Distributed Algorithms
Introduction

Alberto Montresor

University of Trento, Italy

2016/04/26

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
Contents

1 Getting Started
   - Two generals
   - Common Knowledge
   - Byzantine generals
   - Summary

2 Themes of the course
   - Impossible vs practical
   - Classical vs extreme distributed systems
   - Syllabus

3 Distributed Systems
   - Informal definition
   - Challenges

4 Conclusions
Two generals

A thought experiment:
A potential solution

- General $A$: attack at dawn!
- General $B$: ack, attack at dawn!
- General $A$: ack ack, attack at dawn!
- General $B$: ack ack ack, attack at dawn!
- ... 

**Theorem**

*Under this scenario, there is no solution for the Two Generals Problem*

**Proof.**

By contradiction on the number of messages exchanged.
- Let’s assume (by contradiction) that there is at least one solution to this problem under this scenario.

- If there is one, there could be many.

- If there are many, we can find one which uses the minimum number of messages.

- Take the last message of this protocol: it can be received or it can be lost

- The protocol should work in both cases. So we could avoid sending it at all!

- The resulting protocol uses less messages than the minimum, which is a contradiction
Reality Check

**Atomic Commit**

An atomic commit is an operation in which a set of distinct changes is applied as a single operation.

Example: ATM’s withdrawal

- You withdraw 100 euro from an ATM in Trento
- Your balance should be decreased by 100 euro
A pragmatic solution

Probabilistic protocol

- Assuming messengers are caught independently of each other with probability $p$
  - General $A$:
    - Send $n$ messengers
    - Attacks no matter what
  - General $B$:
    - Attacks if receives at least one messenger

- $p^n$ is the probability that the attack will be uncoordinated

Trade-off:
- We can decrease the probability of failure by increasing $n$...
- But at the additional cost of sending more messengers!
- Without be ever certain that the attack will be coordinated!
Muddy children

- $n$ children go playing
- Children are truthful, perceptive, intelligent
- Mom says: “Don’t get muddy!”
- A bunch (say, $k$) get mud on their forehead
- Daddy comes, looks around, and says “Some of you got a muddy forehead”
- Daddy repeatedly ask: “Do you know whether you have a muddy forehead?”

What happens?

Slides from Lorenzo Alvisi
Muddy children

Theorem

- The first $k - 1$ times Daddy asks, they children says “No”
- The $k$-th time, the $k$ children say “Yes”.

Proof.
By induction on $k$
Theorem

The first \( k - 1 \) times Daddy asks, they children say “No”

The \( k \)-th time, the \( k \) children say “Yes”.

Proof.

By induction on \( k \).

Let \( k = 1 \).

- The first time daddy asks, the child with mud on his forehead say yes.
- Because all the other have no mud, and someone has mud on his forehead, it must be him.

Let \( k > 1 \)

- Every child with mud see \( k - 1 \) children with mud on the forehead.
- If there were \( k - 1 \) children with mud, they would have said yes at the \( (k - 1) \)-th time daddy asks, but they didn’t.
- So there are actually \( k \) children with mud, and they all say yes at the \( k \)-th time daddy asks.
Muddy children

Variation 1

- Suppose $k > 1$
- Every one knows that someone has a dirty forehead before Dad announces it
- Does Dad still need to speak up?

- Let $p = "Someone's forehead is dirty"
- Every one knows $p$
- But, unless the father speak, if $k = 2$ not every one knows that everyone knows $p$!
- Suppose $A$ and $B$ are dirty. Before the father speaks $A$ does not know whether $B$ knows $p$
- If $k = 3$, not every one knows that every one knows that every one knows $p$ ...
Muddy children

Variation 2
... the father took every child aside and told them individually (without others noticing) that someone’s forehead is muddy?

Variation 3
... every child had (unknown to the other children) put a miniature microphone on every other child so they can hear what the father says in private to them?
Two generals, reloaded

- There is an entire logic that formalizes what knowledge participants acquire while running a protocol
- Solving the Two Generals Problem requires common knowledge
  - “everyone knows that everyone knows that everyone knows...”

