

Distributed Systems I

Software Architecture

Brae Webb

March 18, 2024



Mathias Verras
@mathiasverraes

There are only two hard problems
in distributed systems:

2. Exactly-once delivery
1. Guaranteed order of messages
2. Exactly-once delivery

Going forward

Investigating architectures that are *distributed*.

Distributed Systems Series

Distributed I *Reliability* and *scalability* of
stateless systems.

Distributed II *Complexities* of *stateful*
systems.

Distributed III *Hard problems* in distributed
systems.

What are the benefits?

- Improved *reliability*.
- Improved *scalability*.
- Improved *latency*.

What are the drawbacks?

- Increased *complexity*.
- Increased *attack vector*.
- Increased *latency*.
- Introduce *consistency* problems.

§ *Fallacies*

A few reasons for complexity

The Fallacies of *Distributed Computing*.

Fallacy #1

The network is reliable.





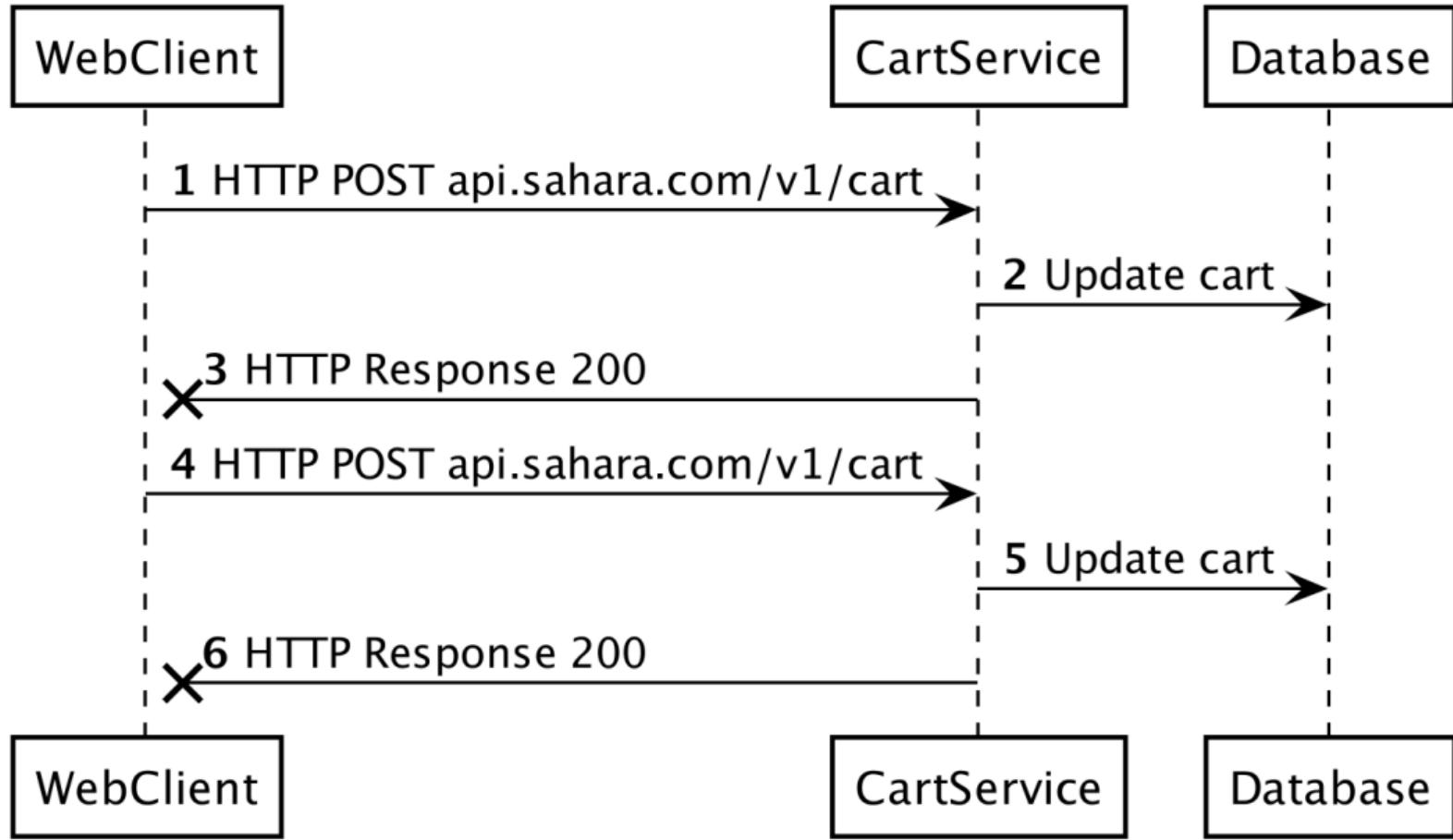




Exponential backoff

```
1  retry = True
2  do:
3      status = service.request()
4
5      if status != SUCCESS:
6          wait(2 ** retries)
7      else:
8          retry = False
9  while (retry and retries < MAX_RETRIES)
```







Fallacy #2

Latency is zero.

