node holding token is single point of failure.

- N+2 messages for one access to CS
- $\mathcal{O}(N)$ messages

11.3 Quorum-based algorithms (week 11)

Instead of asking permission from one node, as in token-based algorithms or from all processes, as in time-stamp-based algorithms, a quorum-based algorithm only needs a subset of all processes to vote for the candidate that wants to access a critical section.

Properties:

- no single point of failure
- permission from subset of processes(request set)
  Assume p = 0.1, i.e. if each node fails with Prob. 10%
  assume only votes from half of the nodes are needed to take decision, then half of the nodes may fail until the system is no longer able to come to a decision. This means, the system can tolerate $\frac{N}{2}$ failures, for N = 100 implies that $p^{50} = 0.1^{50} = (10^{-1})^{50} = 10^{-50}$ failure probability of the mutual exclusion algorithm is very low.
- permission from subset of processes (request set)
- if 2 request sets have non-empty intersection, we are guaranteed that at most one process can access cs
  (e.g. majority needs any subset with at least $[\lceil \frac{(N+1)}{2} \rceil]$ processes
- vote assignment is a set of groups, s.t. each group has majority of votes

Define: Coterie: Let $U = \{u_1, u_2, ..., u_n\}$ be a set and C a non-empty family of subsets, called quorums of U.

Definition Coterie: $C = \{Q_1, Q_2, ..., Q_m\}$ where $\forall 1 \leq i \leq m : Q_i \subseteq U$ is a coterie under U if:

- No quorum is a subset of any other quorum

$$\forall i, j \ : \ i \neq j \neg(Q_i \subseteq Q_j) \text{ (Minimality)}$$

- Every two corums intersect

$$\forall i, j \ : \ Q_i \cap Q_j \neq \emptyset$$

Let C and D be coteries, C dominates D if C $\neq$ D and $\forall Q \in D : (\exists Q' \in C : Q' \subseteq Q)$
A coterie C is non dominated, if no coterie dominates C. Intuitively non-dominant coteries are in some sense optimal, because they can not be further reduced. A smaller coterie is better for reasons of algorithmic complexity.

Metrics for quorums:

1. Quorum size, smaller quorum needs less messages
2. Availability, probability p that there is at least one live quorum in the coterie
3. Load on the busiest node

These metrics are in conflict. There exists no quorum system that is optimal in all metrics.

Voting Systems

- Each process is assigned a number of votes (e.g. 1)
- Let the total number be V
  Majority voting is an example of a quorum, that is a subset of processes with a sum of at least $\frac{V}{2}$ votes.
- Example: two types of quorums, read(R) and write(W) quorums, such that
  1. two parameters R and W, such that R + W > V and
  2. W > $\frac{V}{2}$

If the total number of votes exceeds R, then it is a read quorum, if they exceed W, then it is a write quorum.

1. prevents read-write conflicts
2. prevent w-w conflicts

To Read correct value R replicas must be read to write W replicas must be written.

12 Consistency and Consensus, Distributed Commit

Source: Tanenbaum Distributed Systems, 2007.

- distributed systems use replication to improve performance and/or reliability
- replication for scalability, since it allows to read (and write) in parallel
  How to keep replicas consistent?

Distributed commit is necessary if an item is present on several nodes in a distributed system. Each write operation on the item must be done simultaneously in all nodes, or in none of the nodes.

Requirements and properties

- an operation is performed by a group or none of the group.
- reliable multicast: operations = delivery of a message
- distributed transactions: operations = executions of a transaction
- uses coordinator
- one-phase commit: node sends update to all other nodes who write the update. If a message gets lost, or a node fails and recovers the data in the system is no longer consistent.
- two-phase commit (2 PC) Jim Gray(1978)
- distributed transaction involves several processors each on a different machine
  2 phases with each 2 steps (one message forth and back):
  1. coordinator $\overset{\text{vote request}}{\longrightarrow}$ all participants.

