Coordinating Distributed Systems

Theory and practice

Daniele Venzano

Eurecom

Introduction

Outline

- **What is a distributed system?**
- **The consensus problem**
  - A few examples of distributed consensus
  - CAP theorem
  - Eventually consistent Vs Strongly consistent
  - Fault tolerance: possible faults in distributed systems
- **Consensus protocols**
  - Two phase commit
  - Paxos overview
  - Raft from A to Z
- **Implementations - ZooKeeper**
  - History
  - Architecture
  - Data model
  - Higher-level primitives

What is a distributed system?

- A set of processes seeking to achieve some common goal by communicating with each other

What is a distributed system?

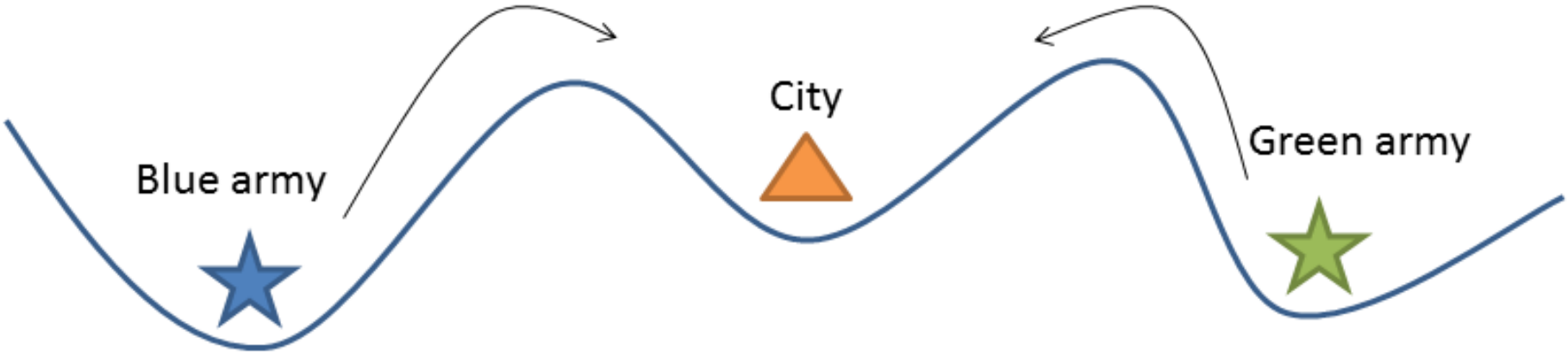
- Like a software matrioshka
  - Multi-threaded process
  - Multi-process on a single server
  - **Multiple processes in a set of servers in the same datacenter**
  - **Multiple processes in a set of geographically distributed servers**
- Why?
  - Inherent distribution (sensors, peer-to-peer, publish-subscribe, ...)
  - Engineering choice (fault tolerance, replication, performance, ...)
- These processes need to coordinate to reach a common goal:
  - data aggregation
  - synchronization
  - transactions
  - ...

The consensus problem

Wedding consensus

- The priest follows a well known protocol to reach a consensus:
  1. Priest: Alice, will you marry Bob ?
  2. Alice: yes
  3. Priest: Bob, will you marry Alice ?
  4. Bob: yes
  5. Priest: You are now husband and wife
- In distributed systems this becomes:
  1. Coordinator: Alice, can you commit key X with value 5 ?
  2. Alice: yes, I can
  3. Coordinator: Bob, can you commit key X with value 5 ?
  4. Bob: yes, I can
  5. Coordinator: Ok, both of you record that X has now a value of 5
- What if Bob flees from the church?

The two generals



- Two generals want to attack a city
- They can only use unreliable messengers to communicate
- They need to attack at the same time to succeed

An infinite number of messages is needed for each general to be sure the other agrees on the time of the attack.

