Consensus/Agreement in Faulty Systems

95

Common Knowledge

There is *common knowledge* of *p* in a group of agents *G* when all the agents in *G* know *p*, they all know that they know *p*, they all know that they all know that they know *p*, and so on *ad infinitum*

96

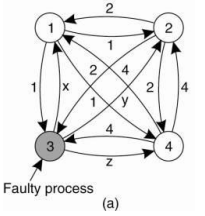
When is Agreement possible?

- The two-armies problem (attaining common knowledge)
  - Reliable armies, unreliable communication
  - ?

97

When is Agreement possible?

- The Byzantine generals problem
  - m unreliable generals (arbitrary failures)



1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

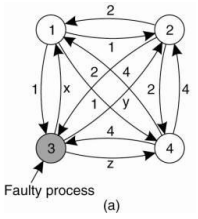
In general, need 3m+1 nodes to tolerate m failures

Tanenbaum and van Steen Figure 7-4: Lamport's solution

98

Byzantine Generals Problem

- Achieving fault tolerance in a group with m arbitrary failures



1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, 3, 4)
4 Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

In general, need 3m+1 nodes to tolerate m failures

Tanenbaum and van Steen Figure 7-4: Lamport's solution

99

Process Resilience

- For a process providing services, we want it to be fault-tolerant
  - Assumption: fail-stop/fail-silent processes
  - Replicate service with several identical processes in a distributed group
  - An abstraction as a single process to outside clients
  - When a message is sent to a group, all members of the group receive it
  - Goal: all servers eventually reach the same state

100

Paxos: Services and roles

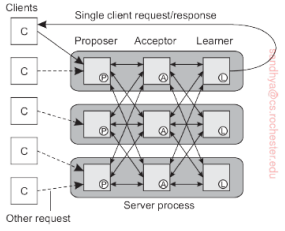


Figure 8.6: The organization of Paxos into different logical processes.

101

Paxos: The role of the leader

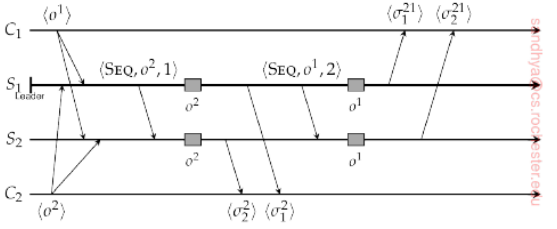
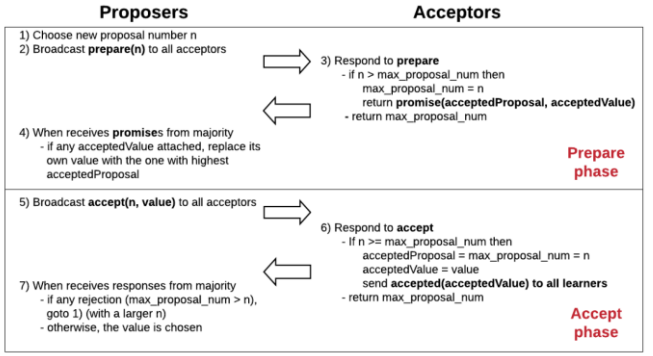


Figure 8.7: Two clients communicating with a 2-server process group.

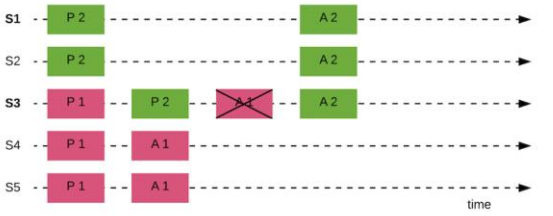
102

Paxos communication phases



- acceptedProposal, max_proposal_num, acceptedValue must be stored on disks

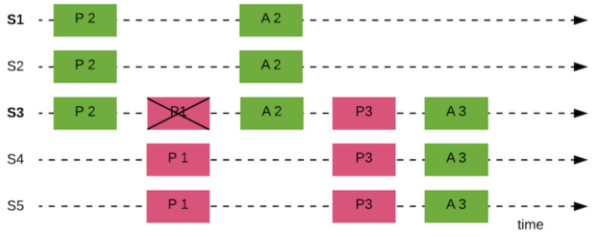
103



- The Color indicates value, bold indicates proposers, P=sends a **prepare** & receives a **promise**, A=sends an **accept** & receives an **accepted**
- When an acceptor **promise** to a **prepare** with proposal number n, it would reject **accept** with lower proposal number