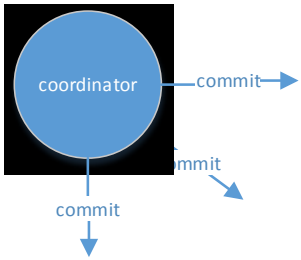
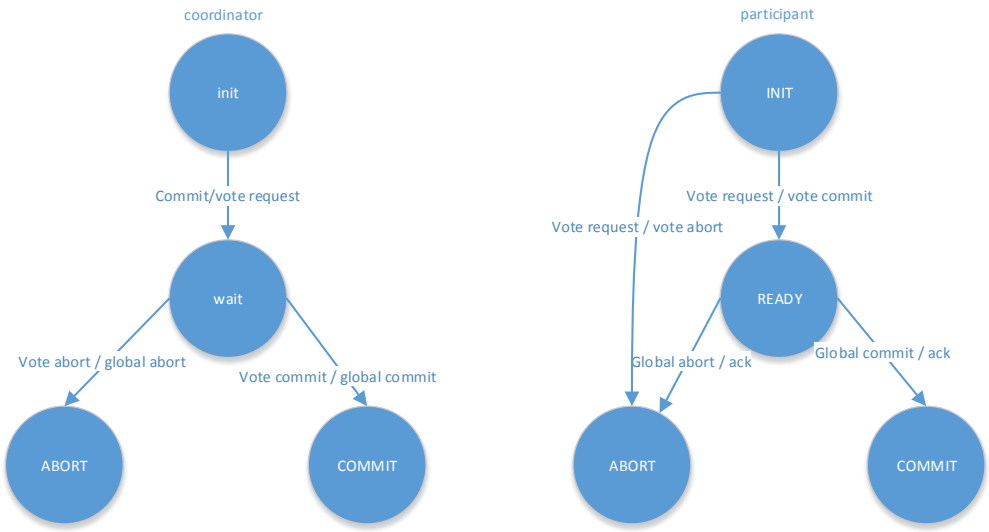


Figure 24:

2. participant $\overset{\text{vote commit}}{\underset{\text{vote abort}}{\longrightarrow}}$ coordinator
3. if all commit, coordinator $\overset{\text{global commit}}{\longrightarrow}$ all participants
   else coordinator $\overset{\text{global abort}}{\longrightarrow}$ all participants.
4. if commit
   participant locally commits
   else
   participant locally aborts.



A process blocks, if it does not receive the message necessary to change its state and proceed in the protocol. The following problems happen if failures occur: the participant or coordinator can block in three states, waiting for an incoming message

- participant can block in state INIT, waiting forever for a vote_request message
- coordinator blocks in state wait, waiting for the votes of all participants, after expiry of timeout coordinator should abort and send a global_abort message.

- participant blocks in state ready, waiting for the global vote from the participant. If it does not receive a message, the participant cannot simply abort, but must try to find out what message has been sent. It contacts other nodes, to find out whether the other node received a commit message, or an abort message.

→ blocking commit protocol

- use "timeout" to unblock
- in state "ready" participant P can contact Q
  - if Q in in commit, coordinator died after sending commit to Q before sending commit to P ⇒ P can commit
  - if Q is in ABORT ⇒ ABORT
  - if Q is in INIT ⇒ ABORT
  - if Q ready ⇒ no decision, contact other participant

Blocking is resolved in 3 phase commit.

12.1 Three Phase Commit (week 12)

2 PC blocks sometimes, not very often and hence is normally used in practice
Three-phase-commit (Skeen, 1981)

- avoids blocking in the presence of fail-stop crashes
- again coordinator and participants

States satisfy the following conditions:

1. there is no state from which directly commit or abort follows
2. there is no sate in which is not possible to make a final decision and from which a transition to a commit state can be made.

⇒ necessary and sufficient conditions for non-blocking commit protocol

- blocking states:
  - participant in state init, waiting for a message from the coordinator, will eventually abort - identical to 2PC.
  - coordinator wait → abort, analogously
    on timeout, coordinator will decide that participant has crashed and will abort.
  - coordinator can block in state precommit. After expiry of timeout, it will assume that the participant has crashed, after voting for commit. The coordinator sends a commit message to all nodes, knowing that the crashed node will recover to prepare_commit and will then commig.
  - participant in ready or precommit, after timeout assume coordinator has failed, as in 2PC
    contact other participant, if all are in precommit ⇒ commit, if Q in init ⇒ abort
- participant can reach precommit only if coordinator was in precommit already