But:

- Common knowledge cannot be achieved by communicating through unreliable channels
A common knowledge puzzle

Albert and Bernard just become friends with Cheryl, and they want to know when her birthday is. Cheryl gives them a list of 10 dates:

- May 15, 16, 19
- June 17, 18
- July 14, 16
- August 14, 15, 17

Cheryl then tells Albert and Bernard separately the month and the day of her birthday respectively.

- Albert: I don’t know when Cheryl’s birthday is, but I know that Bernard does not know too.
- Bernard: At first I didn’t know when Cheryl’s birthday is, but I know now.
- Albert: Then I also know when Cheryl’s birthday is
Byzantine generals

Scenario

- $n$ Byzantine generals encircling a city
- They must decide whether to attack or retreat!
- Messengers are reliable and synchronous
- Generals may be traitors
- Nobody knows which generals are traitors

Problem specification

The generals require an algorithm to reach an agreement such that (i) all loyal generals decide on the same plan of action and (ii) a small number of traitorous generals cannot cause the loyal generals to adopt different plans.
A potential solution

- Wait for a majority of generals to agree

WRONG! Possible scenario:

- 3 generals
- 1 vote “attack”, 1 vote “retreat”
- 1 traitorous general:
  - sends a vote “attack” to the “attack” general
  - sends a vote “retreat” to the “retreat” general
## The problem is solvable

### Byzantine Fault Tolerance (1982)

- A protocol that given \( n \) processes, can tolerate up to \( t \) traitorous generals with \( n \geq 3t + 1 \)
- Example: 4 generals can tolerate up to 1 “byzantine” general

### Practical Byzantine Fault Tolerance (2002)

- PBFT triggered a renaissance in BFT replication research
- Still going on...
Reality Check

- BFT sponsors in 1982
  - NASA
  - The Ballistic Missile Defense System Command
  - Army Research Office

- Nancy Lynch’s book on Distributed Systems:
  - *The agreement problem is a simplified version of a problem that originally arose in the development of on-board aircraft control systems.*

- BitCoin, a peer-to-peer digital currency system, is based on BFT.

- The 8-hour downtime of Amazon S3 in July 2008 is a well-known example of what happens when you don’t use BFT.
Take-home lessons

- We need to properly *model* our distributed systems
  - Reliable / unreliable communication
  - Benign / malicious processes

- Solutions depend on the underlying model
  - “Approximate” or “probabilistic” solutions
  - “Bounded” solution
  - No solution at all!

- Coordinating multiple processes is difficult
  - Unexpected events: failures, malicious behavior
  - Lack of common knowledge
Theory vs practice

Yogi Berra says:
In theory, theory and practice are the same.
In practice, they are not.

Yogi Berra

- “Always go to other people’s funerals, otherwise they won’t go to yours”
- “I really didn’t say everything I said”
- “Nobody goes there anymore; it’s too crowded”
First theme of the course

Impossible vs practical

Several papers about *impossibility results*:


Yet, many of these problems have *practical solutions*:


A general tension in Computer Science:

The Spanish Flu, 1918-1920

- Pandemic: killed 20M people in a relatively short time, more than World War I
- Virus goal: spread itself as quickly as possible
- Unreliable environment:
  -viruses may be killed
  -transmission may fail
- Transmission network is a complex graph
Beyond technology

Flocks of birds

- **Flying in a flock is good:**
  - probability of being killed by a predator is reduced
- **Flying in a flock is bad:**
  - probability of finding (enough) food is reduced
- **Birds self-organize themselves in a flock**
- **No central authority**
Second theme of the course

Classical vs extreme distributed systems

- Classical distributed system problems include agreement, total order broadcast, atomic commit, replication, etc.
- Extreme distributed system problems include self-* properties, scalability, full decentralization, etc.

Special issue on Springer Computing (Sept. 2012)

Alberto Montresor, Gusz Eiben, Maarten van Steen, editors

“Modern distributed systems may nowadays consist of hundreds of thousands of computers, ranging from high-end powerful machines to low-end resource-constrained wireless devices. We label them as extreme distributed systems, as they push scalability and complexity well beyond traditional scenarios.”
Topics

- Introduction
- Models
- Time, clocks, events
- Reliable broadcast
- Epidemic protocols
- Impossibility of consensus
- Consensus and failure detectors
- Complex networks
- Replication
- P2P

- Epidemics: Beyond dissemination
- Atomic Commitment
- Rollback and Recovery
- Group Communication
- Paxos
- Practical Byzantine Fault Tolerance
- Blockchains
- Byzantine, altruistic, rational model
- Distributed frameworks
What is a distributed system?

