Fault Tolerance and Consensus

1

Copyright

- Le slide di questo corso sono in parte personalizzate da quelle fornite come materiale associato ai testi consigliati (Distributed Systems: Concepts and Design, 5/e, Coulouris, Dollimore, Kindberg & Blair, Addison-Wesley, e Distributed Systems: Principles and Paradigms, A. S. Tanenbaum, M. Van Steen, Prentice Hall, 2007). Parte delle slide sono invece di proprietà del docente. Tutte le slide sono soggette a diritto d'autore e quindi non possono essere ri-distribuite senza consenso. Lo stesso vale per eventuali registrazioni o videoregistrazioni delle lezioni.
- The slides for this course are partly adapted from the ones distributed by the publisher of the suggested books for this course (Distributed Systems: Concepts and Design, 5/e, Coulouris, Dollimore, Kindberg & Blair, Addison-Wesley and Distributed Systems: Principles and Paradigms, A. S. Tanenbaum, M. Van Steen, Prentice Hall, 2007). Other slides are from the teacher of this course. All the material is subject to copyright and cannot be redistributed without consent of the copyright holder. The same holds for audio and video-recordings of the classes of this course.

2

Fault Tolerance Basic Concepts

- Being fault tolerant is strongly related to what are called dependable systems
- Dependability implies the following:
  1. Availability. The probability that the system operates correctly at any given moment
  2. Reliability. The ability to run correctly for a long interval of time
  3. Safety. Failure to operate correctly does not lead to catastrophic failures
  4. Maintainability. The ability to "easily" repair a failed system

Failure Models

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

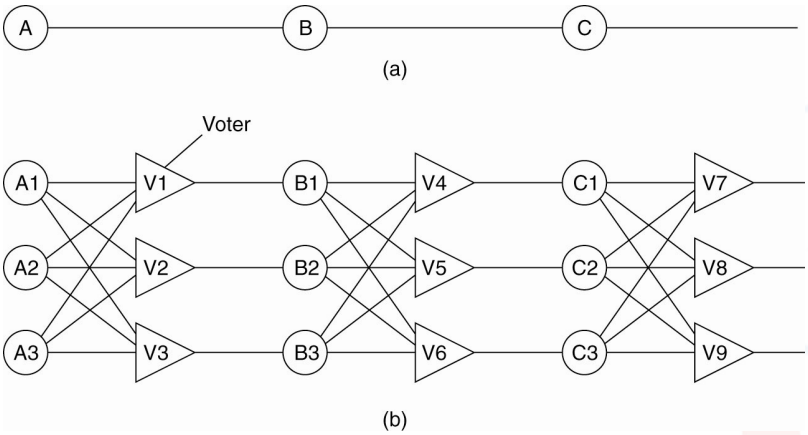
Byzantine failure

Different types of failures.

Failure Masking by Redundancy

- Information redundancy
  - Example: extra bits used to recover transmitted message when noise is present
- Time redundancy
  - Example: a transaction is repeated if aborted because of failure of one of its actions
- Physical redundancy
  - Extra hardware or software (processes) used to recover from failure of some component

Failure Masking by Redundancy



How many faulty devices are tolerated?

Process resilience

- Redundancy using groups of identical processes.
- Each process receives the messages for the group
- A protocol and mechanism for joining and leaving a group is necessary
- How much redundancy is needed?

How much redundancy?

- If faulty processes just stop working (crash failure), k+1 processes provide k-fault tolerance (client gets the same value from all the surviving processes).
- If faulty processes reply with wrong values, at least 2k+1 processes are needed (client can decide by voting)
- If a consensus in the group is needed (e.g., to decide mutual exclusion, election, …) the problem is harder and proved to be impossible in some cases

Agreement problems

- Consensus
  - each process proposes a single value and the non-faulty processes should agree on the same value
- Byzantine generals problem
  - one process (commander) proposes a value *v*. All the correct processes must agree on a value. If the commander is non-faulty they agree on *v*
- Interactive consistency
  - each process proposes a value. All the correct processes agree on a vector of values where each value comes

A solution to one of the problems facilitates a solution to the others

Every Ware Lab

10

Result on Byzantine agreement with synchronous systems

- **At least N=3k+1 processes are needed for k faulty processes**

  [Pease, M., Shostak, R. and Lamport, L. (1980). Reaching agreement in the presence of faults. Journal of the ACM]

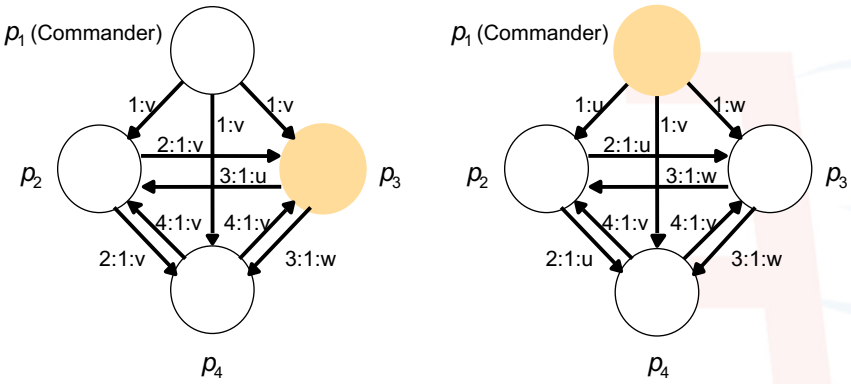
Example of solution for Byzantine generals problem for N=4 and k=1