Consensus examples

Processes need to reach a common goal:

- aggregate functions: sensors calculating average temperature
- synchronization: agree on a value, elect a leader
- reliable broadcast: a message sent to a group is received by all or none
- atomic commit: ensure that processes reach a common decision whether to commit or abort a transaction
- leader election: ensure there is only one process in charge at a given time

Properties of a distributed system

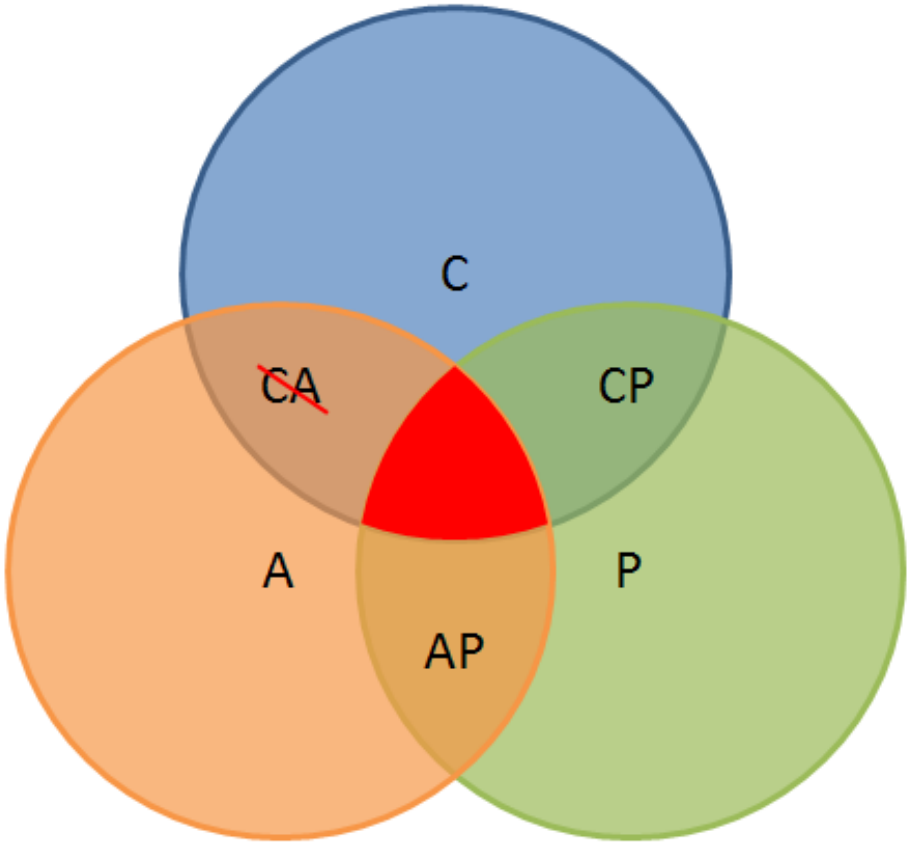
- Linearizability[1]: a given set of operations is *linearizable* if it appears to the rest of the system to occur instantaneously
  - Writes are linearizable operations if every read receives the most recent write or an error
- Availability: a system is *available* if every request to a non-failing node always receives a response, eventually
  - Reading stale data is ok, though
- Partition tolerance
  - The system continues to function properly even if the network loses or delays an arbitrary number of messages

The CAP theorem links these three properties.

