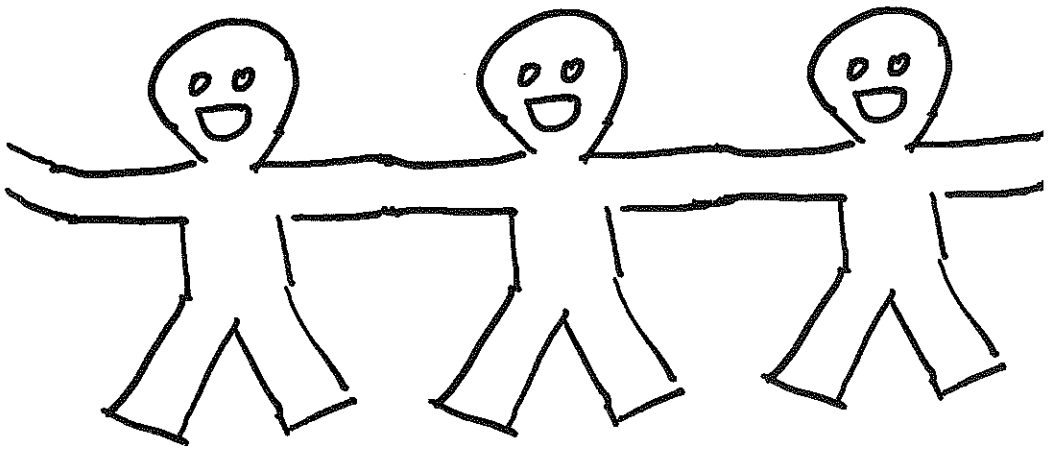
Let's Talk

REPLICATION

in Distributed Systems

Thanks for reading!



by Kasia Sayles

Made for CMPS 128 Spring 2019

Table of Contents

References

Lecture notes from CMPS 128 taught by Professor Lindsey Kuper @lindsey

Extras

https://www.medium.com/baseds

"Explore the basics of ds, every alternate wednesday, for a year."
by Vaidehi Joshi

"How does distributed consensus work?"
by Preethi Kasireddy on medium.com

A Little Intro

Q: Who are you?

Hello, I'm Kasia and I am a senior studying CS.

Q: What made you want to learn about distributed systems?

I'm interested in data and infrastructure engineering.

Q: Why did you choose this topic?

I feel like it's the essence of what people think of as distributed systems.

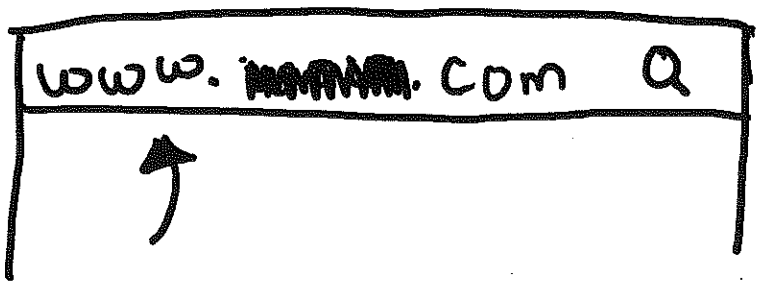
Quick Rundown of Distributed Systems

"I can't get my work done on my computer because somewhere else in the world another computer has failed."

A distributed system is one running on several nodes connected by a network. It is characterized by partial failure.

Partial failure happens when in a group of computers, even if one fails, the rest stay running.

Now, think about when open your browser to visit a website.



Remember this means world wide web, and we could definitely call it a distributed system.

Types of Replication

Replication can be classified in two distinct ways:

1. Active
2. Passive

- Active replication (aka state-machine replication) will execute the operation on every replica server you want.
- Passive replication will have only one replica execute the operation and send updated change to the rest of them.

We also have our commit point again marked as *.

The next request by the client (you) is a read to check the balance. This goes straight to the "tail" server, skipping the "head" and subsequent backups.

~ Drawbacks ~

The write latency[1] is worse than primary-backup replication

1. latency - how long a particular action takes to compile

Replication

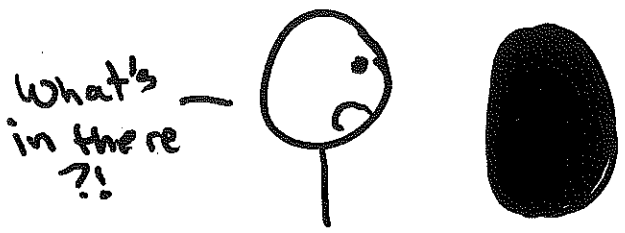
We use replication as a way to make our systems fault tolerant.

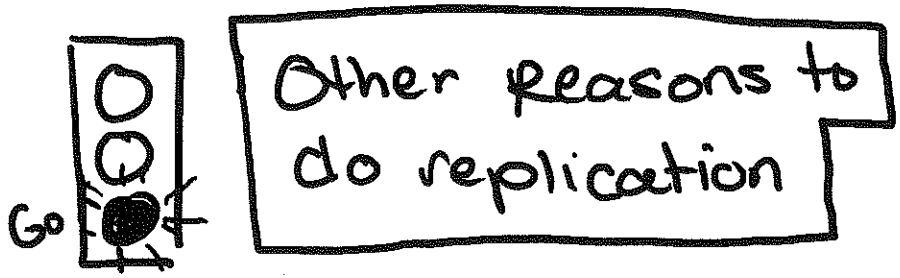
There are four categories of failures that can happen in our systems.

