Readings - Virtillo Ch 1

> [!PDF|] [[Roberto Vitillo - Understanding Distributed Systems - 2nd Edition (2022).pdf#page=19&selection=9,17,11,17|p.1]] > a distributed system is a group of nodes that cooperate by exchanging messages over communication links to achieve some task

- necessities: network, fault-tolerance, giant servers, super-fast streaming
- **communication:** leaky abstractions - must still worry about networks in case of tcp failure
- **coordination:** in the face of failures? –> TCP handshake
  - **ports-and-adapters architecture:** handles API calls by others and exposes own API
- **scalability:**
  - throughput (# requests done/sec)
  - response time (# sec btwn response and request)
  - capacity: max load where most operations fail
  - through *scaling up* (more expensive tech) OR *scaling out* (rent more AWS machines w/ elastic cloud)
- resiliency:
  - faults? cascading? non-deterministic?
  - **availability** = uptime / (uptime + downtime). ideally >=99.9%
- maintainability:
  - unit, integration, and end-to-end tasks before any change
  - safe releasing of changes? backwards compatibility?
  - IT, manual fixes to system health w/out code changes

Defining “Distributed System”

1. conceptually: set of “servers” that cooperate over a network, provide service?
2. group of interconnected (over network) computers (servers, nodes, etc.) coop & agree (protocols) to provide service
3. set of interacting components (msg standard, RPC, IO buffers protobuffs in Google) with specific I/O

- run-time POV: group of software processes running IPC i.e.. HTTP
- implementation POV: group of services that communicate via APIs
- interface: talking btwn nodes (abstraction not expose within nodes)
- **network vs bus**:
  - bus: proprietary internal communication i.e. particular cpu/mobo
  - network: generalized
- coordination: common language/protocol agreement among nodes
  - if “set x=3” so propogate among nodes? read x while not done propogating x?
- transparency: abstraction, just do some task
- distributed sys vs airlines (fault-tolerant) because all local bus (proprietary radio) ### Examples of implicit/explicit
- backends for popular apps - implicit uses
- DNS - implicit
- data processing (spark, flink) - explicit
- ML train (explicit), chatGPT (implicit)
- blockchain - distributed ledger
- online service games ## subclass types
- high performance (sims, data processing)
- info/database (idnexigin)
- pervasive ubiquitous sys (IoT sensor network, security network sys i.e. cams)
- multiple birdhouse cams, all connect how?
  - all connect to one master (pt failure, aka SPOF)
  - all nodes caching (sensors shouldn’t handle data)
  - gossip (all communicate via range, closest, until everyone gets same data) ## anatomy

- nodes - iPad, phone, robot, car, Alexa, Arduino.
  - all susceptible to failures. (mean time between failures)
- server
  - serves
  - balances workload
- databases:
  - tweets/videos into data sink (sucks data) –> filters to diff destination by type which may be more nested distrib. system
  - sharded architecture - not CDN aka store every copy on every country
- network
  - controller at app-lvl talks with phone to keep you connected when walking btwn campus buildings
  - latency: time from node j to i sending
  - bandwidth: volume of data per sec
- storage
  - ssds fail spectacularly
  - hdds slow cheap
  - file system partition distrib? tailor paid services to use case
- types
  - computation, communication, storagi

question

- one failure?
- multiple failures? parallel? cascade? non-deterministic environment? long after deployment?
- order maintained? concurrency
- correctness?
- resources? elascticity/scalability/performance
- bottleneck even w/ doubled nodes?