[1] The CAP theorem calls Consistency what is actually Linearizability

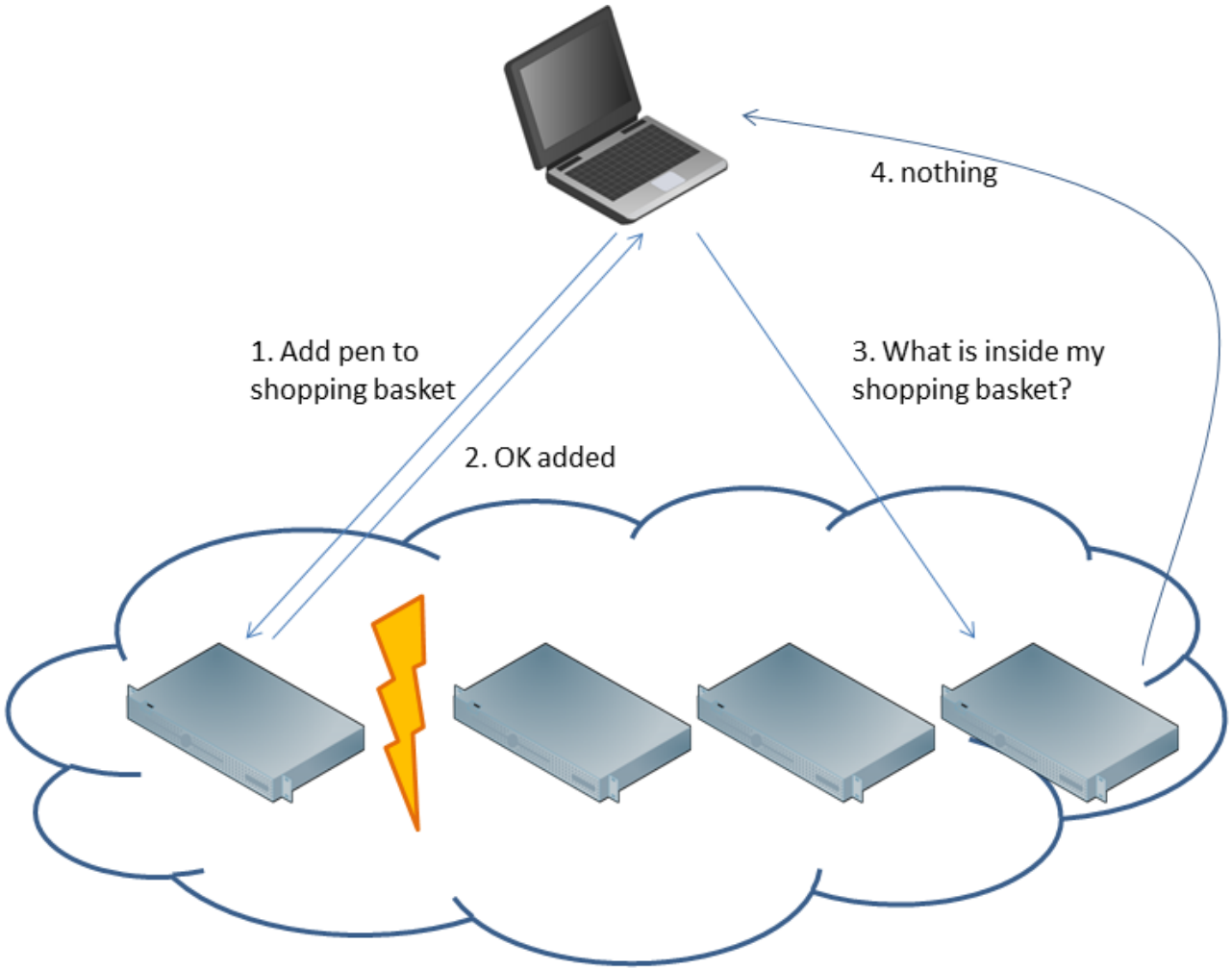
CAP theorem - 1

The theorem says: between C A P, you can choose only two



(... but you need the P ...)

CAP theorem - 2 - Why not all three?



If there is a network partition either C or A will break

CAP theorem - 3 - P

- Network partitions occur outside anyone's control in real life
- Cannot sacrifice the Partition-Tolerance property
- In the event of a network partition either A or C is maintained: it is the choice of the designer
- Practical distributed systems are CP or AP
- Some can be configured to shift between CP and AP (tunable consistency)

Note: a network partition can also be a very slow link

CAP theorem - 4 - Summary

- First stated by Eric Brewer (Berkeley) at the PODC 2000 keynote
- Formally proved by Gilbert and Lynch, 2002[2]

- The CAP theorem formally states the trade-offs among different distributed systems properties
- In practice network Partitions occur, the designer can choose one of Consistency or Availability
- The choice heavily depends on what your application/business logic is

