01.Introduction

Most application nowadays are distributed. Example is email service as their applications use millions of email servers that cooperates to guarantee the working status of the application.
Google tops the list of most visited websites in the world.

Google

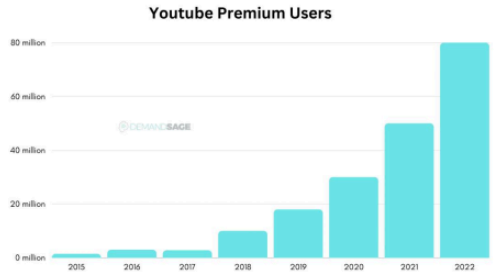
The distributed system goes from the browser to google. There isn't a single machine that can be called Google by the start as it was thought as a cluster of data centres around the world. Nowadays there are two millions of servers that run as one.
The initial idea is to use on the shelf hardware. Strongly software only. Invest billions of hardware but the intelligence is in the software. It can run standard hardware that can break and the software doesn't care(this is why the mind is spent on the SW).
Typical the Google data centre are near rivers/lakes and need a large portion of terrain as they require water to cool all the servers and land to build all the infrastructures.

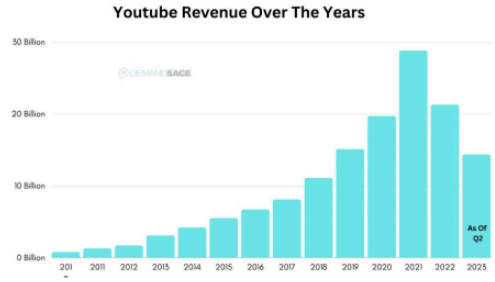
The search engine

- Google processes over 8.5B searches per day
  - ~100k searches per second
  - 63% of this traffic from mobile devices
- ~15% of queries have never been seen before

YouTube (source: youtube statistics)

- Over 1B hours of video are watched each day on YouTube
- 720,000 hours of video are uploaded to YouTube every day
  - 500 hours per minute
- According to Nielsen, YouTube reaches more US adults ages 18-34 than any cable network





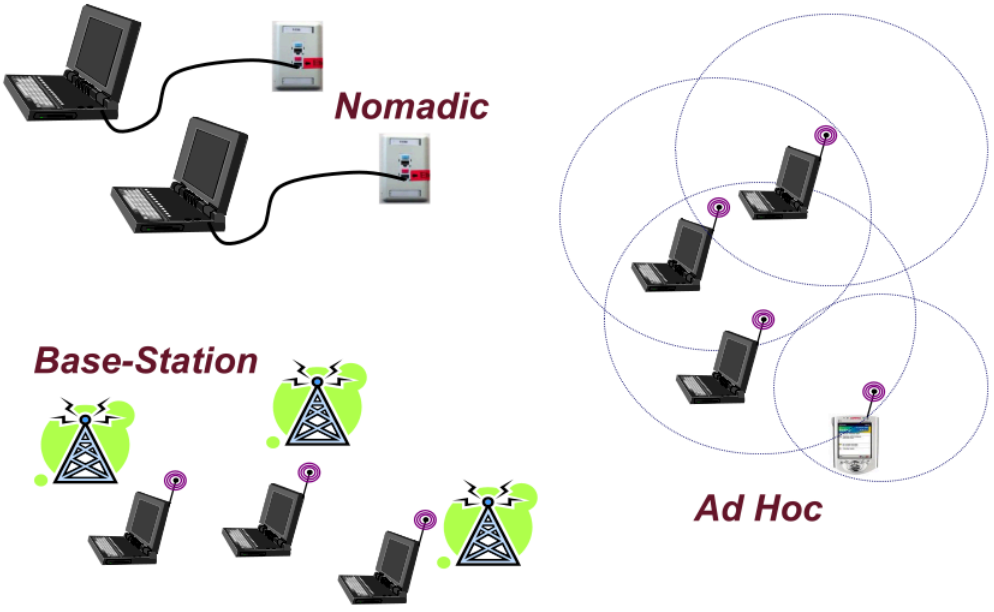
But not all distributed systems are on or related to the internet and the web:

- Intranets (banking and corporate systems, scientific computing, server farms, grids, clouds,...)
- Home networks (home entertainment, multimedia sharing, UPnP)
- PANs (cellular phones, PDAs, wireless headsets, GPS radio receivers, healthcare monitoring devices)

- Wireless Sensor Networks

Pervasiveness: An Internet of Things

Complexity can also derives from the battery some devices use to be mobile. Their functioning is lower from not mobile devices as they are related to the duration of their battery(New level of complexity).
The mobile devices need also to consider the distance and mobility of the devices(Switching on and off, moving the stations meanwhile the service is running).
I want to work while moving, this need to be considered.

