Если вы видите что-то необычное, просто сообщите мне.

Distributed systems

Introduction

I wanted a text that would bring together the ideas behind many of the more recent distributed systems - systems such as Amazon's Dynamo, Google's BigTable and MapReduce, Apache's Hadoop and so on.

In this text I've tried to provide a more accessible introduction to distributed systems. To me, that means two things: introducing the key concepts that you will need in order to have a good time reading more serious texts, and providing a narrative that covers things in enough detail that you get a gist of what's going on without getting stuck on details. It's 2013, you've got the Internet, and you can selectively read more about the topics you find most interesting.

In my view, much of distributed programming is about dealing with the implications of two consequences of distribution:

- that information travels at the speed of light
- that independent things fail independently

In other words, that the core of distributed programming is dealing with distance (duh!) and having more than one thing (duh!). These constraints define a space of possible system designs, and my hope is that after reading this you'll have a better sense of how distance, time and consistency models interact.

This text is focused on distributed programming and systems concepts you'll need to understand commercial systems in the data center. It would be madness to attempt to cover everything. You'll learn many key protocols and algorithms (covering, for example, many of the most cited papers in the discipline), including some new exciting ways to look at eventual consistency that haven't still

made it into college textbooks - such as CRDTs and the CALM theorem.

---

1. Basics

The first chapter covers distributed systems at a high level by introducing a number of important terms and concepts. It covers high level goals, such as scalability, availability, performance, latency and fault tolerance; how those are hard to achieve, and how abstractions and models as well as partitioning and replication come into play.

2. Up and down the level of abstraction

The second chapter dives deeper into abstractions and impossibility results. It starts with a Nietzsche quote, and then introduces system models and the many assumptions that are made in a typical system model. It then discusses the CAP theorem and summarizes the FLP impossibility result. It then turns to the implications of the CAP theorem, one of which is that one ought to explore other consistency models. A number of consistency models are then discussed.

3. Time and order

A big part of understanding distributed systems is about understanding time and order. To the extent that we fail to understand and model time, our systems will fail. The third chapter discusses time and order, and clocks as well as the various uses of time, order and clocks (such as vector clocks and failure detectors).

4. Replication: preventing divergence

The fourth chapter introduces the replication problem, and the two basic ways in which it can be performed. It turns out that most of the relevant characteristics can be discussed with just this simple characterization. Then, replication methods for maintaining single-copy consistency are discussed from the least fault tolerant (2PC) to Paxos.

5. Replication: accepting divergence

The fifth chapter discussed replication with weak consistency guarantees. It introduces a basic reconciliation scenario, where partitioned replicas attempt to reach agreement. It then discusses Amazon's Dynamo as an example of a system design with weak consistency guarantees. Finally, two perspectives on disorderly programming are discussed: CRDTs and the CALM theorem.

Appendix

The appendix covers recommendations for further reading.

Revision #2
Created 2022-09-06 06:53:07 UTC by gasick
Updated 2022-09-06 06:58:07 UTC by gasick