CAP theorem - 5 - Summary

CP-oriented systems:

- BigTable, Hbase, MongoDB, Redis, MemCacheDB, Scalaris, Paxos, ZooKeeper[1]

AP-oriented systems:

- Amazon Dynamo, CouchDB, Cassandra[2], SimpleDB, Riak, Voldemort

In-depth articles on the misuse of the CAP theorem in describing real systems:

- `https://codahale.com/you-cant-sacrifice-partition-tolerance/`
- `https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html`

[2] CA or CP tunable

Consistency models

- Eventual consistency: after a write, every read from the distributed system will *eventually* return the written value, if no new updates are made to it
- Strong consistency: after a write, all reads from the distributed system will return either the old or the new value

Fault tolerance in distributed systems

Faults examples:

- Simple: network partitions, hardware or software crashes, outdated/malicious nodes (bizantine faults)
- Complex: feedback loops that overcompensate in good faith (examples in next slides)

Faults will always happen, design tolerance mechanisms:

- Replication: master-slave ($\rightarrow$ failover) or load balancing
- Isolation: malfunctioning components do not affect the system as a whole

Amazon DynamoDB crash (2015/09/20)

DynamoDB: distributed NoSQL database from Amazon web services (AWS)

1. Network disruption caused timeouts of some storage servers
2. The affected servers tried to re-establish their membership
3. A change of usage pattern caused membership data to be unexpectedly large
4. Membership servers started to overload
5. More storage servers timed-out on health checks to membership servers
6. Cascading failure, stabilized at 55% error rate for customers

Post-mortem: `https://aws.amazon.com/message/5467D2/`

More real-world failures

- Google: https://status.cloud.google.com/summary
- Facebook: https://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919
- Apple: http://appleinsider.com/articles/16/06/02/apples-app-stores-apple-tv-itunes-other-services-h
- Microsoft (Azure): https://azure.microsoft.com/en-us/status/history/

Consensus protocols

Consensus protocols overview

Simplest (but unsafe):

- Two-phase commit

Traditionally studied for modern distributed systems:

