Distributive Systems

Martin Kleppmann

April 2, 2023

Contents

1 Introduction

This type of interaction, where code on one node appears to call a function on another node, is called a **Remote Procedure Call**.

What if..

- the service crashes during the function call?
- a message is lost?
- a message is delayed?
- something goes wrong, is it safe to retry?

Today, the most common form of RPC is implemented using JSON data sent over HTTP. A popular set of design principles for such HTTP-based APIs is known as **representational state transfer** or **REST** , and APIs that adhere to these principles are called **RESTful**. These principles include:

- communication is stateless (each request is self-contained and independent from other requests)

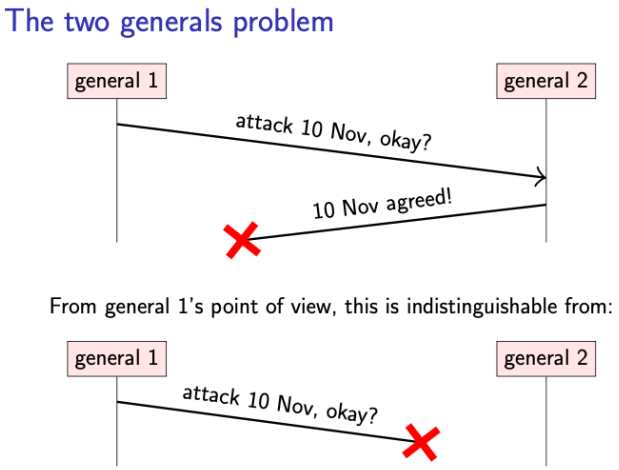
- resources (objects that can be inspected and manipulated) are represented by URLs
- the state of a resource is updated by making a HTTP request with a standard method type, such as `POST` or `PUT`, to the appropriate URL.

2 Models of distributed systems

2.1 The two generals problem

In the two generals problem, we imagine two generals, each leading an army, who want to capture a city. The city's defences are strong, and if only one of the two armies attacks, the army will be defeated. However, if both armies attack at the same time, they will successfully capture the city.

Thus, the two generals need to coordinate their attack plan. This is made difficult by the fact that the two armies are camped some distance apart, and they can only communicate by messenger. The messengers must pass through territory controlled by the city, and so they are sometimes captured. Thus, a message sent by one general may or may not be received by the other general, and the sender does not know whether their message got through, except by receiving an explicit reply from the other party. If a general does not receive any messages, it is impossible to tell whether this is because the other general didn't send any messages, or because all messengers were captured.1



How should the generals decide?

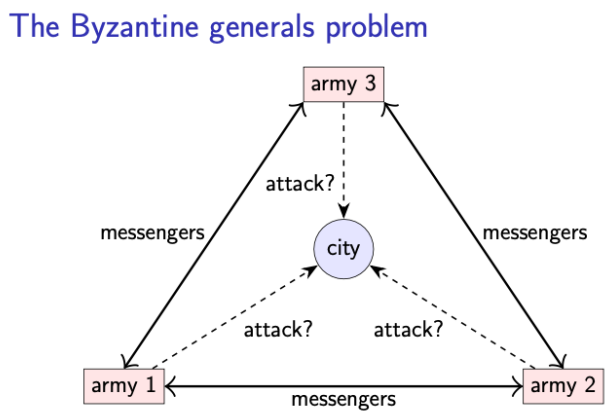
1. General 1 always attacks, even if no response is received?
    - send lots of messengers to increase probability that one will get through
    - if all are captured, general 2 does not know about the the attack, so general 1 loses
2. General 1 only attacks if positive response from general 2 is received?
    - Now general 1 is safe
    - But general 2 knows that general 1 will only attack if general 2's response gets through
    - now general 2 is in the same situation as general 1 in option 1

No common knowledge: the only way of knowing something is to communicate it

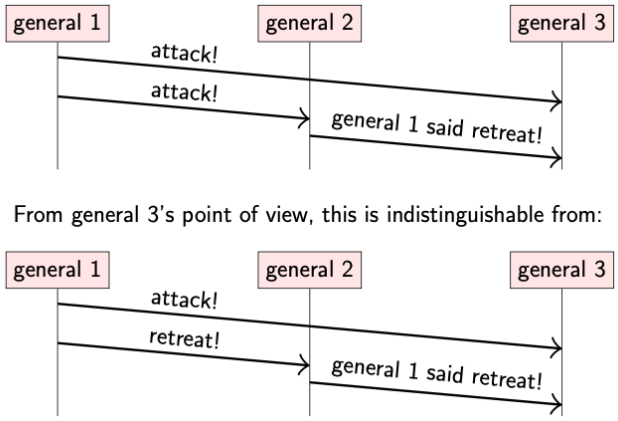
The problem is that no matter how many messages are exchanged, neither general can ever be certain that the other army will also turn up at the same time. A repeated sequence of back-and-forth acknowledgements can build up gradually increasing confidence that the generals are in agreement, but it can be proved that they cannot reach certainty by exchanging any finite number of messages.

2.2 The Byzantine generals problem

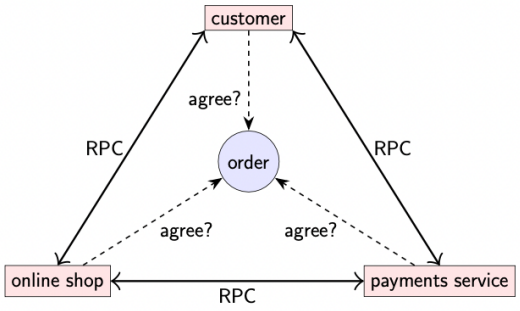
We have armies wanting to capture a city, though in this case there can be three or more. Again generals communicate by messengers, although this time we assume that if a message is sent, it is always delivered correctly.



The challenge in the Byzantine setting is that some generals might be "traitors": that is, they might try to deliberately and maliciously mislead and confuse the other generals. We call the traitors **malicious**, and the others **honest**.



Theorem 2.1. *Need $3f + 1$ generals in total to tolerate f malicious generals*



In distributed systems, some systems explicitly deal with the possibility that some nodes may be controlled by a malicious actor, and such systems are called **Byzantine fault tolerant**.

2.3 Describing nodes and network behaviour

- Two generals problem: a model of networks
- Byzantine generals problem: a model of node behaviour

Capture assumptions in a **system model** consisting of

- network behaviour
- node behaviour
- timing behaviour

Network behaviour: Assume bidirectional **point-to-point** communication between two nodes with one of:

- **Reliable** (perfect) links:
  - A message is received iff it is sent.
  - Message may be reordered.
- **Fair-loss** links:
  - Messages may be lost, duplicated, or reordered
  - If you keep retrying, a message eventually gets through
- **Arbitrary** links:
  - A malicious adversary may interfere with messages

$$Arbitrary \xrightarrow{\text{TLS}} Fair-loss \xrightarrow{\text{retry+dedup}} Reliable$$