Courtesy Ziliang Lin

104



- When an acceptor **promise** to a **prepare** with proposal number n, it would also reject **prepare** with lower proposal number

Courtesy Ziliang Lin

105

CAP Theorem

Any networked system providing shared data can provide only two of the following three properties:

- C: consistency, by which a shared and replicated data item appears as a single, up-to-date copy
- A: availability, by which updates will always be eventually executed
- P: Tolerant to the partitioning of process group (e.g., because of a failing network).

In a network subject to communication failures, it is impossible to realize an atomic read/write shared memory that guarantees a response to every request [Gilbert and Lynch, 2012].

106

Types of Replicas

- Replica creation and placement:
  - Permanent
  - Server-initiated
  - Client-initiated

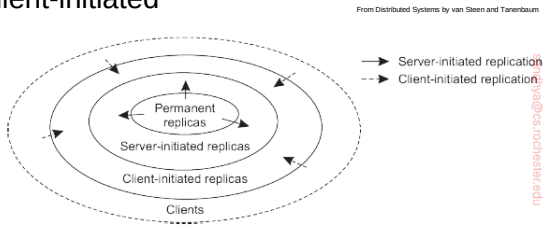


Figure 7.21: The logical organization of different kinds of copies of a data store into three concentric rings.

Keeping Replicas Consistent

- Primary-based
- Quorum-based

Quorum-Based Replication Protocols

- Replicate file on N servers
  - For update: contact majority (N/2+1) for agreement
  - For read: contact majority once again, read is successful if they have the same version
  - In general
    - $N_R+N_W > N$
    - $N_W > N/2$

Update Propagation

- What is propagated?
  - Invalidation
  - Update
  - active replication (move the computation)
- When is it propagated?
  - Pull versus push
  - Leases
  - Epidemic protocols

When is it Propagated?

- Push vs. Pull

Issue	Push-based	Pull-based
State at server	List of client replicas and caches	None
Messages sent	Update (and possibly fetch update later)	Poll and update
Response time at client	Immediate (or fetch-update time)	Fetch-update time

Figure 7.23: A comparison between push-based and pull-based protocols in the case of multiple-client, single-server systems.

From Distributed Systems by van Steen and Tanenbaum

- Leases – a compromise [Gray and Cheriton SOSP'89]
  - Adaptive leases
    - Age-based (lower frequency of writes→ higher lease)
    - Client request frequency based (higher freq → higher lease)
    - Server state space/load based (higher space → lower lease)

111

Epidemic protocols

- Infective: server that holds updates and is willing to spread it
- Susceptible: server that has not been updated yet
- Removed: server that is not willing or not able to spread its update

112

112

Epidemic protocols

- Try to "infect" all members in the group with new updates as fast as possible

113

113

Anti-entropy

- A server P picks up a server Q at random
  - P only *pushes* its own updates to Q
    - Spreads slowly
  - P only *pulls* in new updates from Q
    - Works better when most servers are infective
  - P and Q send updates to each other (*push-pull*)

114

114

Gossiping

- If P is recently updated with data item x, it will contact an arbitrary server Q and try to push the updates to Q
- If Q has already received the update, P will lose interest in spreading it further with some probability
  - No guarantee that all servers will be updated

115

Topics to Come

- Group Communication
- Distributed transactions
- Distributed file systems
- More on GPUs
- Nonblocking data structures/algorithms
- Transactional memory, time permitting