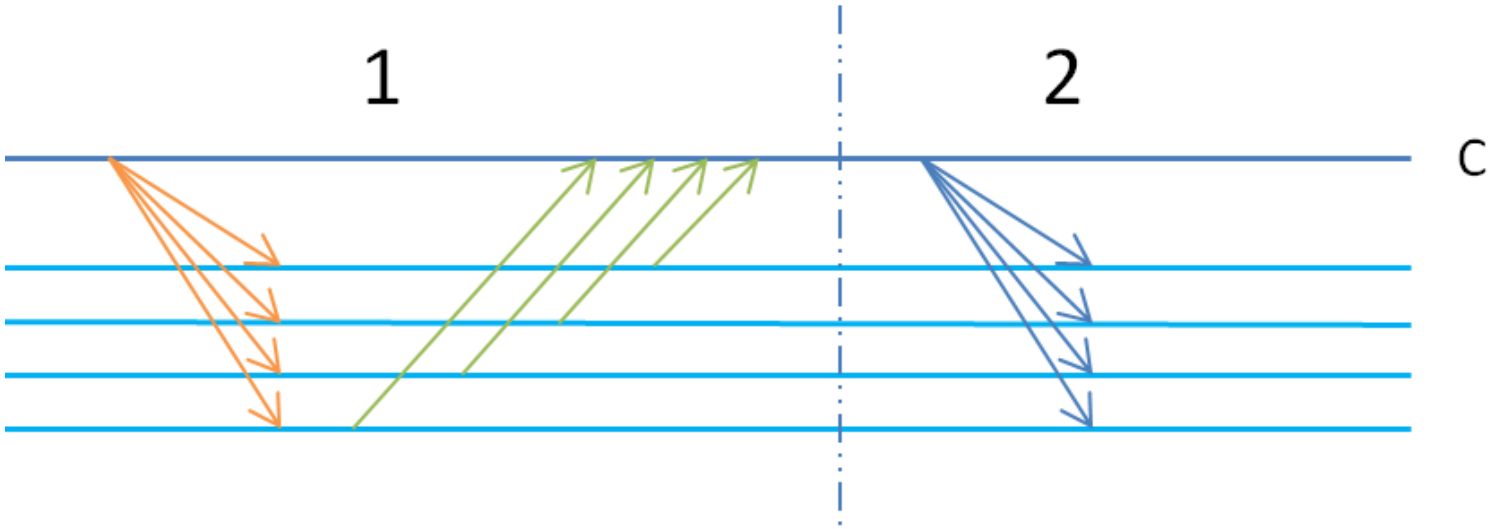
- Paxos
- Raft
- ZAB

Other:

- Lock-step (used in some multiplayer video games)
- The proof-of-work from Bitcoin

Two-phase commit

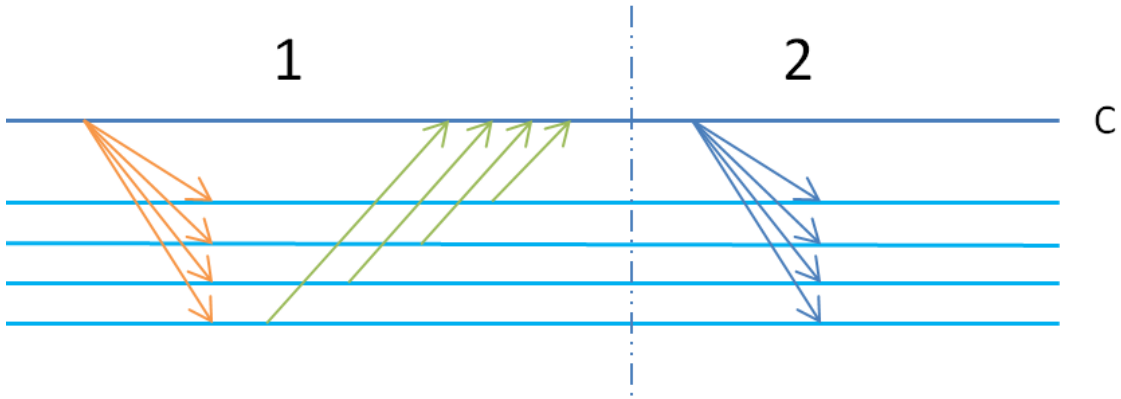
Simplest protocol for consensus



- Phase 1
  - One coordinator node *C* suggests a value to the other nodes
  - C gathers the responses
- Phase 2
  - If all nodes agree, C sends a commit command, abort otherwise

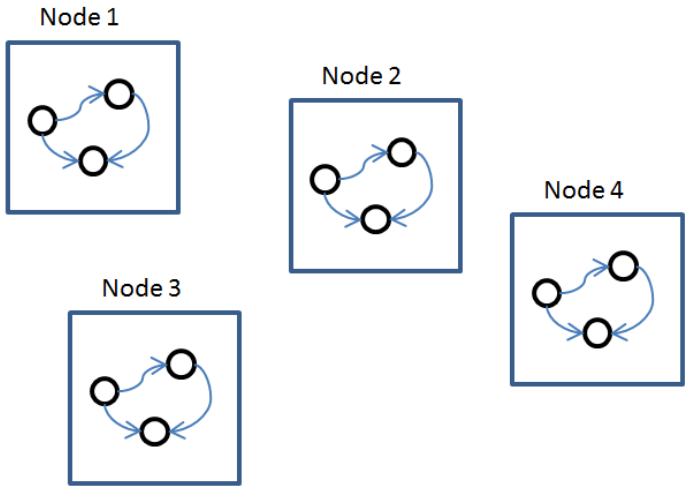
Two-phase commit - Why it is not enough?

Two phase commit cannot make progress if there are simple failures.[7]



- If C fails, nothing can be done until it is restarted
  - If C fails in phase 1, it has to abort the commit and retry
  - If C fails in phase 2, it has to replay the outcome (commit/abort)
  - In both cases the outcome is uncertain until C restarts
- If just one of the other nodes crashes, is slow or unreachable, the value cannot be committed.