- the commander sends a value to each of the lieutenants.
- each of the lieutenants sends the value it received to its peers (more rounds are needed for N>4)
- A lieutenant receives a value from the commander, plus N – 2 values from its peers. If the commander is faulty, then all the lieutenants are non-faulty and each will have gathered exactly the set of values that the commander sent out. Otherwise, one of the lieutenants is faulty; each of its correct peers receives N – 2 copies of the value that the commander sent, plus a value that the faulty lieutenant sent to it. In either case, the correct lieutenants need only apply a simple majority function to the set of values they receive. When no majority exists a special value is used.

Every Ware Lab

11

The algorithm applied in two cases



Faulty processes are shown coloured

Synchronous versus asynchronous systems

- Synchronous systems:
  - Execution on each node is bounded in speed and time
  - Communication links have bounded transmission delay
  - Clock on each node has a bounded drift
- Asynchronous systems:
  - Execution on each node can occur at arbitrary speed
  - Communication links have different and unbounded transmission delay
  - Clock on each node has an unbounded drift

Synchronous versus asynchronous systems

- Internet-based heterogeneous distributed systems are inherently asynchronous
- Coordination and agreement in asynchronous systems is hard and often impossible. We often make (partial) synchronicity assumptions
- In order to cooperate in reaching a common goal we need algorithms that achieve a form of synchronization

Impossibility of agreement in asynchronous systems

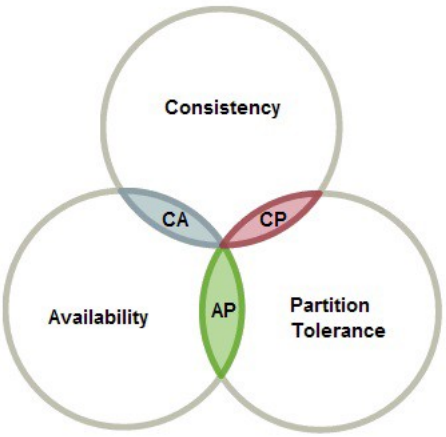
- FLP Theorem (1985). When delays in answering messages are arbitrary there is no guaranteed solution to Byzantine agreement (consensus in presence of Byzantine faults) neither to totally ordered multicast, even if a single node fails.
- There are solutions for partially synchronous systems that can be used to model practical systems.

The CAP theorem for asynchronous systems: only 2 of these are achievable in r/w storage

- *Consistency*: Every read receives the most recent write or an error (implies all nodes see the same data)
- *Availability*: Every request receives a response (system operational at any given time)
- *Partition tolerance*: The system tolerates an arbitrary number of messages lost (or delayed)

Every Ware Lab

The CAP theorem



Real systems relax availability or consistency guarantees (or both)

https://users.ece.cmu.edu/~adrian/731-sp04/readings/GL-cap.pdf

Every Ware Lab

Practical consensus: the Paxos protocol

- introduced by Lamport in 1989. Many versions now exists
- Goal: solving consensus in a network of unreliable or fallible processors ensuring *consistency*
- the basic version does not cover Byzantine failures (but there is Byzantine Paxos). It tolerates k failing nodes with N=2k+1
- there are conditions under which it won't make progress but very unlikely to occur
- used by Google in their Chubby distributed lock service to keep replicas consistent in case of failure. Also used in Google Spanner, Apache Cassandra and many other DS

Practical consensus: the Raft protocol

- introduced by Ongaro and Ousterhout at Stanford in 2013 as a more understandable and easier to implement version of Paxos
- same goal: a solution to state machine distribution preserving safety (consistency)
- based on leader election and log replication
- it does not cover Byzantine failures (the nodes trust the elected leader)
- It tolerates k failing nodes with N=2k+1
- more at https://raft.github.io/

Other important topics

- Reliable client-server communication
  - Example: RPC in presence of failures
- Reliable group communication
  - Example: Atomic multicast
- Distributed commit
  - Generalization of atomic multicast to arbitrary operations performed by members of a group
    (all-or-none semantics)

References

- Textbook (Coulouris et al.) chapter 15
- The CAP theorem. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", S. Gilbert and N. Lynch, ACM SIGACT News, (2002). DOI:10.1145/564585.564601
- "The Byzantine generals problem". L. Lamport et al., ACM Trans. on Programming Languages and Systems, (1982). https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf
- FLP Consensus impossibility result https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf
- A great video tutorial on consensus algorithms (Amr El Abbadi ICDE 2020): https://youtu.be/QXdVW3Tze_c

Homework

- Watch a Google tech talk on the widely used Paxos consensus algorithm https://youtu.be/d7nAGI_NZPk
- Go through the animated explanation of the Raft consensus algorithm http://thesecretlivesofdata.com/raft/