ILDER

@保存到为知笔记 我要踏入分布式[doge]// @网路冷眼 :中文翻译，看参考这里->面向工程师的分布式系统理论 http://t.cn/RqdDIzE

@网路冷眼

【Distributed systems theory for the distributed systems engineer】 http://t.cn/RPNfF9k 面向工程师的分布式系统理论，列出了分布式系统许多理论以及开发的经典应用。干货十足！[赞]

Distributed systems theory for the distributed systems engineer

Gwen Shapira, SA superstar and now full-time engineer at Cloudera, asked a question on Twitter that got me thinking.

My response of old might have been "well, here's the FLP paper, and here's the Paxos paper, and here's the Byzantine generals paper...", and I'd have prescribed a laundry list of primary source material which would have taken at least six months to get through if you rushed. But I've come to thinking that recommending a ton of theoretical papers is often precisely the wrong way to go about learning distributed systems theory (unless you are in a PhD program). Papers are usually deep, usually complex, and require both serious study, and usually *significant experience* to glean their important contributions and to place them in context. What good is requiring that level of expertise of engineers?

And yet, unfortunately, there's a paucity of good 'bridge' material that summarises, distills and contextualises the important results and ideas in distributed systems theory; particularly material that does so without condescending. Considering that gap lead me to another interesting question:

What distributed systems theory should a distributed systems engineer know?

A little theory is, in this case, not such a dangerous thing. So I tried to come up with a list of what I consider the basic concepts that are applicable to my every-day job as a distributed systems engineer; what I consider 'table stakes' for distributed systems engineers competent enough to design a new system. Let me know what you think I missed!

First steps

These four readings do a pretty good job of explaining what about building distributed systems is challenging. Collectively they outline a set of abstract but technical difficulties that the distributed systems engineer has to overcome, and set the stage for the more detailed investigation in later sections

Distributed Systems for Fun and Profit is a short book which tries to cover some of the basic issues in distributed systems including the role of time and different strategies for replication.

Notes on distributed systems for young bloods – not theory, but a good practical counterbalance to keep the rest of your reading grounded.

A Note on Distributed Systems – a classic paper on why you can't just pretend all remote interactions are like local objects.

The fallacies of distributed computing – 8 fallacies of distributed computing that set the stage for the kinds of things system designers forget.

Failure and Time

Many difficulties that the distributed systems engineer faces can be blamed on two underlying causes:

1. processes may fail
2. there is no good way to tell that they have done so

There is a very deep relationship between what, if anything, processes share about their knowledge of *time*, what failure scenarios are possible to detect, and what algorithms and primitives may be correctly implemented. Most of the time, we assume that two different nodes have absolutely no shared knowledge of what time it is, or how quickly time passes.

You should know:

* The (partial) hierarchy of failure modes: crash stop -> omission -> Byzantine. You should understand that what is possible at the top of the hierarchy must be possible at lower levels, and what is impossible at lower levels must be impossible at higher levels.

* How you decide whether an event happened before another event in the absence of any shared clock. This means Lamport clocks and their generalisation to Vector clocks, but also see the Dynamo paper.

* How big an impact the possibility of even a single failure can actually have on our ability to implement correct distributed systems (see my notes on the FLP result below).

* Different models of time: synchronous, partially synchronous and asynchronous (links coming, when I find a good reference).

The basic tension of fault tolerance

A system that tolerates some faults without degrading must be able to act as though those faults had not occurred. This means usually that parts of the system must do work redundantly, but doing more work than is absolutely necessary typically carries a cost both in performance and resource consumption. This is the basic tension of adding fault tolerance to a system.

You should know:

* The quorum technique for ensuring single-copy serialisability. See Skeen's original paper, but perhaps better is Wikipedia's entry.

* About 2-phase-commit, 3-phase-commit and Paxos, and why they have different fault-tolerance properties.

* How eventual consistency, and other techniques, seek to avoid this tension at the cost of weaker guarantees about system behaviour. The Dynamo paper is a great place to start, but also Pat Helland's classic Life Beyond Transactions is a must-read.

Basic primitives

There are few agreed-upon basic building blocks in distributed systems, but more are beginning to emerge. You should know what the following problems are, and where to find a solution for them:

* Leader election (e.g. the Bully algorithm)

* Consistent snapshotting (e.g. this classic paper from Chandy and Lamport)

* Consensus (see the blog posts on 2PC and Paxos above)

* Distributed state machine replication (Wikipedia is ok, Lampson's paper is canonical but dry).

Fundamental Results

Some facts just need to be internalised. There are more than this, naturally, but here's a flavour:

- You can't implement consistent storage and respond to all requests if you might drop messages between processes. This is the CAP theorem.
- Consensus is impossible to implement in such a way that it both a) is always correct and b) always terminates if even one machine might fail in an asynchronous system with crash-* stop failures (the FLP result). The first slides – before the proof gets going – of my Papers We Love SF talk do a reasonable job of explaining the result, I hope. *Suggestion: there's no real need to understand the proof.*
- Consensus is impossible to solve in fewer than 2 rounds of messages in general

Real systems

The most important exercise to repeat is to read descriptions of new, real systems, and to critique their design decisions. Do this over and over again. Some suggestions:

Google:

GFS, Spanner, F1, Chubby, BigTable, MillWheel, Omega, Dapper. Paxos Made Live, The Tail At Scale.

Not Google:

Dryad, Cassandra, Ceph, RAMCloud, HyperDex, PNUTS

Postscript

If you tame all the concepts and techniques on this list, I'd like to talk to you about engineering positions working with the menagerie of distributed systems we curate at Cloudera.

PUBLISHED: August 9, 2014
FILED UNDER: Distributed systems