* Crash fault:
  A process fails by halting
* Omission fault:
  Failure to send or recieve

→

* Timing fault:
  Responds too slow/late
* Byzantine fault:
  Anything is possible

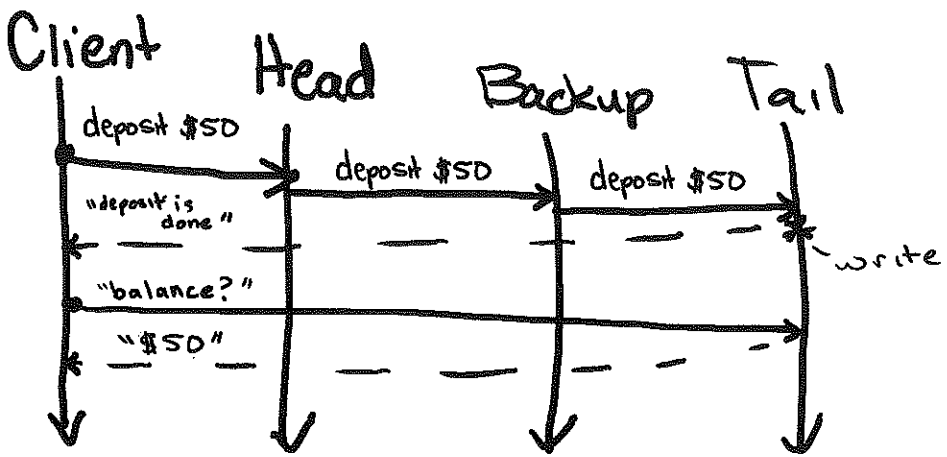




1. The locality advantage

By having multiple machines w/same information, we can access closest one no matter our location just as fast

Chain Replication



Using our mobile bank app example again, as we request to deposit $50 to our account, the "head" server recieves it and passes it to every backup server sequentially ("to its right") until it reaches the "tail" that then tells you deposit successful.

On the diagram, at the *, we call that our commit point. This is when all of our backups have told the primary they recieved the request, so it can officially write.

If I wanted to do a read, i.e check my balance, then primary does not interact with the backups.

~ Drawbacks ~

Everything has to go through primary which can create a bottleneck.

2. The scalability advantage

We can add more machines thus adding more capacity to send/recieve requests

- but what about the cons?

Ah well, it is going to be more expensive and a bit more complicated having to deal with consistency among replicas.

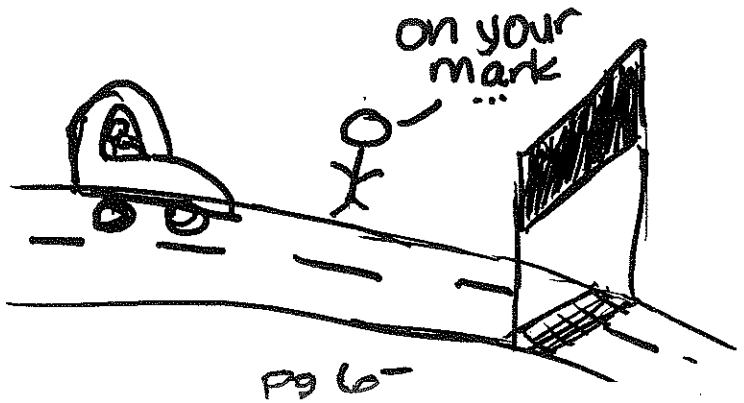
- BUT IT IS STILL WORTH IT -

Implementations of Replication

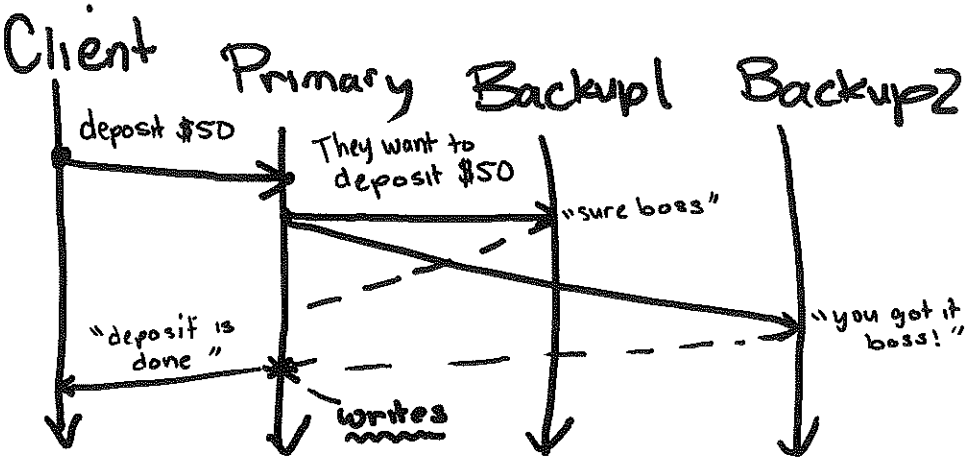
I'm going to show you two ways of doing replication:

- Primary-backup replication
- Chain replication

Each handles reads and writes differently.



Primary-backup



Let's say you are using your mobile bank app. You the client want to do a mobile deposit of $50. When you finish the steps, that primary server to recieve deposit request, will then tell the backups to do the same with their local data. These backups tell the primary they did. Then primary tells you deposit successful.