit . If none of the replicas have a result then the abort the transaction .

The state diagram in the slides shows the various messages passed . The 3PC is always safe irrespective of the failure of the coordinator or the replicas

18.4 Paxos

Consensus is the process of agreeing on one result among a group of participants. This problem becomes difficult when the participants or their communication medium may experience failures. Paxos is a family of protocols for solving consensus in a network of unreliable processors. A consensus algorithm ensures that a single one among the proposed values is chosen. If no value is proposed, then no value should be chosen. If a value has been chosen, then processes should be able to learn the chosen value.

The safety requirements for consensus are:

- 1) Only a value that has been proposed may be chosen,

- 2) Only a single value is chosen, and
- 3) A process never learns that a value has been chosen unless it actually has been.

Liveness requirement:

- 1) Some proposed value is eventually chosen.
- 2) If a value v has been learned, then some learner will eventually learn some value.

Roles in Paxos algorithm:

- Client: issues a request, waits for response
- Acceptor: memory of the protocols, collected into Quorums.
- Proposer: proposes/advocates the Clients request, convinces the Acceptors, resolves conflicts.
- Learner: acts as the replication factor, takes action after Acceptors agree on Clients request progress.
- Quorum: any majority of participating Acceptors

Paxos Works in following two phases

Phase1.

- (a) A proposer selects a proposal number n and sends a prepare request with number n to a majority of acceptors.
- (b) If an acceptor receives a prepare request with number n greater than that of any prepare request to which it has already responded, then it responds to the request with a promise not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.

Phase 2.

- (a) If the proposer receives a response to its prepare requests (numbered n) from a majority of acceptors, then it sends an accept request to each of those acceptors for a proposal numbered n with a value v, where v is the value of the highest-numbered proposal among the responses, or is any value if the responses reported no proposals.
- (b) If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater than n.

Paxoss properties

- P1: Any proposal number is unique.
- P2: Any two set of acceptors have at least one acceptor in common.
- P3: the value sent out in phase 2 is the value of the highest-numbered proposal of all the responses in phase 1.

Paxos Example

- Proposers are p1 and p2.
- Acceptors are a1, a2, and a3.
- p1 sends prepare for proposal 1 to a1 and a2.
- a1 and a2 reply to p1.
- p2 sends prepare for proposal 2 to a2 and a3.
- a2 and a3 reply to p2.
- p1 sends accept request to a1 and a2 for proposal 1 with value pepperoni.
- p1 got to select which value to propose.
- a1 accepts proposal 1.
- a2 does not accept proposal 1.
- a2 promised p2 it wouldn't accept proposals ¡ 2.
- p2 sends accept request to a2 and a3 for proposal 2 with value mushrooms.
- p2 also got to select which value to propose.
- a2 accepts proposal 2.
- a3 accepts proposal 2.
- a2, a3 is a majority of acceptors, so proposal 2 is chosen.
- The chosen value is mushrooms.
- p1 sends prepare for proposal 3 to a1 and a2.
- a1 replies; it last accepted proposal 1 for pepperoni.
- a2 replies; it last accepted proposal 2 for mushrooms.
- p1 sends accept request to a1 and a2 for proposal 3 with value mushrooms.
- Value must match the one from proposal 2.
- a1 and a2 accept proposal 3.

There are following options for learning a chosen value

- Each acceptor, whenever it accepts a proposal, informs all the learners.
- Acceptors informs a distinguished learner (usually the proposer) and let the distinguished learner broadcast the result.

18.5 Recovery

Once the crashed server recovers , operations must be performed for it to recover to the current state . This is done by resynchronizing with the other servers. In case of databases , all replicas maintains logs and the replica which has just recovered can ask another replica for its log and do a log replay to get itself updated . Checkpointing is another means of recovery . The replicas maintain checkpoints and upon a crash it rolls back to its previous checkpoint and then does a log replay from that point .

Independent Checkpointing can lead to inconsistencies in that case the replicas has to rollback until the last consistent state . This cascading rollback can lead to a domino effect . However use of techniques like distributed snapshot solves the problem.