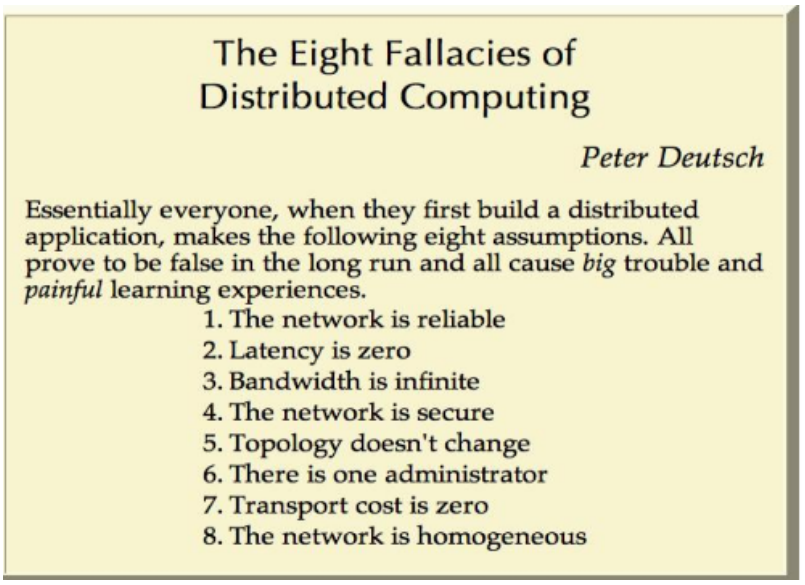


Why we distribute?

- Economics
  - Originally mainframes consolidated processing
  - Price/performance ratio in favor of distribution
- Incremental growth and load balancing
  - Easier to evolve the system and use its resources uniformly
- Inherent distribution
  - e.g., banks, reservation services, distributed games, CSCW, mobile apps
- Reliability
  - Failure does not bring down the whole system (important for mission critical applications)

  However, distributed systems are considerably more difficult to get right!!!
  Distributed systems are more complex than standard/centralised ones. Building

them is harder, parallelising thing increase the complexity. Building them correct is much harder.



1. I can use TCP/IP but it only think about congestion avoidance. It doesn't know if the connection is destroyed. For wireless it is stupid as if you lost a package the only answer is it is lost and so you have to resend it. There is no way to distinguish it if the problem is on the network
2. Causality. Usually I assume that there will be a response to the packages and usually I assume my packages arrive at the same time to all the destinations but packages can take different time to arrive to different locations.
3. It determines delay in transition of the packages.
4. The network can be altered
5. Routing can change and link speed can change. A distributed system is based on the fact that different components can communicate
6. Administrative things results in problems. The design can lead to disrupt of service. If the administrator of networks cannot reach an accord the network cannot use all the routes that are present and this can result in disrupt of service
7. Each packet in the network has a cost
8. The latency is not equal in space and time as bandwidth changes

What is a distributed system?

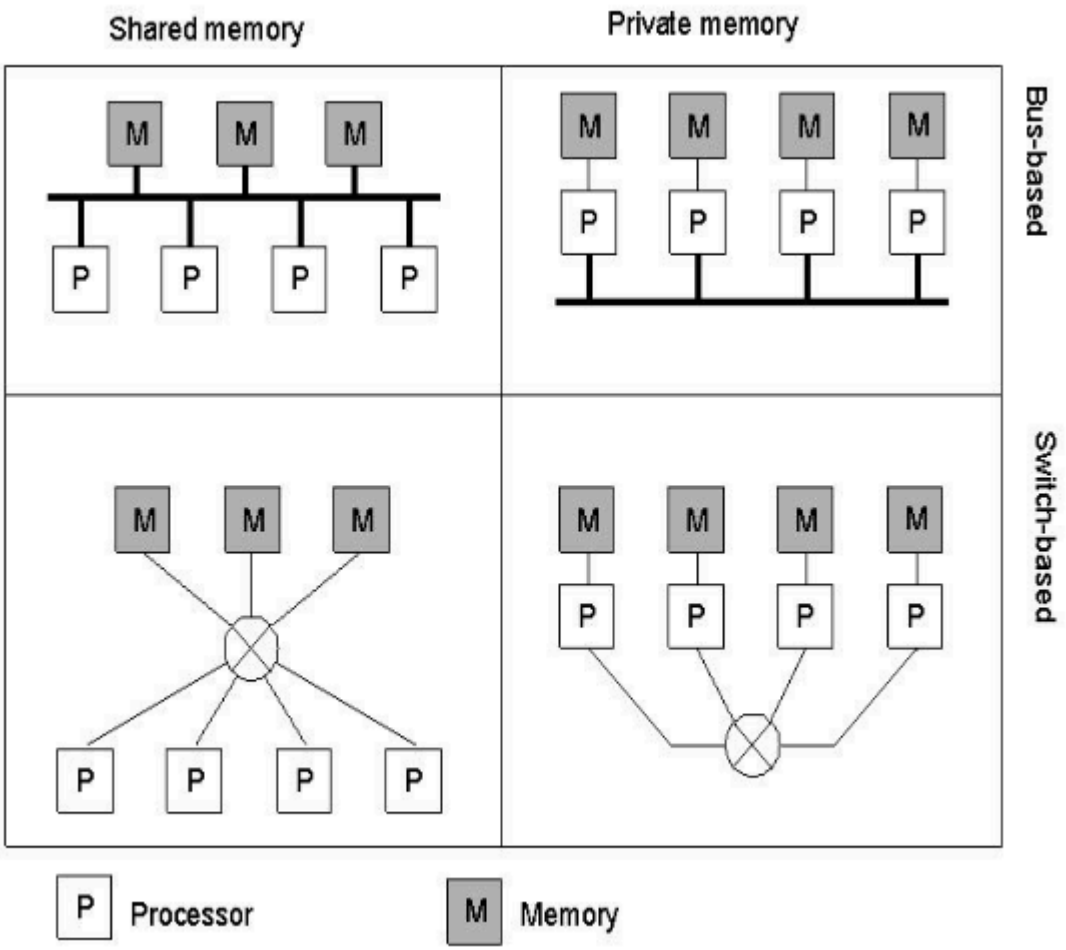
> *"A collection of independent computers that appears to its users as a single coherent system"*
> -- A.S. Tanenbaum, M. van Steen

