Distributed systems

Example, MMOs:

* Huge numbers of players all over the globe
* Need to maintain a consistent view of the shared world
* Need fast response times
* Players don’t need to know where their competitors are or where the data is stored

Example, Smart cities:

* Heterogeneity
* Need for real-time data
* Need to support extensions of the system
* City planners need to have access to all the data at once, without having to deal with different hardware, or where sensors are located

Example, Distribution in space:

* Heterogeneity
* Deployment issues
* Very high network latency
* High cost of replacement

Example: cloud computing systems

* Need to store large amount of data
  + And to provide high cpu power
  + Through many different components (virtual or not)
* Need to provide strong guarantees
  + Security
  + Response times
  + Resistance to failure etc
* Examples
  + Aws
  + Microsoft azure
  + IBM cloud
  + Google cloud

Big data storage solutions

* Data managed by companies like Facebook and Google cannot be stored on single machines
* They are stored on clusters (I.e. interconnected computers acting as a single large hard disk)
* They need to provide:
  + High read/write rates
  + Resistance to failure

Definitions:

Distributed systems:

* A distributed system is one on which components located at networked computers communicate and coordinate their actions only by passing messages
* This definitions leads to the following especially significant characteristics of distributed systems:
  + Concurrency of components
  + Lack of a global clock
  + Independent failures of components

Another definition:

* A distributed system is a collection of independent computers that appear to its users as a single coherent system

Third definition:

* A distributed system is a collection of autonomous computers that are connected a network and middleware. Users of a distributed system perceive the system as a single integrated computing facility.
* A middleware is a distribution software that enables computers to coordinate thei activities and to share the resources of the system.

Data centres:

* Most common running environment for distributed systems
* Eg. Used to store Facebook/Google’s data
* Components
  + Low end servers
  + Mounted within a rack
  + Interconnected using Ethernet switches
* Consists of commodity hardware, mostly
* Often similar hardware, but not identical (low heterogeneity)

Common data center interconnect topology:

* Three tiered design
* Supports ~25k hosts
* Directly connected hosts can communicate with each other with the full speed of their network interface

Other hardware-related aspects

* Uninterruptible power supply (UPS)
* Cooling and ventilation systems
* Protection: fire, physical security
* Management centre (for people to monitor operations)

Operating system layer

* Generally, specialised version of mainstream operating systems (Windows, linux)
* If a data-centre is hosting dedicated servers, it might be up to the tenants
* Examples of specialised OS:
  + Facebook
  + Google
  + Red hat

The middleware layer allows to support heterogenous computers, while offering a single-system view. Some authors identify middleware and distributed systems.

The middleware layer is necessary to address the different challenges posed by distribution.

* Distributed file systems, e.g
  + GFS, Colossus, Hadoop distributed file systems
* Messaging middleware, e.g.
  + Java messaging service, RabbitMQ, Apache ActiveMQ
* Database middleware, e.g.
  + Google spanner, Google BigTable, Apache Cassandra
* Object Middleware, e.g.
  + RMI, CORBA, etc.

Different types of middleware depending on the aims of the data-centre:

* Cloud and web hosting, IsaaS, etc.
* Data-Storage and data-processing

The middleware layer is key to hiding the complexity of a distributed system for it’s users. It needs to be tailored to the specifics of the distributed system.

Challenges of distribution:

* Heterogeneity of hardware, operating systems, programming languages, etc.
* Openness: can the system be extended or re-implemented in different ways?
  + Are the specifications of the system/middleware open-source?
  + Is there a risk of vendor lock-in?
* Security: how to protect the value of data processed across a system?
  + Generally, at the application-level, but supported by middleware
  + Depends on threat, e.g. denial of service vs privacy breaches
* Scalability: will the system remain effective when there is an increase in the number of resources/users
  + How easy is it to add new resources (CPU, bandwidth, memory)
  + Communication bottlenecks?
* Failure handling: can the system survive when some of its components fail?
  + Datacentres: between 1.2 and 16 forced restarts per server, per year
  + Peer-to-peer: users might join/leave at will
  + IoT: devices might run out of battery, lose signal, etc.
  + Example:
    - 5000 servers (machines) in one data-centre
    - Average 2 failures per server per year (forced-restart)
    - (5000x2)/365 = 27 restarts per day
* Concurrency can the system allow several clients to access the same resource at the same time?
  + How do we allow concurrent access while guaranteeing consistency
* Transparency: how to hide the complexity of the system to it’s users? The user should perceive the system as a whole rather than a collection of components.
  + The users are often either programmers of applications
  + We need to provide suitable abstractions
  + Examples of aspects that may be hidden from the user
    - Access: hide differences in how resources are accessed
    - Location: hide where a resource is located
    - Replication: that a resource is replicated
    - Failure: hide the failure and recovery of a resource
    - Concurrency: hide that a resource may be shared by several users
* Quality of service: how can the system meet specific guarantees wrt. Reliability, security, and performance?
  + When cloud tenants pay AWS, they expect certain levels of performance (e.g. start up latency, low failure rate, etc)
  + Some architecture provide adaptive software, that provide different quality of service depending on the circumstances (e.g. low battery)

Entities and communication

1. What are the entities that are communicating in the system?
   1. Java objects, web services, processes, sensors, computers
2. How do these entities communicate?
   1. Inter-process communication via a protocol (e.g. socket programming)
   2. Remote invocations software via a middleware
   3. Group communication, via a middleware
3. What roles and responsibilities do entities have in the overall architecture?
   1. Client-server, peer-to-peer
4. How is each entity mapped to the physical architecture?
   1. One service to multiple servers
   2. VM placements
   3. Mirroring, caching, etc
   4. Mobile code

Example: data centres:

* What are the entities?
* How do they communicate?
* What roles/responsibilities do they have?
* How are they mapped to the physical architecture?

System architecture

* Centralised architecture(e.g. client-server)
  + Server: entity that implement a specific service (e.g. file server)
  + Client: entity that requests a service from a server (by sending a request)
* Decentralised architecture (e.g. peer-to-peer)
  + Clients or servers are split up into logically equivalent parts
  + But each part is operating on it’s own share of the data set
  + Entities may act as client and server at the same time
* Hybrid architecture
  + Combination of the other 2
  + Example; BitTorrent uses a global directory to bootstrap a file exchange through a peer to peer network