Distributed Systems

Exercise Session 1

Selma Steinhoff: selmas@ethz.ch

Disclaimer: these are not official slides, there might be mistakes, treat them this way

Reasons for Distributed Systems

- Geography:
  - Companies are geographically distributed
- Parallelism:
  - Employ multicore processors or computing clusters
- Reliability:
  - Data is replicated to prevent data loss
- Availability:
  - Allow for access at any time, without bottlenecks

Fundamental Goal in Distributed Systems

- One fundamental goal: *state replication* (Definition 7.8)
  - All nodes execute the sequence of commands in the same order

- Approach to state replication (Algorithm 7.3):
  - Client sends one command at a time

Fault-Tolerance in Distributed Systems

- Various problems can occur in practice:
    1. Nodes (= single actor in system) may crash (Chap. 6)
    2. Messages may be lost (Model 7.4)

Approach to State Replication

- Server sends *acknowledgement (ACK)* message (Algorithm 7.6)
  - Client resends command after *timeout*
  - Problems?

Approach to State Replication

- Server sends *acknowledgement (ACK)* message (Algorithm 7.6)
  - Client resends command after *timeout*
  - *Sequence numbers* to identify duplicates

Approach to State Replication

- Server sends *acknowledgement (ACK)* message (Algorithm 7.6)
  - Client resends command after *timeout*
  - *Sequence numbers* to identify duplicates

  - Multiple server?
  - Multiple Clients?

Approach to State Replication

- Server sends *acknowledgement (ACK)* message (Algorithm 7.6)
  - Client resends command after *timeout*
  - *Sequence numbers* to identify duplicates

  - Multiple server ✓
  - Multiple Clients ✗ (Theorem 7.7)

Fault-Tolerance in Distributed Systems

- Various problems can occur in practice:
  1. Nodes (= single actor in system) may crash (Chap. 6)
  2. Messages may be lost (Model 7.4)
  3. Variable message transmission times (Model 7.6)

Approach to State Replication

- Choose one server as a Serializer (Algorithm 7.9)
  - Single point of failure

Approach to State Replication

- Two-Phase Protocol and variants (Algorithm 7.10)
  - How to handle server crashes?
  - How to avoid deadlock with locks?

Paxos – main ideas

- Servers hand out *tickets* = ”weak locks” (Definition 7.11)
  - *Reissuable*: Server can issue ticket, even if previous tickets haven’t been returned
  - *Ticket expiration*: Server will only accept ticket, if it is the most recently issued one

Paxos – main ideas

- Only requires the **majority of servers** to agree
    - Already ensures that there is at most one accepted command
- Servers **notify** clients about their stored command
    - Client can then switch to supporting this stored command

Algorithm 7.13 Paxos

Client (Proposer)	**Server (Acceptor)**

Initialization ....

c ◁ *command to execute*
$t = 0$ ◁ *ticket number to try*

$T_{\max} = 0$ ◁ *largest issued ticket*

$C = \bot$ ◁ *stored command*
$T_{\text{store}} = 0$ ◁ *ticket used to store C*

Phase 1 ....

Clients asks for a specific ticket *t*

1: $t = t + 1$
2: Ask all servers for ticket t

3: **if** $t > T_{\max}$ **then**
4: $T_{\max} = t$
5: Answer with ok(T_{store}, C)
6: **end if**

Server only issues ticket *t* if *t* is the largest ticket requested so far

If client receives majority of tickets, it proposes a command

if a majority answers ok **then**
  Pick (T_{store}, C) with largest T_{store}
  if $T_{\text{store}} > 0$ **then**
    $c = C$
  end if
  Send propose(t, c) to same majority
end if

When a server receives a proposal, if the ticket of the client is still valid, the server stores the command and notifies the client

if $t = T_{\max}$ **then**
  $C = c$
  $T_{\text{store}} = t$
  Answer success
end if

Phase 3 . . . .

if a majority answers success **then**
  Send execute(c) to every server
end if

If a majority of servers store the command, the client notifies all servers to execute the command

Quiz: Paxos

- **How does a node in Paxos know if a majority answered with ok?**
  - All nodes must know how many servers are in the system
- **Does the Paxos algorithm in the script achieve state replication?**
  - No it only shows one instance, for subsequent commands it would need to be restarted
- **Does Paxos achieve consensus?**
  - No, doesn’t terminate in all cases
- **How many nodes could crash so that Paxos still works?**
  - Less than half

Consensus

We want:

- **Agreement:** all (correct) nodes decide for the same value
- **Termination:** all (correct) nodes terminate
- **Validity:** the decision value is the input value of at least one node

Consensus

Impossibility:

- Consensus cannot be solved ***deterministically*** in the asynchronous model with $f \leq n/2$
- ***Probabilistic*** solution?

Randomized Consensus

Easy cases:

- All inputs are equal (all 0 or 1)
- Almost all input values equal

Otherwise:

- Choose a ***random*** value locally. → expected time O(2^n) until all agree (once)

Algorithm 8.15 Randomized Consensus (Ben-Or)

```
v_i ∈ {0,1}          ◁ input bit
round = 1
decided = false

Broadcast myValue(v_i, round)

while true do
  Propose
  Wait until a majority of myValue messages of current round arrived
  if all messages contain the same value v then
    Broadcast propose(v, round)
  else
    Broadcast propose(⊥, round)
  end if

  if decided then
    Broadcast myValue(v_i, round+1)
    Decide for v_i and terminate
  end if
  Adapt
  Wait until a majority of propose messages of current round arrived
  if all messages propose the same value v then
    v_i = v
    decided = true
  else if there is at least one proposal for v then
    v_i = v
  else
    Choose v_i randomly, with Pr[v_i = 0] = Pr[v_i = 1] = 1/2
  end if
  round = round + 1
  Broadcast myValue(v_i, round)
end while
```

Quiz: Randomized Consensus

- **Is it possible to hear propose messages for different values?**
  - No, as a majority of the same value needs to be observed beforehand
- **Why wait for a majority of propose values before setting decide to true?**
  - Otherwise you can get stuck in a scenario, where only one node has decided and terminates while the others are still deciding
- **Does validity still hold while having randomness?**
  - The coin will not be tossed if all nodes start with the same value
- **What is the biggest drawback of the algorithm?**
  - The run time until expected termination is large

Randomized Consensus

Easy cases:

- All inputs are equal (all 0 or 1)
- Almost all input values equal

Otherwise:

- Choose a ***random*** value locally. → expected time $O(2^n)$ until all agree (once)
- Wouldn't it be useful if the nodes could all toss the ***same*** coin? → Shared Coin.