Designing Data-Intensive Applications Notes

Contents

How to Use This Template

- Create a new **Chapter** section for each audiobook chapter.
- Capture key ideas, notable quotes, and takeaways in the provided subsections.
- Add action items or open questions you want to revisit.

1 Chapter # – Title

1.1 General

- Redis, Apache Kafka (boundaries blurred)
- Single tool can't any longer meet data needs
- work is broken down into tasks into multiple tools
- mem-cached, elastic search, solar – application code keeps all this synced
- API hides implementation details from clients
- how do you ensure data remains correct and complete?
- how do you scale?
- what API is best?
- how do you have an issue but the client can still be fine
- Book considers 3 concerns: reliability (toyota style), scalability, maintainibility (chnaging people should be able to work on the system productively)
- following chapters: architectures and algos important to achieve these
- Reliability:
- faults and fault tolerant systems in reliability
- fault is one component of a system deviating from its spec
- failure: system as a whole stops providing the service to the user
- fault tolerance mechanisms are important
- Netflix Chaos Monkey: a revolutionary tool that randomly disables virtual machines and services in its production environment to test system resilience

- hard disk MTTF (mean time to failure) is 10-50 years, 10000 disks, on average 1 disk to die per day
- mo machines mo problems
- Virutal Machine instances stop running without warning because it is sometimes needed somewhere else
- Software Errors:
- independent events generally, a large number don't fail all at once
- automated testing: config changes -> possible issues -> slow rollout so only a few users are affected if at all
- telemetry: track and monitor failures
- discussing scalablility is trying to figure out how to cope with growth in whichever way you are growing (current load, and then incr. load)
- twitter example (Nov 2012 data): 4600 req./s, views 300,000 req/s, tweet volume + fan out. Two ways of implementing follower followee. Global collection is an option and then you merge those where follower and followee (original approach). do more work at write time and less at read time so just write the tweets on their home timelines (new approach). "Fan out load". Twitter now hybrids b/w the 2. Celebs are fetched separately like in Approach 1.
- System performance based on load parameters
- performance numbers: Hadoop – care about "Throughput"
- median is good, mean is whatever
- how bad are outliers? check percentiles like 95th, 99th, 99.5th
- response time requirements at amazon are defined using p999 (99.9th percentile). (slowest one in 1000 requests) They focus on the most valuable customers. The website should be lightning fast for that top 0.1% of person
- 99.99th percentile is too hard and too expensive so you do the next best possible thing

- Percentiles in practice: End user request needs to still wait for the slowest of the parallel processes. So a small percentage of slow processed requests can slow down the process a lot so you need to focus on trying to make the slowest ones faster kinda always. good, better, best. never let it rest. till your good gets better. and your better gets best. bears on 3. 1 2 3 BEARS
- Tail Latency Amplification is what that previous item is called
- Forward Decay, T-Digest, HDR Histogram (algo examples that can calculate a good approx. of percentiles at min cpu and memory cost, the other way would be too expensive – trying to calculate all request times in a minute and sorting them by time)
- Right way of aggregating response time data is histograms
- approaches for coping with load:
- shared nothing architecture – high end machines get expensive so you can't scale out sometimes so you have one machine
- no such thing as a one size fits all scaling architecure: volume of reads/writes, the volume of data to store, data complexity, response time requirements, access patters, usually a combination of them all
- a system designed to handle a 100k 1kB requests in a minute is different that one designed to handle 3 2gB requests in a minute.
- load parameters <8 early stage startups need to be able to iterate on product things more than they need to focus on having a "perfectly scalable" architecture
- don't create legacy software
- 3 design principles: operability, simplicity (not UI but engineers should be able to figure ts out), adaptability (no easy solutions but you create systems that try to achieve this)
- ops teams »»
-

Key Ideas

-
-
-

Architecture / Systems Mentioned

-

Concepts and Definitions

- **Term**: Definition or explanation.

Diagrams / Mental Models

-

Notable Quotes

- “”

Questions / Confusions

-

Practical Takeaways

-

Action Items

-