# Distributed Consensus for Dummies The Raft Protocol

Arnaud Bailly <abailly@murex.com>

2014-04

## How to Win Austerlitz?

## The Rules

- The emperor must coordinate its generals: If only some of its generals carry its orders, he loses everything!

## The Rules

- The emperor must coordinate its generals: If only some of its generals carry its orders, he loses everything!
- The emperor and its generals communicate with each other using *messengers* that carry orders

## The Rules

- The emperor must coordinate its generals: If only some of its generals carry its orders, he loses everything!
- The emperor and its generals communicate with each other using *messengers* that carry orders
- The emperor issues one order to any general, either **attack** or **defend**

## The Rules

- The emperor must coordinate its generals: If only some of its generals carry its orders, he loses everything!
- The emperor and its generals communicate with each other using *messengers* that carry orders
- The emperor issues one order to any general, either **attack** or **defend**
- The goal is to ensure they **all** have the same order when asked to act, ie. they reach **consensus**

## Let’s Try It!

## Possible Assumptions

There can be various assumptions on the way the generals and the emperor coordinates

- Messengers are *reliable*, ie. all messages are delivered and Generals are “perfect”

## Possible Assumptions

There can be various assumptions on the way the generals and the emperor coordinates

- Messengers are *reliable*, ie. all messages are delivered and Generals are “perfect”
- Messengers are *unreliable*: They can be killed, diverted, messages can arrive in wrong order...

## Possible Assumptions

There can be various assumptions on the way the generals and the emperor coordinates

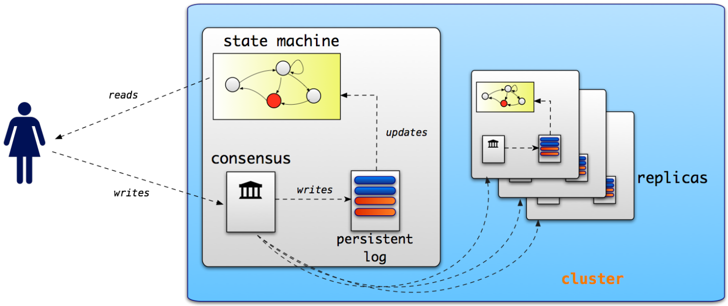
- Messengers are *reliable*, ie. all messages are delivered and Generals are "perfect"
- Messengers are *unreliable*: They can be killed, diverted, messages can arrive in wrong order...
- Generals can be killed and not respond anymore

## Possible Assumptions

There can be various assumptions on the way the generals and the emperor coordinates

- Messengers are *reliable*, ie. all messages are delivered and Generals are "perfect"
- Messengers are *unreliable*: They can be killed, diverted, messages can arrive in wrong order...
- Generals can be killed and not respond anymore
- There is a traitor!

## Basic Architecture



## Fundamental Impossibility

## In an Asynchronous Network...

*It is not possible to reach distributed consensus with arbitrary communication failures*
Distributed Algorithms, *Nancy Lynch, 1997, Morkan-Kaufmann*

## In a Partially Synchronous Network...

*It is possible to reach consensus assuming* f *processes fail and there is an upper bound* d *for all messages provided the number of processes is greater than* 2f
*Nancy Lynch, op.cit.*

## And in Practice?

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.

## Distributed Consensus is Hard...

The 8 Fallacies of Distributed Computing

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

## ... but We Need It

- Distributed transactions coordination

## … but We Need It

- Distributed transactions coordination
  - Several processes should agree on *commit* or *rollback* some operation

## ... but We Need It

- Distributed transactions coordination
  - Several processes should agree on *commit* or *rollback* some operation
- Distributed Fault-Tolerant data stores (eg. ZooKeeper, Spanner)

## … but We Need It

- Distributed transactions coordination
  - Several processes should agree on *commit* or *rollback* some operation
- Distributed Fault-Tolerant data stores (eg. ZooKeeper, Spanner)
- Distributed Locking (eg. Google's Chubby)

## Practical Consensus

## The Leader: Paxos

The Part-Time Parliament, *L.Lamport*

> *Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned de- spite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forget- fulness of their messengers.*

## Paxos Principles

- Core algorithm is called *Single-Decree Synod* and describes how a single proposed value is accepted by the distributed processes

## Paxos Principles

- Core algorithm is called *Single-Decree Synod* and describes how a single proposed value is accepted by the distributed processes
- Assumes non-Byzantine failures

## Paxos Principles

- Core algorithm is called *Single-Decree Synod* and describes how a single proposed value is accepted by the distributed processes
- Assumes non-Byzantine failures
- Extension to *multiple decrees* is supposed to be straightforward but...