Definition (pragmatic)
A collection of independent, autonomous hosts connected through a communication network. Hosts communicate via message passing to achieve some form of cooperation.

Definition (by negation)
A parallel system where there are no:
- shared, global clock
- shared memory
- accurate failure detection
What is a distributed system?

**Definition (optimistic)**

A collection of independent computers that appears to its users as a single coherent system [Andrew Tanenbaum]

**Definition (pessimistic)**

A distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable [Leslie Lamport]
Motivation

Inherent distribution
Applications which require sharing of resources or dissemination of information among geographically distant entities are “natural” distributed systems

Examples:
- Bank with several branches
- Distributed file systems, shared devices, etc.
Motivation

Distribution as an artifact

Distribution may be an artifact of an engineering solution to satisfy some specific requirements such as:

- fault-tolerance
- increased performance / cost ratio
- minimum level of Quality of Service (QoS)

Examples:

- Replicated servers
Independent failures

From distributed systems to failures

- Dependencies between hosts augment the effect of failures
- Distributed systems make the life of developers more difficult
- Availability example:
  - $n$ dependent hosts, probability of failure $p$
  - Availability: $(1 - p)^n$

From failures to distributed systems

- Providing a replicated service via multiple independent processes enables fault tolerance
- Distributed systems should simplify the life of users
  - $n$ independent hosts, probability of failure $p$
  - Availability: $1 - p^n$
**Parallel systems**

**Definition**

Multiple processors that:

- *share memory*
  Communication based on shared memory and synchronization mechanisms

- *share time*
  Access to a common clock

- *share fate*
  No independent failures
The true spirit of distributed systems

- **Parallel systems** exploit determinism to achieve efficiency

- **Distributed systems** address uncertainty created by:
  - multiplicity of control flows
  - absence of shared memory and global time
  - presence of failures
Challenges – blah, blah

- Heterogeneity
- Openness
- Security
- Failure handling
- Transparency
Heterogeneity

Levels:
- Network
- Computing hardware
- Operating systems
- Programming languages
- Multiple implementations

Middleware

Software layer that abstracts from the above providing a uniform computational model
Openness

The degree to which a distributed system can be extended and re-implemented

- Public interfaces
- Standardization

Examples:

- Corba
- Java2EE
- .Net
- Web Services
Security aspects

- **Confidentiality**: avoiding the disclosure of the content of a message to a party distinct from the intended receiver

- **Integrity**: avoiding the corruption of the transmitted contents by a third party

- **Availability**: the capability of providing a service in spite of malicious behavior
Failure handling

- Failure **detection** (e.g., message checksum)
- Failure **masking** (e.g., email retransmission)
- Failure **tolerance** (e.g., replicated servers)
- Failure **recovery** (e.g., log files)

Distributed systems use a mix of these techniques
Transparency

- **Access transparency**: Hides differences in data representation and invocation mechanisms
- **Location transparency**: enables resources to be accessed without knowledge of their location
- **Concurrency transparency**: enables several processes to operate concurrently using shared resources without interference
- **Replication transparency**: enables multiple resource instances to be used to increase reliability and performance without knowledge of the replicas by users
- **Failure transparency**: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- **Migration transparency**: allows the movement of resources and clients within a system without affecting operations
What’s next in the course?

- Distributed system modeling
  - Which kind of failures exist?
  - Which kind of failures we tolerate?

- Problem specification
  - Formal description of the problem

- Algorithms, algorithms, algorithms
  - Pseudo-code descriptions of algorithms

- Proofs
  - Just writing the code is not enough!
  - Sometimes, impossibility proofs

- Reality checks
  - Learn about real systems where these protocols are applied

A. Panconesi. *Coordination and the fall of Eastern Roman Empire.*  
http://www.dsi.uniroma1.it/~asd3/dispense/fall.pdf
Reality Check: Interesting links

- The S3 incident
- Solving the unsolvable
- The rise and fall of Corba
- Notes on Distributed Systems for Young Bloods