Distributed Computing I

March 28, 2022

Brae Webb

Presented for the Software Architecture course
at the University of Queensland

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Last Updated on 2022/03/24

Distributed Systems I

Software Architecture

March 28, 2022

Brae Webb

1 Introduction

We have started looking at distributing our applications. Service-based architectures distribute business processes such that each process is deployed on a separate physical machine [1]. The distributed machines act as if they are the one system, so service-based architectures are our first look at distributed systems. Deploying services independently reduces the complexity of deployment, however, it starts to introduce a new range of complexity.

Definition 1. Distributed System

A system with multiple components located on *different machines* that communicate and coordinate actions in order to *appear as a single coherent system* to the end-user.

2 Reliable Software

We want, and in some cases, need, our software to be *reliable*. Our motivation for reliable software ranges from live or death situations all the way to finance motivations. At the extreme end we have radiation therapy machines exposing patients to too much radiation and causing fatalities [2]. On the less extreme end, we have reliability we have outages from Facebook causing $60 million of lost revenue [3]. Regardless of the motivation, we need reliable software. But what does it mean for software to be reliable?

2.1 Fault Tolerance

In an ideal world we would produce fault-proof software, where we define a fault as something going wrong. Unfortunately, we live in an extremely non-ideal world. Faults are a major part of our software world. If we could develop bug-free software, hardware would still fail on us. If we could invent perfect always operational hardware, then we would still be subject to cosmic bit flipping[1].

Instead, we learnt long ago that fault tolerance is the best way to develop reliable software. John von Neumann was one of the first to integrate the notion of building fault tolerance to develop hardware systems in the 1950s [4]. Fault tolerance systems are systems which are designed to be able to recover in the event of a fault. By anticipating faults, rather than putting our heads in the sand, we can develop much more reliable software.

A part of this philosophy is to write defensive software which anticipates the *likely* software logic faults (bugs). The *likely* modifier is important here, if we write paranoid code which has no trust of other systems, our code will become incredibly complex and ironically more bug prone quickly. Instead, use heuristics and experience to anticipate systems which are likely to fail and place fault guards around such systems.

Aside from software logic faults, we have catastrophic software faults and hardware faults. These cause the software or hardware to become unusable. This occurs in practice far often than you might expect. How should we tolerate this type of fault?

[1]https://www.youtube.com/watch?v=AaZ_RSt0KP8

2.2 Distributing Risk

For faults which cause a system to become unusable we can't simply program around it. We need a mechanism for recovering from the fault without relying on the system to work at all as expected. One recovery approach is to duplicate and distribute the original system. If we duplicate our software across two machines then we have halved our risk of the system going completely down. If we replicate our software across thousands of machines then the likelihood of a complete system failure due to a hardware failure is negligible. Google doesn't go down over a hardware failure. We can imagine that the Google servers have many hardware failures in a week but this doesn't cause an outage.

So we have one very important motivation for creating distributed systems, to ensure that our software system is *reliable*.

3 Distributed Architecture

- The simplest distributed architecture = horizontally scaling immutable isolated machines.
- This rarely happens in real world.
- Instead we need need to have systems which talk.
- This communicate introduces all sorts of vulnerabilities.

4 Fallacies of Distributed Computing

4.1 The Network is Reliable

4.2 Latency is Zero

4.3 Bandwidth is Infinite

4.4 The Network is Secure

4.5 The Topology Never Changes

4.6 There is Only One Administrator

4.7 Transport Cost is Zero

4.8 The Network is Homogeneous

5 Modern Scaling

5.1 Health-checks

5.2 Auto-scaling

5.3 Load-balancing

References

[1] R. Thomas, "Service-based architecture," March 2022. https://csse6400.uqcloud.net/handouts/service-based.pdf.

[2] N. Leveson and C. Turner, "An investigation of the therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[3] S. Janardhan, "More details about the October 4 outage." https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/, October 2021.

[4] J. van Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components, automata studies," vol. 34, pp. 43–98, 1956.