## Paxos Principles

- Core algorithm is called *Single-Decree Synod* and describes how a single proposed value is accepted by the distributed processes
- Assumes non-Byzantine failures
- Extension to *multiple decrees* is supposed to be straightforward but...
- ... Lamport omits a lot of details!

## Paxos Implementation

*While Paxos can be described with a page of pseudo-code, our complete implementation contains several thousand lines of C++ code. Converting the algorithm into a practical, production-ready system involved implementing many features and optimizations – some published in the literature and some not.*

*Paxos Made Live - An Engineering Perspective, T.Chandra et al.*

## The Challenger: Raft

- In Search of an Understandable Consensus Algorithm, D.Ongaro and J.Osterhout, 2013

## The Challenger: Raft

- In Search of an Understandable Consensus Algorithm, D.Ongaro and J.Osterhout, 2013
- Novel algorithm designed with *understandability* in mind

## The Challenger: Raft

- In Search of an Understandable Consensus Algorithm, D.Ongaro and J.Osterhout, 2013
- Novel algorithm designed with *understandability* in mind
- Dozens of implementations in various language

## Principles of Operation

- *Leader-follower* based algorithm

## Principles of Operation

- *Leader-follower* based algorithm
- Each instance is a Replicated state machine whose states is uniquely determined by a linear *persistent log*

## Principles of Operation

- *Leader-follower* based algorithm
- Each instance is a Replicated state machine whose states is uniquely determined by a linear *persistent log*
- Leader election proceeds in *monotonically increasing terms* when timeout fires

## Principles of Operation

- *Leader-follower* based algorithm
- Each instance is a Replicated state machine whose states is uniquely determined by a linear *persistent log*
- Leader election proceeds in *monotonically increasing terms* when timeout fires
- Leader orchestrates *safe log replication* to its *followers*

## Non-Core Features

- Supports cluster membership changes w/o service interruption

## Non-Core Features

- Supports cluster membership changes w/o service interruption
- Log compaction for efficient operations

## Java Implementation: Barge

*https://github.com/mgodave/barge !*

- OSS project started by Dave Rusek with contributions from Justin Santa Barbara and yours truly

## Java Implementation: Barge

*https://github.com/mgodave/barge !*

- OSS project started by Dave Rusek with contributions from Justin Santa Barbara and yours truly
- Still very young but usable, provides 2 transport methods: Raw TCP and HTTP

## Java Implementation: Barge

*https://github.com/mgodave/barge !*

- OSS project started by Dave Rusek with contributions from Justin Santa Barbara and yours truly
- Still very young but usable, provides 2 transport methods: Raw TCP and HTTP
- Feature complete w.r.t base protocol but missing *cluster reconfiguration* and *log compaction*

## Java Implementation: Barge

*https://github.com/mgodave/barge !*

- OSS project started by Dave Rusek with contributions from Justin Santa Barbara and yours truly
- Still very young but usable, provides 2 transport methods: Raw TCP and HTTP
- Feature complete w.r.t base protocol but missing *cluster reconfiguration* and *log compaction*
- Friendly (Apache 2.0) License, *Pull Requests* are welcomed

## Demo

## Questions?

## Credits

- Napoléon à Austerlitz

## Credits

- Napoléon à Austerlitz
- Retour vers le futur

## Credits

- Napoléon à Austerlitz
- Retour vers le futur
- L'Académie

## Credits

- Napoléon à Austerlitz
- Retour vers le futur
- L'Académie
- Nancy Lynch at CSAIL