State-machine replication



- Paxos and Raft use distributed state machines
- State machines are fully deterministic
- Committed operations are executed by all state machines in the cluster in the same order

Example operation: set variable *x* to value 5

Paxos

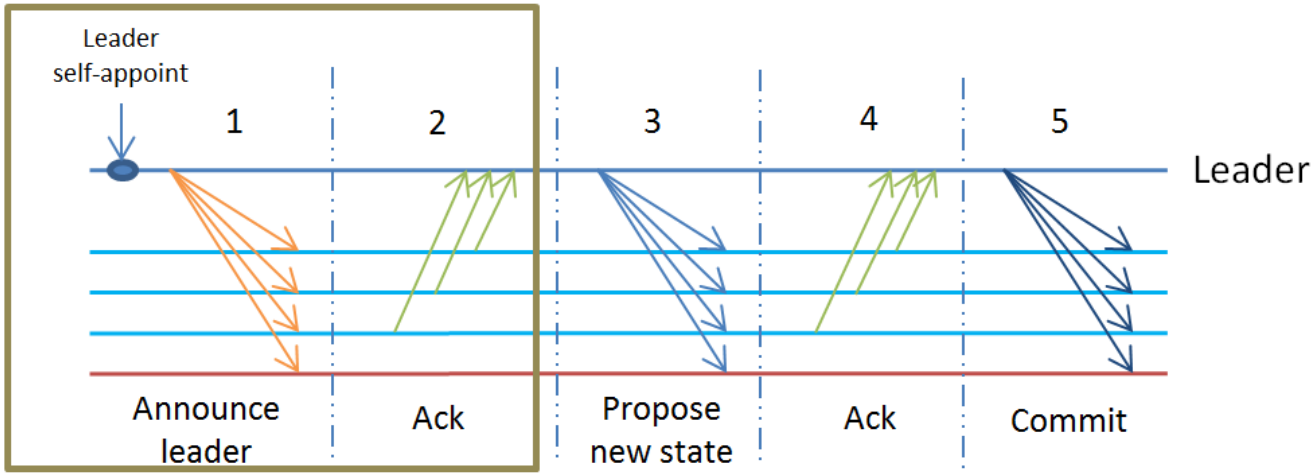
Old and well-known family of protocols for distributed consensus.

- first presented in 1989, paper published in 1998[3]
- has been formally proven to be safe

Complex and difficult to implement

- Basic Paxos protocol decides on a single output value
- Multi-Paxos extends the Basic protocol for practical use
- Many variants: cheap, fast, generalized, byzantine
- No reference implementation
- Many papers try to make it more approachable[5, 4, 1]
- Paxos needs a large enough subnet to be non-faulty for a long enough time, otherwise it might never terminate