Network Statistics

Home to UQ

Home to us-east-1

EC2 to EC2

Network Statistics

Home to UQ 20.025ms

Home to us-east-1

EC2 to EC2

Network Statistics

Home to UQ 20.025ms

Home to us-east-1 249.296ms

EC2 to EC2

Network Statistics

Home to UQ 20.025ms

Home to us-east-1 249.296ms

EC2 to EC2 0.662ms

Fallacy #3

Bandwidth is infinite.

Definition 1. Stamp Coupling

Components which share a composite data structure.

Fallacy #4

The network is secure.





Fallacy #5

The topology never changes.

Fallacy #6

There is only one administrator.

Fallacy #7

Transport cost is zero.

Remember

Distributed systems are *hard*.

The choice to use them should be *well considered*.

When you need to, maybe prove it?



Sept 13-14, 2019
thestrangeloop.com

Or, more realistically,

Use existing algorithms and software.

Distributed Systems Series

Distributed I *Reliability* and *scalability* of
stateless systems.

Distributed II Complexities of stateful systems.

Distributed III Hard problems in distributed
systems.

Stateless vs. Stateful Systems

Stateless Does *not* utilize *persistent data*.

Stateful Does utilize *persistent data*.

Question

What makes software *reliable*?

Definition 2. Reliable Software

Continues to work, even when things go wrong.

Definition 3. Fault

Something goes wrong.

Death, taxes, and computer system failure are all inevitable to some degree.

Plan for the event.

- Howard and LeBlanc

Reliable software is

Fault *tolerant.*

Problem

Individual computers fail *all the time*

Solution

Spread the risk of faults over *multiple computers*, or, *nodes*.

Spreading Risk

If you have software that works with *just one* computer, spreading the software over *two* computers *halves* the risk that your software will fail.

Spreading Risk

If you have software that works with *just one* computer, spreading the software over *two* computers *halves* the risk that your software will fail.

Adding *100* computers reduces the cuts the risk by *100*.



Question

Who has used *auto-scaling*?

Auto-scaling terminology

Auto-scaling group A *collection of instances* managed by auto-scaling.

Auto-scaling terminology

Auto-scaling group A *collection of instances* managed by auto-scaling.

Capacity Amount of instances *currently* in an auto-scaling group.

Auto-scaling terminology

Auto-scaling group A *collection of instances* managed by auto-scaling.

Capacity Amount of instances *currently* in an auto-scaling group.

Desired Capacity Amount of instances *we want to have* in an auto-scaling group.

Auto-scaling terminology

Auto-scaling group A *collection of instances* managed by auto-scaling.

Capacity Amount of instances *currently* in an auto-scaling group.

Desired Capacity Amount of instances *we want to have* in an auto-scaling group.

Scaling Policy How to determine the desired capacity.

Auto-scaling terminology

Auto-scaling group A *collection of instances* managed by auto-scaling.

Capacity Amount of instances *currently* in an auto-scaling group.

Desired Capacity Amount of instances *we want to have* in an auto-scaling group.

Scaling Policy How to determine the desired capacity.

Minimum/Maximum Capacity *Hard limits* on the minimal and maximum amount of instances.

What we really want

Desired Capacity Amount of *healthy* instances
we want to have in an auto-scaling group.

Health check

Mechanism to determine whether an instance is *healthy*.

Auto-scaling

An example















In Summary

Simplicity

Reliability

Scalability

In Summary

Simplicity *Minimal network communication* (compared to other distributed systems), less impacted by fallacies.

Reliability

Scalability

In Summary

Simplicity *Minimal network communication* (compared to other distributed systems), less impacted by fallacies.

Reliability Traffic is spread to various services, still *partially operational* if one goes down. Auto-scaling allows for *basic replication*.

Scalability

In Summary

Simplicity *Minimal network communication* (compared to other distributed systems), less impacted by fallacies.

Reliability Traffic is spread to various services, still *partially operational* if one goes down. Auto-scaling allows for *basic replication*.

Scalability Auto-scaling and load balancing allows *individual services to scale*. However, the *database is a bottle-neck*.