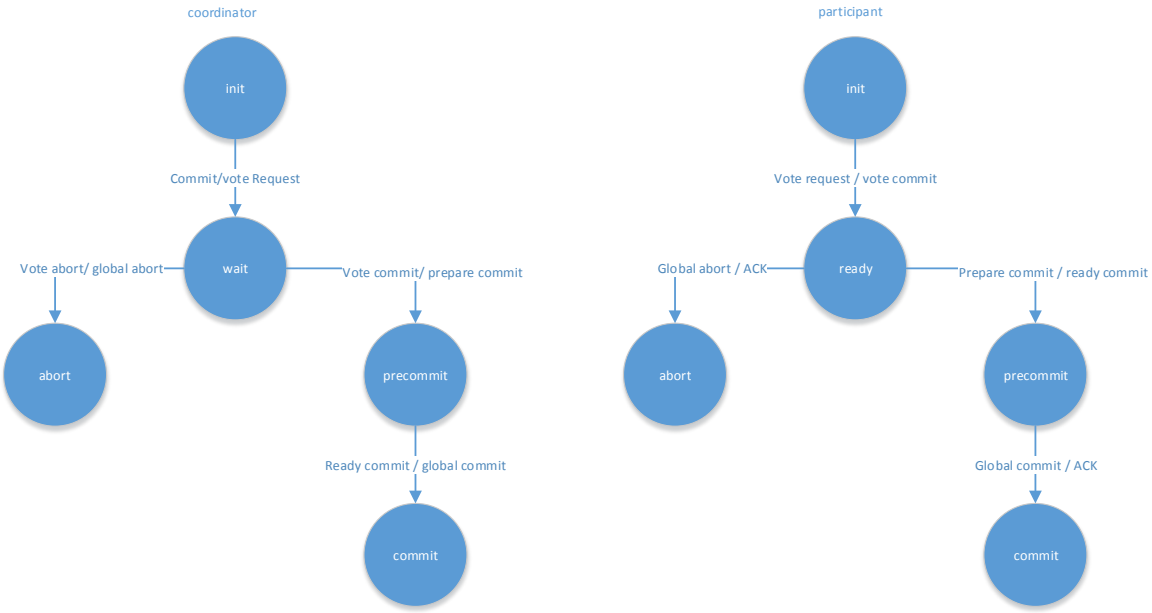


Figure 25: 3-Phase Commit protocol

– if each Q that P was contact is in ready and if this is the majority of participants ⇒ abort, this is the safe solution, as if is unknown to what state a failed node will recover. If the failed node recovers to init, abort will be the only correct decision. If the node recovers to precommit, still abort can not do any harm.

– in 2PC a crashed node can recover to commit, while all others are still in ready. This can not happen with 3PC. With 3PC if any process is in ready state, no crashed process will recover to a state other than init, abort, or precommit.

– 3PC is not provably correct.
⇒ Paxos [4] only uses majority votes

13 Snapshot algorithm (Chandy-Lamport)

– create global distributed state

algorithm

P_i

```
send marker msg to all other nodes P_j
record local state
record all messages on channel P_i - P_j
until marker msg is received from P_j
```

[4] https://en.wikipedia.org/wiki/Paxos_(computer_science)

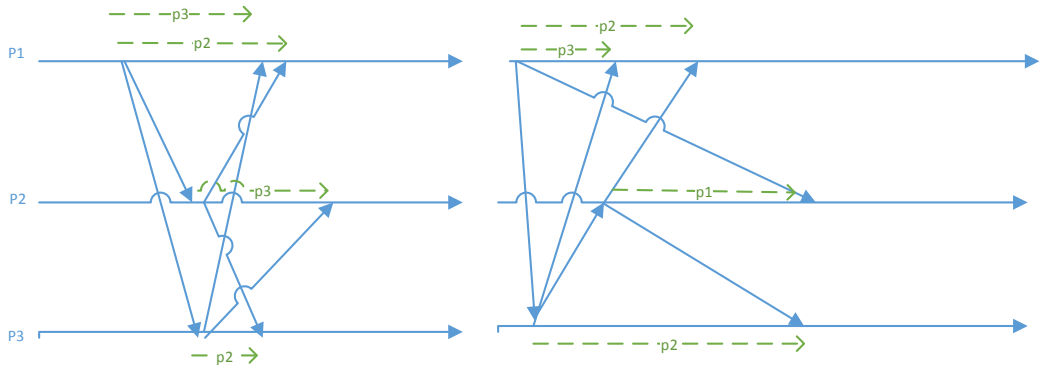


Figure 26: Snapshot algorithm

14 exam

- critical section, verteilte ressourcen leider nicht gemacht(algorithmen)
- viel gemacht happend before logische Zeit
- unterschiedliche uhren algorithmen , eigendschaften von systemem mit den uhren kommt dran
- reihe von algorithmen: vorwiegend Zeitachsen welche nachrichten werden verschickt zum ablauf des protokols, 4 verschiedenen aufgaben mit algorithmen leader election mutual exclusion, snapshot, distributed commit
- leader election ¿ viele algorithmen
- beispiel: gegeben zeitahse, zeiche nachrichten ein, wieviele braucht man, eigenet sich nciht für alle
- beschriebe pgegebenen pseudocode, oder man bekommt zeitdiagram und schreibe selbst pseudocode
- gegeben algorithmus und man lässt eine zeile weg-¿ ist folgende bedingung erfüllt?
- fingertabellen
- leader election : vergleiche 2 algorithmen: token based unterschiede zu timestamp
- algorithmen kennen? erstellten pseudocode muss man kennen, snapshot nicht
- snapshot: finde heraus, ob alle nachrichten geschickt wurden, oder ob unsinnige gesendet wurden. gibt es überflüssige Nachrichten? was ist, wenn es fehlende nachrichten gäbe?
- seite a4 beidseitig erlaubt
- pseudocode komplett auf den spicker? wenn verstanden wie , dann nein, die logik muss klar sein
- ratschläge: fragen sie die tutoren, schreibe erklärungen dazu