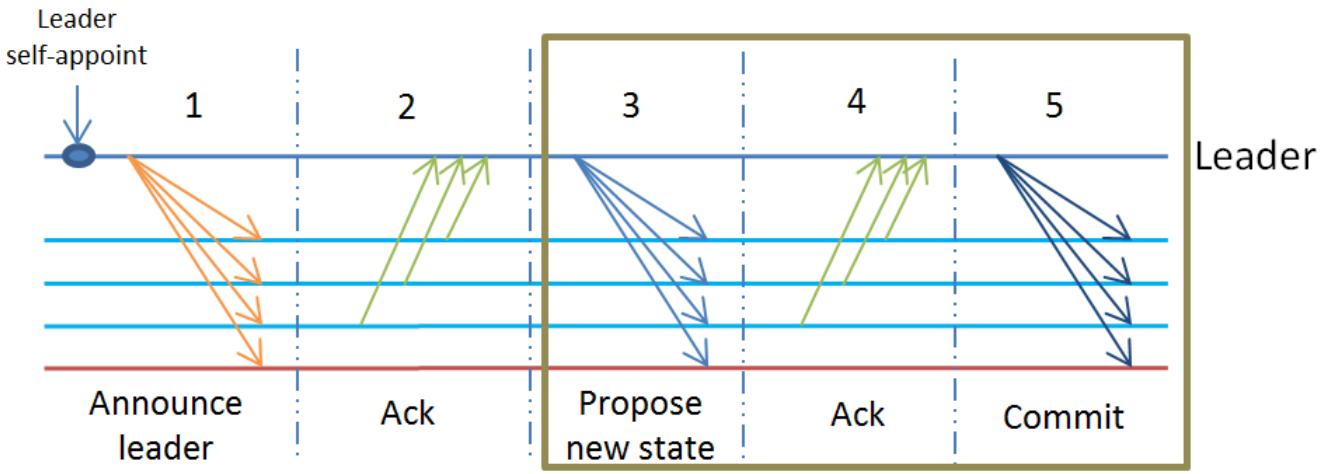
A Paxos round - 1



Phase 1 of a very simplified Paxos round:

- A node in the cluster self-appoints leader and chooses a new ballot ID
- Sends a ballot proposal to the other nodes (1)
- The other nodes return the highest ballot ID they know (2)
- If a majority responds with the proposed ID, the leader is confirmed for this this round

A Paxos round - 2



Phase 2 of a very simplified Paxos round:

- The leader proposes a new value (or a state transition) (3)
- The leader receives the answers (4)
- If a majority accepts the new value, the leader orders a commit (5)
- Otherwise a new round must be started

Raft - introduction

Modern implementation of a consensus protocol

- Published at Usenix 2014[6]
- Has been built to be easy to understand and implement

Random facts:

- The authors released a reference C++ implementation (LogCabin)
- Many implementations in other languages are available
- Lots of educational material, lectures and assignments are available

Raft - introduction

- Raft replicates entries in a distributed log
- Each log entry is a state machine command
- Once an entry has been committed, it is executed by each state machine
- Raft ensures global ordering on the committed log entries
- $\Rightarrow$ all state machines are synchronized on the last committed command
- Snapshotting is used to keep the log size limited

Raft - learning tools

1. Raftscope: `http://bigfoot-m2.eurecom.fr/raftscope/`
2. Secret lived of Data: `http://thesecretlivesofdata.com/raft/`

Some implementations

- ZooKeeper (ZAB)
- Consul (Raft + Serf)
- etcd (Raft)
- OpenReplica/ConCoord (Paxos)

ZooKeeper

History

Architecture

Data model

API

Laboratory session - leader election

Context:

- Sensors are streaming data to your cluster
- Need a central node to do aggregations, the system must be highly available
- Implement the leader election algorithm on top of ZooKeeper

Objective:

1. Start three processes
2. One of them will become the active leader
3. Stop/crash the leader
4. One of the surviving processes automatically takes the lead
5. The crashed node restarts without disrupting the current leadership (bonus)

References I

[1] Chandra, T. D., Griesemer, R., and Redstone, J.
Paxos made live: An engineering perspective.
In *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing* (New York, NY, USA, 2007), PODC '07, ACM, pp. 398–407.

[2] Gilbert, S., and Lynch, N.
Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.
SIGACT News 33, 2 (June 2002), 51–59.

[3] Lamport, L.
The part-time parliament.
ACM Trans. Comput. Syst. 16, 2 (May 1998), 133–169.

References II

[4] Lamport, L.
Paxos made simple.
SIGACT News 32, 4 (Dec. 2001), 51–58.

[5] Lampson, B. W.
How to build a highly available system using consensus.
http://research.microsoft.com/en-us/um/people/blampson/58-Consensus/WebPage.html.

[6] Ongaro, D., and Ousterhout, J.
In search of an understandable consensus algorithm.
In *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (Philadelphia, PA, June 2014), USENIX Association, pp. 305–319.

References III

[7] Robinson, H.
Consensus protocols: Two-phase commit.
http://the-paper-trail.org/blog/consensus-protocols-two-phase-commit/.