> *"One in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages"*
> *-- G. Coulouris, J. Dollimore, T. Kindberg*

> *"One on which I cannot get any work done because some machine I have never heard of has crashed"*
> *-- L. Lamport*

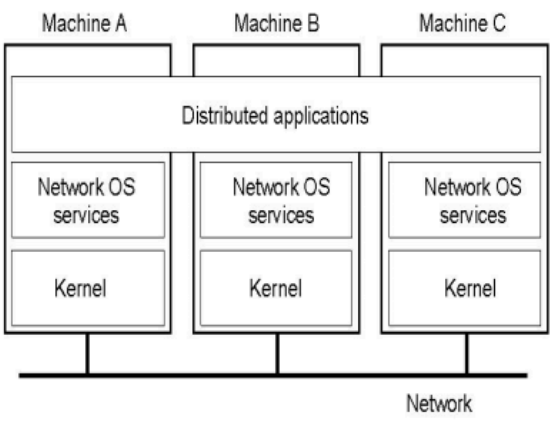
Independent computer aren't single block. They don't share memory or clock.
The only form of communication is done by passing messages and not reading memory.
There is a single service that is offered by multiple machines.

Parallel computing vs. distributed computing



A parallel system use a bus to distribute the data between the different processors, it has a central memory/all memory cell can be accessed by every processor. A distributed system need a private memory for every processor.

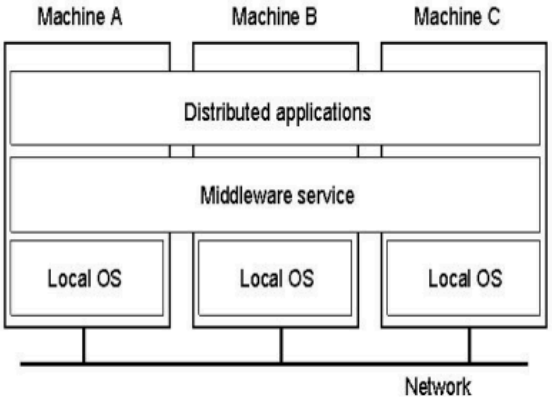
Middleware



Network/OS based distributed system

- Middleware provides horizontal services to help building distributed applications
- It also masks platform differences

Middleware based distributed system



You can design the application to run on top of all the services and don't hide the fact that there are multiple machines. Another routing where you develop on top o libraries(middleware) where it tries to hide the difference machine as much as possible to the point where they are seen as a single machine. The middleware then take care of masking all the different machine. Usually we use the first mode(also for the project). The typical difference can be an architectural difference.

Defining features:

- Concurrency: In centralized systems, concurrency is a design choice. In distributed systems, it is a fact of life to be dealt with: computers are always (co)-operating at the same time
- Absence of a global clock: In centralized systems, the computer’s physical clock can be used for the purpose of synchronization. In distributed systems, clocks are many and not necessarily synched. The synchronisation protocol used in programming languages uses the internal clock of the machines(cannot happen in a distributed system)

- Independent (and partial) failures: Centralized systems typically fail completely. Distributed systems fail only partially, and often due to communication. Each component can fail independently leaving the others still running. Hard/impossible to detect failures. Moreover, recovery is made more complicated by the fact that the application state is itself distributed. Since the failures are partial the users cannot accept that the system collapse, as they should. The programmer need to mask the failures

Some challenges:

- Heterogeneity: Of hosts, platforms, networks, protocols, languages, …
- Openness: A distributed system should foster interoperability through standard access rules. Protocols and interfaces play a key role.
- Security: The ease of attaching a node to the system can be exploited maliciously
- Scalability: Use decentralized algorithms to allow the system to grow with the lowest possible impact on performance. It means that the performance can go down when you increase the level of request to the systems BUT they have to go down smoothly.
- Failure handling: Hosts can fail, links are unreliable, the two are usually undistinguishable, global state complicates matters. Detecting, masking, tolerating, recovery from failures. Failures need to be partials.
- Concurrency: Synchronization without a physical clock and without shared memory. It is hard to synchronise without a single clock.
- Transparency: Hide the most to simplify the life of programmers/users. The programmer need to know where things are running to not loose control of the process.